FAST COMPUTATION OF MAXIMUM ENTROPY / MINIMUM DIVERGENCE FEATURE GAIN

Harry Printz
IBM Watson Research Center
Yorktown Heights, NY 10598 USA
printz@watson.ibm.com

ABSTRACT

Maximum entropy / minimum divergence modeling is a powerful technique for constructing probability models, which has been applied to a wide variety of problems in natural language processing. A maximum entropy / minimum divergence (MEMD) model is built from a *base model*, and a set of *feature functions*, also called simply *features*, whose empirical expectations on some training corpus are known. A fundamental difficulty with this technique is that while there are typically millions of features that could be incorporated into a given model, in general it is not computationally feasible, or even desirable, to use them all. Thus some means must be devised for determining each feature's predictive power, also known as its *gain*. Once the gains are known, the features can be ranked according to their utility, and only the most gainful ones retained. This paper presents a new algorithm for computing feature gain that is fast, accurate and memory-efficient.

1. INTRODUCTION

In this paper, we present a new technique, called the *approximated function method*, for computing the gain of a feature of an MEMD probability model. The gain is a well-known [4], well-studied [2] and widely-used figure of merit that arises naturally in the theory of MEMD models. Because our method performs only one pass over the training corpus, and consumes only a modest amount of memory, it constitutes a significant advance over the current state of the art. The best existing method requires either multiple passes to determine a feature's gain, or makes one pass, but consumes so much memory that it is infeasible for large problems with sophisticated base models. Our algorithm can determine solutions to a prior specified accuracy, and the incremental cost of attaining higher accuracy is small. Furthermore, our algorithm is perfectly parallelizable, among features, corpus segments, or both.

The rest of this paper is organized as follows. In Section 2. we introduce the mathematical framework appropriate to the problem, and present some basic results. In Section 3. we describe the idea that underlies our method, and in Section 4. we work through a full derivation. Section 5. reports on practical experience with our method, and Section 6. is a summary of the paper. The Appendix gives a narrative summary of our notation.

2. BACKGROUND

MEMD modeling is especially valuable for constructing language models, which are used in computer systems that process and manipulate natural language. This is the application that motivated this work, and from which our example will be drawn. However, the techniques described here can be applied to models built on any finite discrete space.

As argued in [2], the appropriate figure of merit for determining the rank of a feature is its *gain* with respect to the base model, computed over some fixed corpus. The gain is defined as follows. Let $\mathcal{C}$ denote the corpus, containing N events—in the case of language models, N words—and let $P(\mathcal{C})$ be the probability of the corpus according to the base model. Now construct an MEMD model from the base model, constrained by the feature f alone, and let α^* be the exponent determined for f. Let $P_{\alpha^*}(\mathcal{C})$ be the probability of the corpus according to the resulting model. Then the gain of the single feature f, which we will write as G_f, is defined as

$$G_f = \frac{1}{N} \log \frac{P_{f\alpha^*}(\mathcal{C})}{P(\mathcal{C})} = \frac{1}{N} \log P_{f\alpha^*}(\mathcal{C}) - \frac{1}{N} \log P(\mathcal{C}). \tag{1}$$

Thus the gain measures the improvement in cross-entropy due to f, or more simply, the information content of f.

Note that this approach requires knowing the value of α^*, which is the exponent associated with f in a trained MEMD model that includes only this feature. Thus the computational task boils down to training such a model. Because this motivates both the prior art and our own approach, we proceed to examine just what this entails.

Let w be the *future* that is to be predicted, and h the *history* upon which the prediction is based. Writing $q(w \mid h)$ for the base model and $f(w,h)$ for the feature function, we define the conditional exponential model $p_{f\alpha}(w \mid h)$ by

$$p_{f\alpha}(w \mid h) = \frac{q(w \mid h) e^{\alpha f(w,h)}}{Z(\alpha, h)}. \tag{2}$$

Here $Z(\alpha, h)$ is a normalizer, defined by

$$Z(\alpha, h) = \sum_w q(w \mid h) e^{\alpha f(w,h)}. \tag{3}$$

Now suppose that any given w and h occur together in the training corpus $\mathcal{C}$ with frequency $\tilde{c}(w,h)$. Then the probability of the corpus according to the model $p_{f\alpha}$, written $P_{f\alpha}(\mathcal{C})$, is given by

$$P_{f\alpha}(\mathcal{C}) = \prod_{w,h} p_{f\alpha}(w \mid h)^{\tilde{c}(w,h)}. \tag{4}$$

It is a remarkable fact, demonstrated in reference [4], that the MEMD model, subject to the constraint equating the empirical and model expectations of f, is precisely $p_{f\alpha^*}(w \mid h)$, where α^* is uniquely determined by

$$\alpha^* = \operatorname*{argmax}_\alpha P_{f\alpha}(\mathcal{C}). \tag{5}$$

We will exploit this result as follows. Let us define the function $G_f(\alpha)$ by

$$G_f(\alpha) = \frac{1}{N} \log \frac{P_{f\alpha}(\mathcal{C})}{P(\mathcal{C})}. \tag{6}$$

Note that the gain G_f is $G_f(\alpha^*)$. Since $P(\mathcal{C})$ is defined by

$$P(\mathcal{C}) = \prod_{w,h} q(w \mid h)^{\tilde{c}(w,h)} \tag{7}$$

it does not depend upon α, and so we have at once

$$\alpha^* = \operatorname*{argmax}_\alpha P_{f\alpha}(\mathcal{C}) = \operatorname*{argmax}_\alpha G_f(\alpha). \tag{8}$$

Thus the computation of the gain reduces to finding the α^* that maximizes $G_f(\alpha)$, and then determining the value of $G_f(\alpha^*)$. This is the problem that we solve.

3. THE BASIC IDEA

The task is to find $\alpha^* = \operatorname{argmax}_\alpha G_f(\alpha)$, and then evaluate $G_f(\alpha^*)$. The maximization is a straightforward exercise in calculus. The current method for solving this problem [1] proceeds by applying Newton-Raphson iteration to find a root of the equation $G_f{}'(\alpha) = 0$. For this reason, we will call this the *Newton-Raphson method*.

The drawback of this method is that on each iteration, the values of $G_f{}'(\alpha_j)$ and $G_f{}''(\alpha_j)$ are needed to estimate a new root, α_{j+1}. This in turn entails either a complete pass over the corpus, or inspection of each value q_{hf}, a quantity that is defined below. For a large collection of features, and a corpus containing tens of millions of positions, the former approach requires too much time, and the latter requires too much memory. The computation simply becomes infeasible. An additional computation is required to determine $G(\alpha^*)$.

Our method proceeds as follows. We begin by determining the exact value of $G_f{}'(\cdot)$ at a collection of S sample points $\alpha_0 \ldots \alpha_{S-1}$. Next we determine an approximation $\widehat{G}_f{}'(\cdot)$ to $G_f{}'(\cdot)$, by interpolating a function through the S points $\langle \alpha_0, G_f{}'(\alpha_0)\rangle, \ldots, \langle \alpha_{S-1}, G_f{}'(\alpha_{S-1})\rangle$. With this approximation in hand, we solve the equation $\widehat{G}_f{}'(\alpha) = 0$ for its unique root $\hat{\alpha}$. We then use $\hat{\alpha}$ as the solution to our original problem.

This explanation, though accurate, begs the following question: how does this help? If we have to solve the equation $\widehat{G}_f{}'(\alpha) = 0$ in the end, why not just solve $G_f{}'(\alpha) = 0$ in the first place? Moreover, we have just argued that evaluating $G_f{}'(\cdot)$ at a single point is difficult, and here we are proposing to evaluate it at S sample points.

The answer is two-fold. First, while determining $G_f{}'(\cdot)$ is difficult, determining its value at many points simultaneously, all known in advance, can be done at little additional computational effort. Most of the work consists of determining at each corpus position whether or not the feature function is active, and then accessing the appropriate base model probabilities. This information need be determined only once at each position, independent of S. (For this reason, the incremental cost of attaining greater accuracy with our method is small.)

Second, for each feature, only the S values $G_f{}'(\alpha_0) \ldots G_f{}'(\alpha_{S-1})$ need be retained, rather than the much larger set $\{q_{hf}\}$. These S values allow us to replace each difficult-to-compute function $G_f{}'(\cdot)$ by a very-easy-to-compute function $\widehat{G}_f{}'(\cdot)$; thereafter we work with the latter.

For clarity, in the rest of this document we will treat the case of finding the gain of one single feature function f, and we will suppress the f-subscript on the symbol G. However, in a typical application of these results, f is a member of a very large collection F of candidate features. The intended application of this work is to compute the gain, in parallel, of all its members.

4. DETAILED DERIVATION

We proceed to develop the approximated function method from first principles. By our earlier argument in equations (2)–(8) above, the problem is to find $\alpha^* = \operatorname{argmax}_\alpha G(\alpha)$, and then evaluate $G(\alpha^*)$. The notation we use below is discussed in the appendix.

4.1. Finding α^*

Our strategy in this section will be to manipulate $G(\alpha)$ so that its dependence upon f is localized to a single term. We then differentiate this manipulated form and cull the portion depending upon f; this will be the function that we sample and approximate.

By use of equations (2), (4), (6) and (7), the function $G(\alpha)$ may be rewritten as

$$\begin{aligned} G(\alpha) &= \sum_{w,h} \tilde{p}(w,h) \log \left(Z(\alpha,h)^{-1} q(w \mid h) e^{\alpha f(w,h)} \right) \\ &\quad - \sum_{w,h} \tilde{p}(w,h) \log q(w \mid h) \end{aligned} \tag{9}$$

where we have exploited the definition $\tilde{p}(w,h) = \tilde{c}(w,h)/N$. Expanding the argument of the log in the first summation and canceling as appropriate we have

$$G(\alpha) = \tilde{p}[f] \cdot \alpha - \sum_h \tilde{p}(h) \log Z(\alpha, h), \tag{10}$$

where $\tilde{p}[f]$ is the empirical expectation of $f(w,h)$.

If we now define $f_h = \{w \mid f(w,h) = 1\}$ and $q_{hf} = \sum_{w \in f_h} q(w \mid h)$, and make the change of variables $\gamma = e^\alpha - 1$; equivalently $\alpha = \log(\gamma+1)$; then after some algebra we obtain

$$G(\log(\gamma+1)) = \tilde{p}[f] \left(\log(\gamma+1) - \sum_h \frac{\tilde{p}(h)}{\tilde{p}[f]} \log(\gamma q_{hf} + 1) \right). \tag{11}$$

Next observe that each empirical history h^i is unique, since at a minimum it consists of all words $w^0 \ldots w^{i-1}$, and these uniquely identify position i of the corpus. Since there are N positions in the corpus, we have

$$\tilde{p}(h) = \begin{cases} 1/N & \text{if } h = h^i \text{ for some } i = 0 \ldots N-1 \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Hence we may restrict the h-summation above to those histories that actually appear in the corpus, yielding

$$G(\log(\gamma+1)) = \tilde{p}[f] \left(\log(\gamma+1) - r_f \sum_{h \in \mathcal{C}} \log(\gamma q_{hf} + 1) \right). \tag{13}$$

The notation $h \in \mathcal{C}$ beneath the sigma means that the summation proceeds serially over the corpus. Here we have written $r_f = 1/(N \cdot \tilde{p}[f])$ and moved this quantity outside the summation, since it does not depend upon h.

From here on we focus our attention on the quantity in parentheses. Writing

$$m(\gamma) = \log(\gamma+1) - r_f \sum_{h \in \mathcal{C}} \log(\gamma q_{hf} + 1) \tag{14}$$

we note that $G(\log(\gamma+1))$ and $m(\gamma)$ are related by multiplication by a constant. Hence it suffices to maximize $m(\gamma)$. Differentiating $m(\gamma)$ with respect to γ, and setting the result equal to 0, we obtain

$$m'(\gamma) = \frac{1}{\gamma+1} - r_f \sum_{h \in \mathcal{C}} \frac{q_{hf}}{\gamma q_{hf} + 1} = 0. \tag{15}$$

We can rewrite this as $m'(\gamma) = k(\gamma) - l(\gamma) = 0$ where

$$k(\gamma) = \frac{1}{\gamma+1} \quad \text{and} \quad l(\gamma) = r_f \sum_{h \in \mathcal{C}} \frac{q_{hf}}{\gamma q_{hf} + 1}. \tag{16}$$

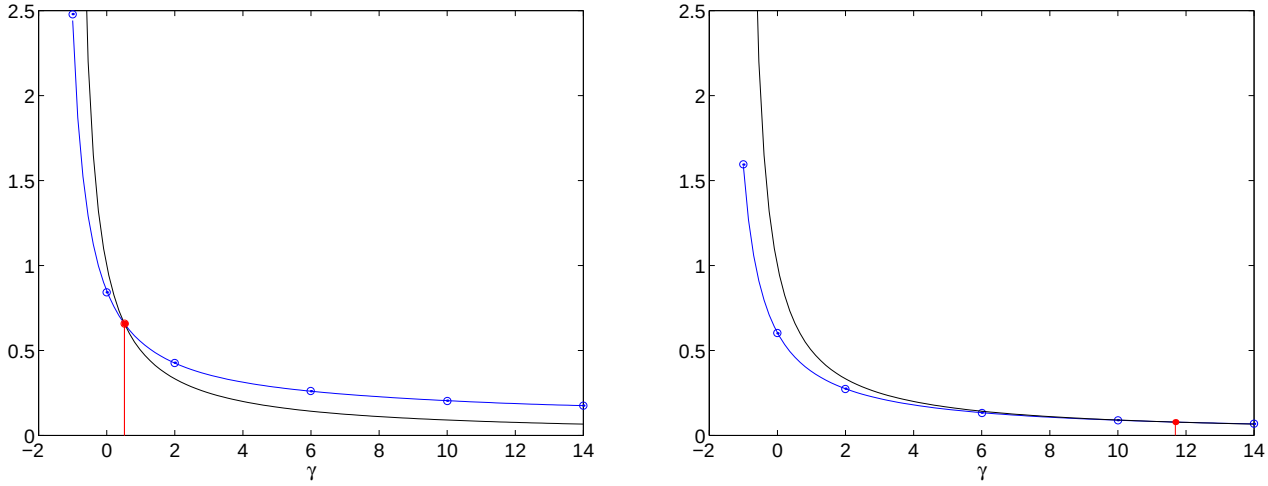

Figure 1: Operation of the Approximated Function Method. The circled points are the $\langle \gamma_j, l_j \rangle$ values, the line passing through them is the interpolated function $\hat{l}(\gamma)$, and the black line is the function $k(\gamma)$. Their intersection yields $\gamma^\star$. Left, typical trigger feature, $\gamma^\star = 0.521$. Right, typical link feature, $\gamma^\star = 11.687$. The exact $\gamma^\star$ values for these features are .545 and 11.589 respectively.

The maximum of $m(\gamma)$ is attained at the point $\gamma^\star$, where $k(\gamma^\star) = l(\gamma^\star)$.

Note that for each history h, if $q_{hf} \neq 0$ then the summand simplifies to $1/(\gamma + {q_{hf}}^{-1})$, and if $q_{hf} = 0$ then it vanishes. Thus we will write

$$l(\gamma) = r_f \widehat{\sum_{h \in \mathcal{C}}} \frac{1}{\gamma + {q_{hf}}^{-1}} \tag{17}$$

where the $\widehat{\ }$ indicates that the sum is restricted to those positions where $q_{hf} \neq 0$. We will refer to $l(\gamma)$, so defined, as the *gainsum derivative*.

Now we apply the basic principle of the approximated function method. Choosing a set Γ of sample points $\gamma_0 \ \ldots \ \gamma_{S-1}$, we scan through the corpus, accumulating the S quantities

$$e_j = \widehat{\sum_{h \in \mathcal{C}}} \frac{1}{\gamma_j + {q_{hf}}^{-1}} \tag{18}$$

for $j = 0 \ \ldots \ S-1$. The e_j are known as the *unscaled derivative sample values*, and the individual terms as the *unscaled summands*. As a matter of efficiency, we can accumulate the quantity $1/r_f = N \cdot \tilde{p}[f]$ at the same time: it is easy to see that this is just the number of empirical positions where $f(w,h) = 1$. Note that the computations for $1/r_f$ and the e_j parallelize perfectly with respect to segmentation of the corpus.

We now define $l_j = r_f \cdot e_j$ and these S values are exact, to the machine precision. We call the l_j the *gainsum derivative sample values*; these are just the e_j values, scaled by r_f. Finally, we interpolate an estimate $\widehat{l}(\gamma)$ through the S points $\{\langle \gamma_0, l_0 \rangle \ \ldots \ \langle \gamma_{S-1}, l_{S-1} \rangle\}$, and numerically solve the equation

$$k(\gamma) - \widehat{l}(\gamma) = 0 \tag{19}$$

for $\gamma^\star$. This technique is illustrated in Figure 1. We refer to (19) as the *canonical equation* of our method. With $\gamma^\star$ in hand, we work backward through the substitutions we have made, obtaining $\alpha^\star$ as $\log(\gamma^\star + 1)$.

4.2. Computation of $G(\alpha^\star)$

It remains to compute the value of $G(\alpha^\star)$. Of course it is possible to do this by evaluating $m(\gamma^\star)$ by equation (14), and then multiplying by $\tilde{p}[f]$ as indicated by equation (13) to obtain the gain. But this requires a second pass over the data, and though this is less costly than the pass that accumulated the l_j values, it remains a significant computational task.

Fortunately there is a better approach, which is based upon numerical integration of $\widehat{l}(\gamma)$. Recall from its definition that $m'(\gamma) = k(\gamma) - l(\gamma)$. Hence by the fundamental theorem of calculus,

$$m(\gamma) - m(0) = \int_0^\gamma m'(x)\,dx = \log(\gamma + 1) - \int_0^\gamma l(x)\,dx \tag{20}$$

where we have integrated $k(\cdot)$ explicitly. Approximating $l(x)$ by $\widehat{l}(x)$ in this integral, and observing from equation (14) that $m(0)$ vanishes identically, we obtain the following approximation $\widehat{m}(\gamma)$ to $m(\gamma)$:

$$\widehat{m}(\gamma) = \log(\gamma + 1) - \int_0^\gamma \widehat{l}(x)\,dx. \tag{21}$$

We will see below how to develop a compact, explicit expression for $\widehat{l}(\gamma)$, for which the numerical integration is easy to perform. Thus we have at last

$$G(\alpha^\star) = G(\log(\gamma^\star + 1)) \approx \tilde{p}[f] \cdot \widehat{m}(\gamma^\star) \tag{22}$$

and our work is done. Note that because logarithms are taken to the base e here, this formula yields the gain in *nats*. To get the gain in *bits*, multiply the result of equation (22) by $\log_2 e$.

4.3. Interpolating $\widehat{l}(\gamma)$

We have investigated three ways to develop the approximation $\widehat{l}(\gamma)$: decaying exponentials, cubic splines, and polynomial reciprocals. Each one yields a different canonical equation $k(\gamma) - \widehat{l}(\gamma) = 0$ to be solved for $\gamma^\star$. Of them, the method of polynomial reciprocals offers the best performance, and it is the only one we will discuss.

In this method, we approximate $l(\gamma)$ by the reciprocal of a polynomial $c(\gamma)$, thus $\widehat{l}(\gamma) = 1/c(\gamma) = 1/(c_0 + c_1\gamma + c_2\gamma^2 + \cdots + c_n\gamma^n)$. The coefficients of $c(\gamma)$ are obtained by a least squares fit to the reciprocals of the sample points. The resulting canonical equation can be transformed to the polynomial equation

$$(c_0 - 1) + (c_1 - 1)\gamma + c_2\gamma^2 + \cdots + c_n\gamma^n = 0, \tag{23}$$

feature	exact		degree 4			degree 5		
	$\gamma^\star$	$G(\gamma^\star)$	$\gamma^\star$	$G(\gamma^\star)$	$\%\Delta G/G$	$\gamma^\star$	$G(\gamma^\star)$	$\%\Delta G/G$
trigger	.54518	$2.2419\ 10^{-5}$	.52145	$2.0275\ 10^{-5}$	9.56	.54941	$2.2571\ 10^{-5}$	0.68
link	11.5896	$2.1696\ 10^{-3}$	11.6867	$2.1694\ 10^{-3}$	0.01	11.4321	$2.1715\ 10^{-3}$	0.09

Table 1: Accuracy of the Approximated Function Method. For each feature, this table reports exact $\gamma^\star$, $G(\gamma^\star)$ and two approximations to these quantities, and the percent error in G of the approximations (thus, $100\cdot|\Delta G|/G$). The G values are given here in bits; that is, the logarithm in equation (1) is taken to the base 2.

which exhibits better numerical behavior.

In our tests, this method yielded the best results. The $\widehat{l}(\gamma)$ displayed in Figure 1 were both obtained this way. To the eye the $\widehat{l}(\gamma)$ fit the data exactly, which may lead the reader to believe that we contrived this, by taking $n = S - 1$. In fact this is not so: we used $S = 6$ and $n = 4$ for this figure.

5. PRACTICAL EXPERIENCE

We have investigated the accuracy and memory requirements of our technique for a large MEMD model, described in [2, 3]. Our investigation of accuracy proceeded as follows. Recall from equation (1) that $G_f(\alpha^\star) = (1/N)\log(P_{f\alpha^\star}(\mathcal{C})/P(\mathcal{C}))$, where $P_{f\alpha^\star}(\mathcal{C})$ is the probability of the corpus $\mathcal{C}$ according to a trained single-feature MEMD model $p_{f\alpha^\star}(w \mid h)$. Thus by training such models, for selected features, by the improved iterative scaling algorithm [4], we can obtain exact values for $\alpha^\star$ and $G_f(\alpha^\star)$—exact to the machine precision—for comparison with the results of the approximated function method.

Table 1 supplies such a comparison. For two different features, the table gives exact $\gamma^\star$ and $G(\gamma^\star)$, along with two estimates of these same quantities, as computed by our method. All the computations were performed with six sample points, $\Gamma = \{-1, 0, 2, 6, 10, 14\}$, and using the polynomial reciprocal approximation. The different estimates were obtained using reciprocals of polynomials of degree 4 and 5 respectively. It is apparent from the table that the approximated function method yields excellent results, with agreement to better than 1%.

Our method requires only modest amounts of memory. If F is the set of candidate features, and $S = |\Gamma|$ is the number of sample points, then our method's memory requirement $M(|F|, S)$ grows as $B + R\cdot|F| + C\,|F|\cdot S$, where B is a base memory allocation to hold the vocabulary and other fixed objects, R is the memory needed to represent a feature, and C is the size of a single e_j value. Note that neither the size of the corpus, nor the characteristics of the base model, influence this expression. For the ranking of 1,538,996 features discussed in [2], the memory requirement at $S = 6$ was a modest 86 MB.

6. SUMMARY

We have presented the approximated function method—a fast, accurate and memory-efficient algorithm for computing MEMD feature gain. As reported in [2], we have used this method to compute the gain of over a million and a half features, on a corpus of over 40 million words. We hope that by enabling the ranking of large feature sets on large corpora, this method will lead to better MEMD models, more widely used.

ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation, under grant IRI-9314969. I thank Adam Berger for many interesting conversations in the course of this work, and Stephen Della Pietra, Vincent Della Pietra and John Lafferty for comments on this manuscript.

APPENDIX

We are interested in computing a family of conditional models $p(w \mid h)$. Here w is the *future*, drawn from a set $V = \{w\}$, and h is the *history*. In the case of language models, w is a word, and V is a fixed finite vocabulary.

We assume we are supplied with some large but finite training corpus $\mathcal{C}$, which consists of an ordered sequence of N futures, $w^0\ w^1\ \ldots\ w^{N-1}$. For language models, this corpus is typically divided into sentences; the corpus may also comprise additional information, such as the parse of each sentence. At any given position w^i of the corpus, the history h^i at this position consists of all the words $w^0\ \ldots\ w^{i-1}$ that precede w^i. It may also include a great deal of additional information as well, such as the parse of the sentence that contains w^i.

We write $q(w \mid h)$ for the fixed *base model* that is the starting point for our computation, and $p(w \mid h)$ for the MEMD model we compute. Typically $q(w \mid h)$ is either the uniform model or an ngram model.

We write $\tilde{p}(w, h)$ for the fixed empirical probability model on $\{w\} \times \{h\}$, and $\tilde{p}(h)$ for its w-marginal. That is, $\tilde{p}(h) = \sum_{w \in V} \tilde{p}(w, h)$. By "empirical probability model" we mean that $\tilde{p}(w, h)$ is obtained as a raw ratio of counts, $\tilde{p}(w, h) = \tilde{c}(w, h)/N$, where $\tilde{c}(w, h)$ is the number of times future w appears with history h in the corpus $\mathcal{C}$. We say that the *future, history* pair w, h is an *empirical point* if $\tilde{c}(w, h)$ is non-zero; that is, if future w appears somewhere in the corpus with history h.

Finally, $f(w, h)$ is a fixed binary-valued feature function, defined on the space of *all* pairs $\{w\} \times \{h\}$. This space is typically infinite in the second coordinate. However, it suffices to consider only empirical histories, necessarily a finite set. We assume that there is at least one empirical point w, h such that $f(w, h) = 1$. The feature function $f(w, h)$ is said to be *active* when $f(w, h) = 1$; otherwise *inactive*.

The expectation of $f(w, h)$ with respect to $\tilde{p}(w, h)$, written $\tilde{p}[f]$, is defined as $\tilde{p}[f] = \sum_{w,h} \tilde{p}(w, h) f(w, h)$. Since $\tilde{p}(w, h) = \tilde{c}(w, h)/N$, this is just the number of positions $w^i\ h^i$ where $f(w^i, h^i) = 1$, divided by the corpus size.

REFERENCES

[1] A. Berger, S. Della Pietra, V. Della Pietra, "A Maximum Entropy Approach to Natural Language Processing," *Computational Linguistics*, 22(1): 39–71, March 1996.

[2] A. Berger, H. Printz, "A Comparison of Criteria for Maximum Entropy / Minimum Divergence Feature Selection," *Proceedings of the Third Conference on Empirical Methods in Natural Language Processing*, 97–106, Granada, Spain, June 1998. Available at `www.cs.cmu.edu/~aberger/lm.html`.

[3] A. Berger, H. Printz, "Recognition Performance of a Large-Scale Dependency Grammar Language Model," *Proceedings of the Fifth International Conference on Spoken Language Processing*, Sydney, Australia, November 1998. Available at `www.cs.cmu.edu/~aberger/lm.html`.

[4] S. Della Pietra, V. Della Pietra, and J. Lafferty, *Inducing Features of Random Fields*, Technical Report CMU-CS-95-144, School of Computer Science, Carnegie Mellon, Pittsburgh, PA, May 1995.