Feature Selection in a French MEMD Language Model

George Foster

January 20, 2000

1 Introduction

Language models which estimate $p(w|\mathbf{h})$, the probability that a word w will follow a sequence $\mathbf{h}$ of previous words, have many applications in natural language processing, including speech recognition, machine translation, language identification, character recognition, and text compression. From a machine learning perspective, language modeling is a challenging problem because vocabularies are typically on the order of 10,000 or 100,000 words, training corpora can contain tens or hundreds of millions of words or more, and for any history $\mathbf{h}$ the true distribution $p(w|\mathbf{h})$ is extremely unlikely to be identical to the distribution for any other history. From a natural language perspective, language modeling is among the problems best suited to ML techniques, because there is no need for expensive and subjective hand-labelling of examples,[1] and because a simple, automatic, and relatively application-independent evaluation procedure exists.

The most widely-used language model is the *trigram* [8], which is based on the assumption that only the last two words in a history are significant when predicting what will come next: $p(w|\mathbf{h}) \approx p(w|w'', w')$, where $\mathbf{h}$ ends with w'', w'. Advantages of the trigram are that it is conceptually simple, very efficient to train and run, and gives surprisingly good results. Its main disadvantage is that it relies on a very large number of parameters. This complicates implementation and makes maximum likelihood estimates from relative frequencies unreliable (for example, any trigram not observed in the training corpus will be assigned a probability of zero, whereas intuitively even the largest training corpora will not include a large number of rare but linguistically valid trigrams). An effective and popular solution to the

[1] Although some human input is nevertheless required in establishing heuristics to identify words and sentences in running text, and in specifying a mapping from a potentially infinite set of "words" (including items like numbers, dates, proper nouns, etc) encountered in text to a finite vocabulary, which most language models assume.

latter problem is to smooth raw ML trigram estimates by incorporating information from more reliable bigram and unigram estimates.

Although the smoothed trigram is obviously flawed as a model of natural language, it has proven surprisingly difficult to come up with alternatives that significantly outperform it. One problem is that it is hard to find an effective way of incorporating additional information into a trigram. To illustrate this, consider a *cache* model [10, 9] consisting of a unigram distribution estimated from relative frequencies over the last several hundred words of $\mathbf{h}$. Clearly this captures information about recent lexical preferences that would be useful in enhancing static trigram predictions at the current position. However, the optimum method for combining trigram and cache distributions is not obvious. A standard approach is to take a linear combination of the form $a\, p_{\text{trigram}}(w|\mathbf{h}) + b\, p_{\text{cache}}(w|\mathbf{h})$, where $a + b = 1$, but this method has the drawback that it tends to average over the strengths and weaknesses of both models. In a context where the trigram is able to make a confident prediction but the cache is unsure, for example, the combining weights will dilute the strength of the trigram's prediction. In principle this could be remedied by making the combining weights depend on $\mathbf{h}$, but in practice (as in this example) it is not always obvious how to do this in such a way that the weights themselves can be estimated reliably.

An alternate approach for combining information from different sources has recently become popular for NLP applications. This technique, known as Maximum Entropy/Minimum Divergence (MEMD) modeling, uses a collection of *features*—real-valued functions $f(w, \mathbf{h})$—to capture arbitrary relations between a word and its context. There are no restrictions on what can constitute a feature. For example, a feature could return the frequency of w in the last hundred words of $\mathbf{h}$ (analogous to a cache model); or it might capture the fact that w belongs to a particular grammatical category and that the last few words in $\mathbf{h}$ belong to a particular sequence of grammatical categories; or it might indicate w's semantic class. The main strength of MEMD modeling is that it provides great flexibility for combining disparate sources of information; and furthermore, in contrast to techniques like linear combination, it does so in a theoretically-justified way, as will be described in the next section.

The drawback to MEMD is that it is very computationally expensive. For example, although it would be possible to construct the equivalent of a trigram model within this framework, the implementation would be very cumbersome. Fortunately, there is a way around this: MEMD also provides for a *reference* distribution, which plays a role similar to that of a Bayesian prior. In the case of language modeling, a natural candidate for this distribution is a smoothed trigram. Using a trigram as a reference model gives the MEMD model good baseline performance, and one can hope to find a set of features that will improve substantially on this baseline without sacrificing too much of the trigram's efficiency as a result.

The main challenge in MEMD modeling is finding a maximally informative set of features. Linguistically-motivated heuristics can be used to define feature sets, but a better approach is to use linguistic knowledge to define an initial large pool of candidate features and then select a subset from this pool on the basis of empirical evaluation (the assumption here is that the initial pool is either too large to be practical or so large as to overfit the training corpus). Berger et al [3] have described a powerful and natural algorithm for feature selection, but unfortunately it is too expensive to be used for full-scale language models trained over large corpora. In a recent paper [12], Printz suggests a much more efficient variation on Berger et al's algorithm, which has the potential to make automatic feature selection more viable for language modeling.

For my project I implemented a MEMD language model for French which uses a smoothed trigram as a reference distribution and a set of *triggers* as features. Triggers are functions of the form:

$$f_{uv}(w, \mathbf{h}) = \begin{cases} 1, & u \text{ occurs in the last } L \text{ words of } \mathbf{h}\text{, and } v = w \\ 0, & \text{else} \end{cases}$$

where u and v are words in the vocabulary, and the "window length" parameter L is the same for all triggers. Triggers are a kind of generalization of a unigram cache model (which can be thought of as a collection of "self triggers" f_{uu}), but with frequency replaced by a binary indicator function for reasons of efficiency and robustness. They were first used by Rosenfeld [11, 13] within a ME framework very similar to the one I describe here, but with the important difference that in Rosenfeld's case trigrams were incorporated as features instead of within a reference distribution. Triggers have also been used by Berger and Printz [2].

The aim of my experiments was twofold: to evaluate Printz' algorithm, both from a practical standpoint and empirically by comparing it to a simple heuristic method for selecting trigger features; and to measure the performance improvement over a trigram obtainable by using trigger features within an MEMD framework, in comparison to the improvement given by a roughly equivalent cache model in a linear combination.

The rest of this paper is structured as follows: section 2 gives a formal description of the MEMD method, section 3 describes the feature selection algorithms, section 4 describes the training algorithms, section 5 describes the experimental setup and results, and section 6 concludes.

2 Maximum Entropy/Minimum Divergence Model

A MEMD model has the form:

$$p(w|\mathbf{h}) = q(w|\mathbf{h}) \exp(\alpha \cdot \mathbf{f}(w, \mathbf{h}))/Z(\mathbf{h}), \tag{1}$$

where $q(w|\mathbf{h})$ is a reference distribution, $\mathbf{f}(w, \mathbf{h})$ maps $(w, \mathbf{h})$ into an n-dimensional feature vector, α is a vector of feature weights (the parameters of the model), and $Z(\mathbf{h})$ is a normalizing factor (the sum over all w of the numerator term). Letting $q(w|\mathbf{h}) = \exp(f_0(w, \mathbf{h}))$, we can write:

$$\log s(w, \mathbf{h}) = f_0(w, \mathbf{h}) + \alpha \cdot \mathbf{f}(w, \mathbf{h}) \tag{2}$$

where $s(w, \mathbf{h})$ is the numerator in (1). It is easy to see that the right hand side of (2) represents a single-layer neural net with $n+1$ inputs, a fixed weight of 1 on input 0, and a single output which gives a score for $(w, \mathbf{h})$. (1) applies a softmax function summed over all words to this output. Alternately, the model can be thought of as computing a function from $\mathbf{h}$ to a vector of scores, one for each word. In this case, the net consists of $|W|$ units, where W is the vocabulary, the input and output of w'th unit are described by (2), and the vector of weights α is constrained to be identical for all units.

As their name implies, MEMD models can also be given an information-theoretic interpretion. Suppose we are confident that the expected values of the feature functions with respect to the empirical distribution $\tilde{p}$ defined by the data are an accurate reflection of their true values. If this is all that is known about the true distribution, a reasonable estimation strategy is to use the distribution that has maximum entropy among all those which have the desired expected values. The idea is to avoid making any inferences which are not supported by the data. If, in addition to the data, some prior (or reference) distribution q is available, then finding the distribution with maximum entropy can be generalized to finding the distribution p with minimum Kullback-Liebler divergence (relative entropy) $D(p||q)$ from the reference, where $D(p||q) = \sum_x p(x) \log[p(x)/q(x)]$. Given q and a set of expected-value constraints of the form:

$$E_p[f_i(w, \mathbf{h})] = E_{\tilde{p}}[f_i(w, \mathbf{h})], \ i = 1 \ldots n,$$

it can be shown [3] that the distribution which satisfies the constraints with minimum divergence from q is unique and has the form:

$$p(w, h) = q(w, h) \exp(\alpha \cdot \mathbf{f}(w, \mathbf{h}))/Z$$

where Z is the sum over all $(w, \mathbf{h})$ of the numerator. (For most NL applications, Z is too expensive to compute, so the conditional form (1) is used instead of the joint—fortunately, Z cancels out when deriving the conditional using Bayes' law.) Remarkably, it can also be shown that the minimum discrimination information distribution is also the maximum likelihood distribution with respect to α, and furthermore that likelihood is a convex function of α, so hillclimbing techniques are guaranteed (in principle) to find the MDI/ML distribution in this case.

In certain fields such as physics (where the technique originated [7, 6]), this result is of direct practical utility, because the expected values of a relatively small number of parameters can be measured with precision. However, for most NLP applications—including language modeling—it is not, because the reliable statistics on which the model should be based are unknown. Features must therefore be established using heuristics, as described in the next section.

It is interesting to examine intuitively how a model of the form (1) differs from linear combination as a method of combining different sources of information. To simplify the comparison, suppose that there are two equally-valuable sources of information to be combined, captured in the case of linear combination by two distributions p_1 and p_2 with a weight of .5, and in the case of MEMD by two feature functions f_1 and f_2, each with a weight of 1. Suppose that, in some context, p_1 and f_1 both make a maximally strong prediction of some word w, while p_2 and f_2 are maximally uncertain. Then the resulting linear combination will assign a probability of .75 to w, but the MEMD model will assign a probability approaching 1. Thus MEMD exhibits a built-in bias in favour of strong predictions, (and also strong anti-predictions) which may lead it to outperform a linear combination whenever such preferences are based on reliable empirical evidence.

3 Feature Selection

The intuition which underlies Berger et al's feature-selection algorithm is simply that, because training a MEMD model involves maximizing the likelihood, the optimum set of features of a given size should be the one which assigns highest likelihood to the corpus. Since determining the likelihood associated with all feature sets is combinatorially impossible, a sensible strategy is to grow a set one feature at a time by greedily adding at each step the new feature which gives the greatest increase in likelihood over the current model. However, even this is very expensive, because it necessitates training a new model for each candidate feature that is to be evaluated. Berger et al therefore advocate holding all parameters of the current model fixed and optimizing only the weight associated with the current candidate feature. Formally, define the *gain*, G_f, due to a feature f to be its associated maximum log-likelihood ratio:

$$G_f = \max_{\alpha} G_f(\alpha) = \max_{\alpha} \frac{1}{T} \, log \frac{p_{f\alpha}(\mathbf{w})}{p(\mathbf{w})}$$

where $\mathbf{w} = w_1, \ldots, w_T$ is the training corpus, p is the model at the current step, and $p_{f\alpha}$ is p with the addition of the feature f with a weight of α. The feature which gets added to the model at each step will be the one with the highest gain.

The essential step in feature selection is therefore that of maximizing $G_f(\alpha)$ in α. Berger et al observe that this can be done in parallel for all candidate features at once, and suggest using Newton's method to find the root of the derivative $G'_f(\alpha)$. Each iteration of Newton's method requires one pass over the corpus to calculate $G'_f(\alpha)$ and $G''_f(\alpha)$ for the current value of α.[2] Although this is not substantially more expensive than training a model (see the next section), it must be kept in mind that the purpose of feature selection is to winnow a very large initial pool of features down to a manageable size. If the initial pool is too large to permit training in a reasonable time, it will also be too large to permit gains to be computed in a reasonable time.[3] Another difficulty with this method of feature selection stems from its incremental nature: searching a large pool of features for the best subset of size 100k, for example, would require 100k feature-addition steps and therefore at least this many passes over the corpus.

Printz' method addresses both these problems. First, he suggests that Newton's algorithm can be replaced by an approximated function method

[2] Alternately, certain values associated with each feature can be calculated in a single pass over the corpus and stored in memory, allowing the algorithm to proceed without any future references to the corpus. For large feature sets and corpora the memory requirements of this technique are prohibitive.

[3] Note, though, that there is one important difference between training and computing gains: features interact during training but not when computing gains. This means that gains could still be computed, sequentially, for a set of features that was too big to fit in memory at once.

which requires only a single pass over the corpus. Second, he advocates doing away with repeated feature addition steps by simply using the gain over the reference model as a measure of a feature's worth in the final model. His method can therefore be summarized as follows: compute approximate gains with respect to the reference model in a single pass over the corpus, rank features in order of decreasing approximate gain, then select the optimum cutoff point in this ranking. In theory, this seems a radically suboptimal search, but it is justified if it allows for the fruitful exploration of very large feature sets which would otherwise remain inaccessible.[4]

The core of Printz' algorithm is the approximated function method, which in essence works as follows. For each feature f (in parallel, if desired), compute the value of $G'_f(\alpha)$ over a pre-determined set of test points $\{\alpha_1, \dots, \alpha_s\}$. Fit these points using least-squares polynomial interpolation, then use the interpolating function to solve numerically for the root, which approximates $\hat{\alpha}$, the optimum weight for f. Finally, integrate the interpolating function numerically (over a range which depends on $\hat{\alpha}$) to get an approximation for G_f.[5] Of all these steps, only the computation of $G'_f(\alpha)$ depends on the corpus, so the algorithm requires only a single pass (assuming a memory overhead of S times the number of features, for storing intermediate values of $G'_f(\alpha_1), \dots, G'_f(\alpha_s)$).

The approximated function method has a fairly large number of parameters which must be tuned, including stopping criteria for the three numerical algorithms used (least-squares polynomial, Newton-Raphson root-finding, and Romberg integration), order of the interpolating polynomial, and number and location of the test points. Of these, I found that performance was particularly sensitive to the polynomial order and the location of test points, with severe degradation in cases where the test points failed to bracket $\hat{\alpha}$, or where $\hat{\alpha}$ was a large negative value and the polynomial order was low. Furthermore, I found that the accuracy of the numerical approximation to G_f was very sensitive to small errors in the estimate of $\hat{\alpha}$, suggesting that, where time permits, it might be better to use an additional pass over the corpus to calculate the true values of $G_f(\tilde{\alpha})$, (where $\tilde{\alpha}$ is the approximation to $\hat{\alpha}$) instead of relying on numerical integration. Table 3 shows the performance

[4]My own experience underscores this last point: machine-time limitations forced me to evaluate Printz' algorithm by comparing it only to a simple heuristic, rather than to Berger et al's algorithm as well, which would have been more interesting.

[5]I have glossed over many details here, in the process obscuring the reason for using numerical rather than analytic integration when the interpolating function is apparently a polynomial. In actual fact, for technical reasons, the interpolating function is the reciprocal of a polynomial, and the least-squares fit is to the reciprocals of the test points.

$G_f(\hat{\alpha})$	$G_f(\tilde{\alpha})$	$\tilde{G}_f(\tilde{\alpha})$	$\hat{\alpha}$	$\tilde{\alpha}$	trigger
7.43e-05	7.20e-05	6.78e-05	0.810	1.012	m. $\rightarrow$ m.
2.50e-05	1.98e-05	7.38e-05	-0.389	-2.510	gouvernement $\rightarrow$ président
1.84e-05	1.26e-05	4.88e-05	-0.385	-1.490	je $\rightarrow$ loi
8.17e-06	6.41e-06	1.61e-06	0.281	0.186	président $\rightarrow$ est
7.65e-06	7.21e-06	1.17e-05	-0.258	-0.645	ministre $\rightarrow$ canada
3.82e-06	3.82e-06	3.89e-06	-0.103	-0.104	est $\rightarrow$ je
2.69e-06	2.42e-06	4.52e-06	-0.133	-0.182	a $\rightarrow$ gouvernement
2.69e-06	2.01e-06	5.84e-06	-0.161	-0.364	gouvernement $\rightarrow$ canada
9.87e-07	3.44e-07	3.05e-06	-0.098	-0.238	nous $\rightarrow$ ministre
6.88e-07	6.50e-07	1.04e-06	-0.070	-0.234	loi $\rightarrow$ canada

Table 1: Performance of Printz' algorithm on a set of trigger features selected randomly from among frequent words, over a 390k word training corpus. Features are ordered by decreasing true gain $G_f(\hat{\alpha})$ calculated by training, for each feature, a single-feature model with a reference trigram distribution, to get the optimum weight $\hat{\alpha}$. $\tilde{\alpha}$ denotes the approximation to $\hat{\alpha}$, and $\tilde{G}_f$ is the approximated gain.

of Printz' algorithm on a small set of triggers, and illustrates that—for this test at least—the resulting feature ordering does not correspond to the true ordering. However, because the results seem fairly reasonable, to save time I adopted Printz' parameter values verbatim, and used a 4th order polynomial with the (logscale) test points $\{-1, 0, 2, 6, 10, 14\}$ for all experiments.

4 Training

A simple way to train MEMD models is by gradient ascent. The log-likelihood of the corpus is given by:

$$\log p(\mathbf{w}) = \sum_{t=1}^{T} \log q(w_t|\mathbf{h}_t) + \alpha \cdot \mathbf{f}(w, \mathbf{h})) - \log Z(\mathbf{h}_t).$$

Differentiating with respect to the ith weight:

$$\begin{aligned} \frac{\partial \log p(\mathbf{w})}{\partial \alpha_i} &= \sum_{t=1}^{T} f_i(w_t, \mathbf{h}_t) - \frac{\partial \log Z(\mathbf{h}_t)}{\partial \alpha_i} \\ &= \sum_{t=1}^{T} f_i(w_t, \mathbf{h}_t) - \sum_{w} \frac{q(w|\mathbf{h}_t) f_i(w, \mathbf{h}_t) \exp(\alpha \cdot \mathbf{f}(w, \mathbf{h}_t))}{Z(\mathbf{h}_t)}. \end{aligned}$$

This expression can be used to update each parameter in the usual way, either in "batch" mode:

$$\alpha_i \leftarrow \alpha_i + \eta \frac{\partial \log p(\mathbf{w})}{\partial \alpha_i},$$

or by using the individual terms in the sum to update parameters at each position in the corpus.

Although gradient ascent is simple to implement, its convergence properties are heavily dependent on the gradient step η, and it can be difficult to specify ahead of time an optimum value (or progression of values) for this variable. Berger et al [3] describe a specialized training algorithm for MEMD called *Improved Iterative Scaling* (IIS), based on earlier work by Darroch and Ratcliff [5], which converges much faster than gradient ascent and does not require setting a step parameter.

Each iteration of IIS involves solving numerically (via Newton's algorithm) for a weight update quantity Δ_i in the following equation:

$$E_p[f_i(w, \mathbf{h}) \exp(\Delta_i \mathrm{f}^{\#}(w, \mathbf{h}))] = E_{\tilde{p}}[f_i(w, \mathbf{h})],$$

where $\mathrm{f}^{\#}(w, \mathbf{h})$ gives the number of features which are active (take on non-zero values) for $(w, \mathbf{h})$, and the expected values are with respect to $p(w, \mathbf{h})$ and $\tilde{p}(w, \mathbf{h})$ respectively. Once Δ_i has been calculated for each weight α_i, all weights are updated according to: $\alpha_i \leftarrow \alpha_i + \Delta_i$, and the process continues until convergence.

A major problem with this equation is that, for language models, the expectation with respect to the joint model $p(w, \mathbf{h})$ is impossible to compute due to the sum over $\mathbf{h}$. To get around this, a common solution is to use the distribution $\tilde{p}(\mathbf{h})p(w|\mathbf{h})$ instead of $p(w, \mathbf{h})$; in other words, to constrain the marginal of the joint model to its empirical value during training. Under this assumption, and the fact that $\tilde{p}(\mathbf{h}_t) = 1/T$ at each position t in the corpus, the above equation becomes:

$$\sum_{t=1}^{T} \sum_{w} p(w|\mathbf{h}_t) f_i(w, \mathbf{h}_t) \exp(\Delta_i \mathrm{f}^{\#}(w, \mathbf{h}_t)) = T E_{\tilde{p}}[f_i(w, \mathbf{h})]$$

The key to implementing IIS efficiently is to observe that this can be rewritten as a polynomial by making the transformation $\gamma_i = \exp(\Delta_i)$:

$$\sum_{t=1}^{T} \sum_{w} p(w|\mathbf{h}_t) f_i(w, \mathbf{h}_t) \gamma_i^{\mathrm{f}^{\#}(w, \mathbf{h}_t)} = T E_{\tilde{p}}[f_i(w, \mathbf{h})].$$

Thus if the set of (integer) values that $f^{\#}()$ takes on whenever feature i is active throughout the corpus is known,[6] the main part of the algorithm reduces to collecting the values of the coefficients corresponding to these exponents. At the end of each iteration, when the coefficients are known for all features, Newton's algorithm can be applied to solve efficiently for the value of γ_i that satisfies the polynomial equation for each feature.

The costly part of IIS is computing $p(w|\mathbf{h})$ for each word in the vocabulary and each history in the corpus. Rather than describing in detail how I dealt with this, I will sketch an efficient method for calculating $Z(\mathbf{h})$, which indicates the essence of the approach; similar techniques were applied to gradient ascent and Printz' feature selection algorithm. The main idea is to organize the lookup for the features which are active (ie, non zero) on $\mathbf{h}$ to supply a set of words to which at least one of these features applies. Calling this set A, we can write:

$$\begin{aligned}\sum_w q(w|h)\exp(\alpha\cdot\mathbf{f}(w,\mathbf{h})) &= \sum_{w\in A} q(w|h)\exp(\alpha\cdot\mathbf{f}(w,\mathbf{h})) + \sum_{w\notin A} q(w|h)\\ &= \sum_{w\in A} q(w|h)[\exp(\alpha\cdot\mathbf{f}(w,\mathbf{h})) - 1] + 1,\end{aligned}$$

eliminating the sum over all "non-active" words in the vocabulary, which can increase efficiency considerably when only a relatively small set of words are active for each history, and when the reference model takes non-negligible time to compute.

In practical tests, I found that IIS gave much better results than gradient ascent, usually converging within 20 iterations or so compared with 50 or more—for models that can take days to train, this is a significant savings. I used IIS for all MEMD training described in the next section.

5 Experiments

5.1 Test Setup

I ran all experiments using the French text of the Canadian Hansard corpus (transcripts of parliamentary proceedings), for the years 86-94. After certain filtering operations,[7] the corpus consisted of about 34M words in 1008

[6] This can be established in a single preliminary pass through the corpus, since these values don't change during training.

[7] Eliminating sentences longer than 40 words, and those which do not translate into a single English sentence. This was done for other work, and has no relevance to language

name	purpose	files	words
train A	main training	918	31,709,800
train B	trigram interpolation parameters	30	1,082,350
train C	feature cutoff	30	1,241,581
test	test corpus	30	1,103,320

Table 2: Corpus division; note that the four segments shown are contiguous and in chronological order.

files, with each file usually corresponding to a single day's parliamentary record. An important characteristic of the Hansard is that it is chronologically ordered, so adjacent files tend to be similar, and later files contain more information about earlier ones than the converse. Since most language model applications will involve new (ie, future) Hansard text, this means that using a random selection of files as a test corpus will give optimistic results when the rest of the corpus has been used for training. A more realistic evaluation can be obtained by training on some initial portion of the Hansard and testing on a later portion. Because of this, I split the corpus into contiguous train and test portions with the training set further subdivided into a main block (A) and two "held-out" blocks (B and C), as shown in table 5.1. Due to time constraints I used only a single train/test split.

All training and testing was performed on a version of the corpus in which both tokens (ie, word occurences) and sentences had been automatically identified. To control for out-of-vocabulary words, all models used the same vocabulary, consisting of all words which appeared more than once in blocks A and B, plus one special unknown word *UNK*. During training, any word with frequency 1 in the corpus was mapped to UNK; during testing, any word not found in the vocabulary was mapped to UNK, and the probability of each UNK token was divided by the total number of out-of-vocabulary words encountered in the text.

The models' performance was evaluated using the standard *perplexity* measure [1]. This is a geometric average over token probabilities, $p(\mathbf{w})^{-1/T}$, where $p(\mathbf{w}) = \prod_{t=1}^{T} p(w_t|w_1 \dots, w_{t-1})$ is the probability assigned to the corpus by the model. Perplexity is a useful measure because it takes on convenient values (on the order of 100), is independent of corpus size, and has an intuitive interpretation as the size of a uniform distribution (assumed

---

modeling, but available disk space did not permit me to maintain a separate version of the corpus with these constraints removed.

to contain the correct word) which would give the same performance as the model being evaluated. One thing to note about this measure is that its value is infinite whenever a model assigns zero probability to a word in the corpus.

5.2 Reference Model

The reference model for the MEMD models used in all experiments was a standard interpolated trigram, of the form:

$$\begin{aligned} p(w|\mathbf{h}) &= \phi_3(w'', w')\tilde{p}_3(w|w'', w') + \phi_2(w'', w')\tilde{p}_2(w|w') + \\ &= \phi_1(w'', w')\tilde{p}_1(w) + \phi_0(w'', w')p_0(w), \end{aligned}$$

where $\tilde{p}_i()$ is the empirical distribution over i-grams, $p_0(w)$ is a uniform distribution over all words in the vocabulary, and ϕ_i is the weight associated with the ith distribution when w'', w' are the last two words in $\mathbf{h}$. Following standard practice, I let the weights depend on the frequency of the conditioning bigram, ie $\phi(w'', w') = \phi(\text{freq}(w'', w'))$, so there is one set of weights for each distinct bigram frequency in the training corpus.[8]

The first step in creating this model was to count ngrams over block A. More specifically, for $i = 1 \ldots 3$, i-gram frequencies were collected over sentences by prepending $i - 1$ special markers to the beginning of each sentence, then sliding a window of length i along the sentence, counting one i-gram at each position, until the end of the window reached the end of the sentence. This resulted in 7,267,001 trigrams, 1,786,611 bigrams, and 66,509 unigrams; the total vocabulary contains 66,718 words. For each i, the associated empirical distribution is defined as:

$$\tilde{p}_i(w_i|w_1, \ldots, w_{i-1}) = \frac{\text{freq}(w_1, \ldots, w_i)}{\sum_{w_i} \text{freq}(w_1, \ldots, w_i)}.$$

The next step was to estimate maximum likelihood values for the combining weights $\phi_i(w'', w')$. The corpus used for this must be distinct from the one used to collect ngram frequencies, otherwise there will be a bias toward higher-order ngrams; it is easy to see that in the case when both corpora are identical, maximum likelihood will assign a weight of 1 to the empirical trigram distribution. Because there are far fewer combining weights than ngram parameters, (26,570 versus over 9M), I used the much smaller block B to estimate them. Maximum likelihood values were obtained from the

[8] This is not the optimum way of constructing an ngram model [4], but it gives good results and is easy to implement.

model	perplexities			
	train A	train B	train C	test
$\tilde{p}_3$	18.51			
$\tilde{p}_2$	66.24			
$\tilde{p}_1$	703.61			
p_0	67549.6			
p	23.57	47.17	55.98	61.05

Table 3: Perplexities of the trigram reference distribution p and its components on different segments of the corpus. The rise in perplexity with "distance" from block A reflects the chronological nature of the Hansard. The perplexities of B, C, and test for the empirical models are not shown because they are infinite.

EM algorithm, which in this case takes a particularly simple form. Each iteration collects expected values associated with the current model in each bigram context:

$$c_i(w'', w') = \sum_{t:\text{suff}(\mathbf{h}_t)=w'',w'}^{T} \frac{\tilde{p}_i(w_t|\mathbf{h}_t)\phi_i(w'', w')}{p(w_t|\mathbf{h}_t)} \quad i = 0 \ldots 3, \; \forall(w'', w') \tag{3}$$

where $\mathbf{h}_t = w_1, \ldots, w_{t-1}$, and the sum is over all such histories that end in w'', w'. At the end of each iteration, each parameter is updated according to:

$$\phi_i(w'', w') \leftarrow \frac{c_i(w'', w')}{\sum_{i=0}^{3} c_i(w'', w')}, \quad i = 0 \ldots 3, \; \forall(w'', w'). \tag{4}$$

Table 5.2 shows some of the perplexities associated with the trigram model and its components. These are quite low, indicating that the Hansard is a fairly homogeneous corpus.

5.3 Evaluating Printz' Method

I evaluated Printz's feature selection method by comparing it to an obvious heuristic for selecting trigger features: the mutual information (MI) between trigger and target words, defined as:

$$I(u; v) = \sum_{u,v} \tilde{p}(u, v) \log \frac{\tilde{p}(u, v)}{\tilde{p}(u)\tilde{p}(v)}$$

where $\tilde{p}(u, v)$ is the empirical joint distribution over trigger pairs $u \to v$, $\tilde{p}(u)$ and $\tilde{p}(v)$ are the left and right marginals of this, and the sum is over the four terms (u, v), $(\bar{u}, v)$, $(u, \bar{v})$, and $(\bar{u}, \bar{v})$.

To establish $\tilde{p}(u, v)$, I counted every occurrence of a word preceding another word in the A and B corpora. In order to reduce the size of the table that needed to be stored, pairs were limited to those where the trigger and target both occurred within the same sentence, and where neither word was among the 45 most frequent function words in the vocabulary. To make the triggers complementary to the information captured by the trigram model, pairs where the target occurred within 2 words of the trigger were also eliminated. Of the resulting approximately 20M pairs, I retained the 2M with the highest MI scores which also had pair frequencies greater than or equal to five.

My original plan was to use these pairs as an initial pool for feature selection using Printz' algorithm. However, this would not have been feasible in the available time, so I ran the algorithm on a much smaller pool of just the top 100k pairs (which still required several days to process). (Unfortunely, 100k features is suboptimal—the perplexity of a held-out training corpus versus number of features continues to drop past this point, so the comparison is less interesting than it might have been.) In order that the information available to the algorithm would be similar to that on which MI scores were based, I used a MEMD model with a fixed trigger-window length of 15.

After computing approximate gains with Printz' algorithm, I sorted the trigger list in order of decreasing gain and compared it to the same list sorted in order of decreasing MI score. The results are significantly different, as shown in table 5.3. Empirical comparisons were made by training MEMD models (again with window length 15) using the top n features in both lists, for each n in $\{1000, 2000, 5000, 10000, 20000, 50000, 100000\}$. Due to time constraints, the training corpus was limited to the last 100 files in A and B (3.6M tokens in total). The performances of these models over the test corpus are shown in figure 1 and table 5. In all cases (except on the 100k feature set, where the two models are identical), the gain-based model outperforms the MI-based model by a small margin.

To test the significance of this result, I computed individual perplexities for each of the 30 files in the test corpus for both MI and gain models at each feature-set size. Since these perplexities show significant variation from file to file, with only a barely discernable upward trend with time, it seems reasonable to treat them as iid. Although the standard deviation for any particular model is high, the deviation of the difference between MI and gain

MI ranking	gain ranking
m. $\rightarrow$ président	<< $\rightarrow$ >>
<< $\rightarrow$ >>	ne $\rightarrow$ pas
m. $\rightarrow$ monsieur	ils $\rightarrow$ ils
président $\rightarrow$ paproski	suppléant $\rightarrow$ monsieur
hon. $\rightarrow$ monsieur	seulement $\rightarrow$ mais
hon. $\rightarrow$ président	elle $\rightarrow$ elle
hon. $\rightarrow$ ministre	m. $\rightarrow$ m.
voix $\rightarrow$!	monsieur $\rightarrow$ suppléant
m. $\rightarrow$ –	non $\rightarrow$ mais
– $\rightarrow$ président	n' $\rightarrow$ pas

Table 4: Top ten trigger pairs for MI and gain ranking methods. (The angle brackets are French quotation marks, rendered incorrectly by LaTeX.)

perplexities at a given feature-set size is low; most of the global deviation appears to be caused by the reference trigram, which is by far the strongest determinant of a file's perplexity. Computing the standard normal cutoff points $\sqrt{30}\bar{\Delta}/S$ for perplexity differences Δ using the figures in the last column of table 5 gives values in the approximate range 10–30, so the null hypothesis that the gain-based models are no better than MI-based models has vanishingly small probability.

5.4 Evaluation of MEMD Models

The second set of tests I performed was simply aimed at measuring the improvement of the trigger-based MEMD models over the reference trigram, and comparing this improvement to that given by a cache model incorporated within a linear combination. Although, as mentioned earlier, the MEMD and cache models have certain similarities, they are not similar enough to permit well-founded claims that, for instance, MEMD combinations are superior to linear combination or the contrary. The interest in comparing these two models is therefore mainly practical, to see whether MEMD models give more improvement over the baseline trigram than cache models, which are far simpler to implement and far more efficient.

As described above, a basic cache model is just a fixed-length buffer of the last L words in $\mathbf{h}$, over which an empirical unigram distribution is calculated. For each value of L in $\{15, 50, 100, 200, 400\}$, I combined the resulting cache

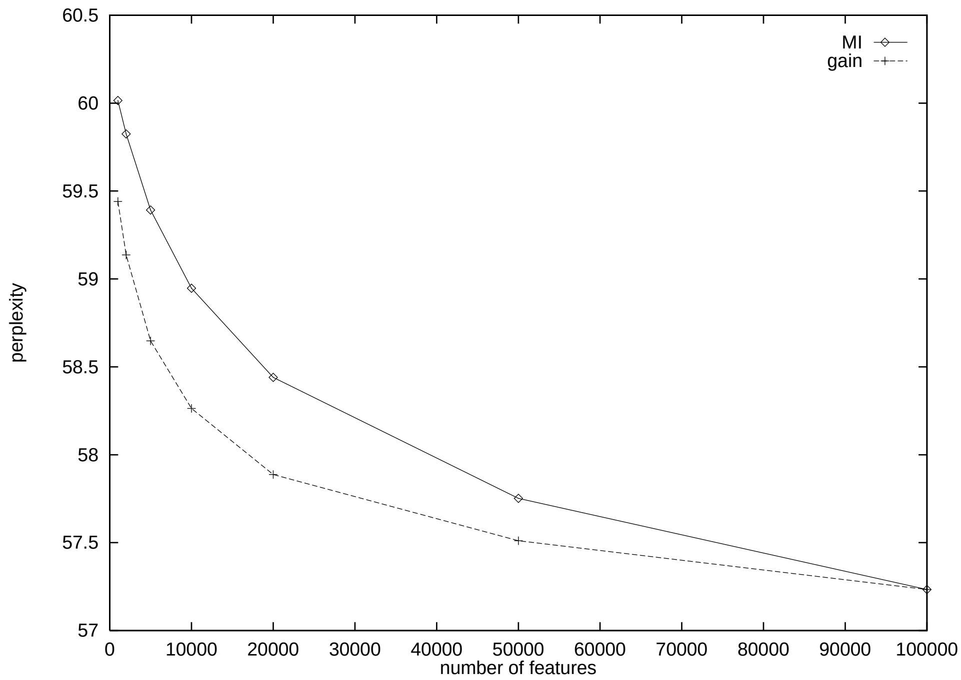

Figure 1: Performance of MI and gain feature selection methods. Each point represents the performance over the test corpus of a MEMD model using a feature set of the given size.

number of features	MI avg	MI sdev	gain avg	gain sdev	Δ avg	Δ sdev
1000	60.02	7.38	59.44	7.35	0.57	0.112
2000	59.82	7.38	59.14	7.34	0.69	0.106
5000	59.39	7.36	58.65	7.35	0.74	0.107
10000	58.95	7.34	58.26	7.35	0.68	0.127
20000	58.44	7.32	57.89	7.35	0.55	0.129
50000	57.75	7.29	57.51	7.34	0.24	0.120
100000	57.23	7.20	57.23	7.20	—	—

Table 5: Average and standard deviation of perplexity over files in the test corpus, for top MI- and gain-ranked feature sets of the given sizes. The column marked Δ reflects the differences in perplexities between each MI model and the corresponding gain model.

model	perplexity		Δ	
	avg	sdev	avg	sdev
trigram	60.48	7.39	—	—
cache 15	60.57	7.44	-0.09	0.123
cache 50	60.17	7.31	0.31	0.146
cache 100	60.07	7.26	0.41	0.183
cache 200	60.05	7.25	0.43	0.208
cache 400	60.08	7.25	0.40	0.215
MEMD 10^3	59.44	7.38	1.04	0.153
MEMD 10^4	58.26	7.34	2.27	0.201
MEMD 10^5	57.23	7.20	3.25	0.312

Table 6: Average and standard deviation of perplexity over files in the test corpus, for each model listed. The numbers beside the cache models indicate the window length L, and the numbers beside the MEMD models indicate the number of features. The column marked Δ reflects the differences in perplexities between each model and the reference trigram.

model with the reference trigram using a single pair of context-independent weights, estimated over corpus B with the EM algorithm as a special case of equations (3) and (4).

The results are shown in table 6 (the MEMD results in this table are reproduced from table 5). Although no model gives a tremendous improvement over the trigram, the MEMD models clearly outperform the cache models, with even the smallest 1000-feature model doing significantly better than the optimum 200-word cache model. Interestingly, the cache model which is based on the last 15 words actually performs worse than the reference trigram over the test corpus—this indicates that the MEMD models are more efficient than the cache in using the relatively limited amount of context available to them. The best model tested was the 100k-feature MEMD model, which achieved over 5% lower perplexity than the reference, albeit at significantly increased computational cost. By the same analysis as used in the previous section, all these results are statistically significant.

6 Conclusion

I have described some theoretical and practical aspects of the MEMD framework for statistical modeling, as applied to the problem of natural language modeling. I implemented a class of MEMD language models which use binary trigger (word pair) features to improve on the performance of a reference trigram model, and tested them over a large French corpus drawn from the Canadian Hansard. I found that a recent algorithm for automatic feature selection due to Printz yields trigger models which are significantly better than those containing the same number of features selected on the basis of mutual information scores. MEMD models also outperformed an optimized linear combination of a trigram and a cache model, even when the number of triggers used was as small as 1000, and even though the context available to the trigger model was limited to the previous 15 words. Despite these positive results, however, a major obstacle to the use of MEMD techniques for language modeling remains the fact that, for any significant number of features, these models are very computationally expensive to train and run.

References

[1] L. R. Bahl, J. K. Baker, F. Jelinek, and R. L. Mercer. Perplexity: a measure of difficulty of speech recognition tasks. In *94th Meeting of the Acoustical Society of America*, Miami, December 1977.

[2] Adam Berger and Harry Printz. A comparison of criteria for maximum entropy / minimum divergence feature selection. In *Proceedings of the 3nd Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 97–106, Granada, Spain, 1998.

[3] Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

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

[5] J.N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *Annals of Mathematical Statistics*, 43:1470–1480, 1972.

[6] E. T. Jaynes. *Probability Theory: The Logic of Science*. Unpublished Manuscript, 1996. ftp://bayes.wustl.edu/.

[7] E.T Jaynes. Information theory and statistical mechanics. *Physical Review*, 5(106):620–630, 1957.

[8] F. Jelinek and R. L. Mercer. Interpolated estimation of Markov source parameters from sparse data. In E. S. Gelsema and L. N. Kanal, editors, *Pattern Recognition in Practice*. North-Holland, Amsterdam, 1980.

[9] F. Jelinek, B. Merialdo, S. Roukos, and M. Strauss. A dynamic language model for speech recognition. In *Fourth DARPA Speech and Natural Language Workshop*, pages 293–295, Pacific Grove, California, February 1991. Morgan Kaufmann.

[10] Roland Kuhn and Renato De Mori. A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 12(6):570–583, June 1990.

[11] Raymond Lau, Ronald Rosenfeld, and Salim Roukos. Trigger-based language models: A maximum entropy approach. In *Proceedings of*

the International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 1993, pages 45–48, Minneapolis, Minnesota, 1993. IEEE.

[12] Harry Printz. Fast computation of Maximum Entropy/Minimum Divergence feature gain. In *ICSLP*, volume 5, pages 2083–2086, Sydney, December 1998.

[13] Ronald Rosenfeld. A maximum entropy approach to adaptive statistical language modelling. *Computer Speech and Language*, 10:187–228, 1996.