Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, July 2002, pp. 9-16.

Sequential Conditional Generalized Iterative Scaling

Joshua Goodman
Microsoft Research
One Microsoft Way
Redmond, WA 98052
joshuago@microsoft.com

Abstract

We describe a speedup for training conditional maximum entropy models. The algorithm is a simple variation on Generalized Iterative Scaling, but converges roughly an order of magnitude faster, depending on the number of constraints, and the way speed is measured. Rather than attempting to train all model parameters simultaneously, the algorithm trains them sequentially. The algorithm is easy to implement, typically uses only slightly more memory, and will lead to improvements for most maximum entropy problems.

1 Introduction

Conditional Maximum Entropy models have been used for a variety of natural language tasks, including Language Modeling (Rosenfeld, 1994), part-of-speech tagging, prepositional phrase attachment, and parsing (Ratnaparkhi, 1998), word selection for machine translation (Berger et al., 1996), and finding sentence boundaries (Reynar and Ratnaparkhi, 1997). Unfortunately, although maximum entropy (maxent) models can be applied very generally, the typical training algorithm for maxent, Generalized Iterative Scaling (GIS) (Darroch and Ratcliff, 1972), can be extremely slow. We have personally used up to a month of computer time to train a single model. There have been several attempts to speed up maxent training (Della Pietra et al., 1997; Wu and Khudanpur, 2000; Goodman, 2001). However, as we describe later, each of these has suffered from applicability to a limited number of applications. Darroch and Ratcliff (1972) describe GIS for joint probabilities, and mention a fast variation, which appears to have been missed by the conditional maxent community. We show that this fast variation can also be used for conditional probabilities, and that it is useful for a larger range of problems than traditional speedup techniques. It achieves good speedups for all but the simplest models, and speedups of an order of magnitude or more for typical problems. It has only one disadvantage: when there are many possible output values, the memory needed is prohibitive. By combining this technique with another speedup technique (Goodman, 2001), this disadvantage can be eliminated.

Conditional maxent models are of the form

$$P(y|\overline{x}) = \frac{\exp \sum_i \lambda_i f_i(\overline{x}, y)}{\sum_{y'} \exp \sum_i \lambda_i f_i(\overline{x}, y')} \qquad (1)$$

where $\overline{x}$ is an input vector, y is an output, the f_i are the so-called indicator functions or feature values that are true if a particular property of $\overline{x}, y$ is true, and λ_i is a weight for the indicator f_i. For instance, if trying to do word sense disambiguation for the word "bank", $\overline{x}$ would be the context around an occurrence of the word; y would be a particular sense, e.g. financial or river; $f_i(\overline{x}, y)$ could be 1 if the context includes the word "money" and y is the financial sense; and λ_i would be a large positive number.

Maxent models have several valuable properties. The most important is constraint satisfaction. For a given f_i, we can count how many times f_i was observed in the training data, *observed*$[i] = \sum_j f_i(\overline{x_j}, y_j)$. For a model $P_{\overline{\lambda}}$ with parameters $\overline{\lambda}$, we can see how many times the model predicts that f_i would be expected: *expected*$[i] = \sum_{j,y} P_{\overline{\lambda}}(y|\overline{x_j}) f_i(\overline{x_j}, y)$. Maxent models have the property that *expected*$[i]$ = *observed*$[i]$ for all i. These equalities are called *constraints*. An additional property is that, of models in the form of Equation 1, the maxent model maximizes the probability of the training data. Yet another property is that maxent models are as close as possible to the uniform distribution, subject to constraint satisfaction.

Maximum entropy models are most commonly learned using GIS, which is actually a very simple algorithm. At each iteration, a step is taken in a direction that increases the likelihood of the training

data. The step size is guaranteed to be not too large and not too small: the likelihood of the training data increases at each iteration and eventually converges to the global optimum. Unfortunately, this guarantee comes at a price: GIS takes a step size inversely proportional to the maximum number of active constraints. Maxent models are interesting precisely because of their ability to combine many different kinds of information, so this weakness of GIS means that maxent models are slow to learn precisely when they are most useful.

We will describe a variation on GIS that works much faster. Rather than learning all parameters of the model simultaneously, we learn them sequentially: one, then the next, etc., and then back to the beginning. The new algorithm converges to the same point as the original one. This sequential learning would not lead to much, if any, improvement, except that we also show how to cache subcomputations. The combination leads to improvements of an order of magnitude or more.

2 Algorithms

We begin by describing the classic GIS algorithm. Recall that GIS converges towards a model in which, for each f_i, *expected*$[i]$ = *observed*$[i]$. Whenever they are not equal, we can move them closer. One simple idea is to just add $\log observed[i]/expected[i]$ to λ_i. The problem with this is that it ignores the interaction with other λs. If updates to other λs made on the same iteration of GIS have a similar effect, we could easily go too far, and even make things worse. GIS introduces a slowing factor, $f^{\#}$, equal to the largest total value of f_i: $f^{\#} = \max_{j,y} \sum_i f_i(\overline{x_j}, y)$. Next, GIS computes an update:

$$\delta_i = \frac{\log \textit{observed}[i]/\textit{expected}[i]}{f^{\#}} \qquad (2)$$

We then add δ_i to λ_i. This update provably converges to the global optimum. GIS for joint models was given by Darroch and Ratcliff (1972); the conditional version is due to Brown et al. (Unpublished), as described by Rosenfeld (1994).

In practice, we use the pseudocode of Figure 1.[1] We will write I for the number of training instances,

expected$[0..F] = 0$
for each training instance j
  for each output y
    $s[j, y] := 0$
    for each i such that $f_i(\overline{x_j}, y) \neq 0$
      $s[j, y] \mathrel{+}= \lambda_i \times f_i(\overline{x_j}, y)$
  $z := \sum_y e^{s[j,y]}$
  for each output y
    for each i such that $f_i(\overline{x_j}, y) \neq 0$
      expected$[i] \mathrel{+}= f_i(\overline{x_j}, y) \times e^{s[j,y]}/z$

for each i
  $\delta_i = \frac{1}{f^{\#}} \log \frac{\textit{observed}[i]}{\textit{expected}[i]}$
  $\lambda_i \mathrel{+}= \delta_i$

Figure 1: One Iteration of Generalized Iterative Scaling (GIS)

and F for number of indicator functions; we use Y for the number of output classes (values for y). We assume that we keep a data structure listing, for each training instance $\overline{x_j}$ and each value y, the i such that $f_i(\overline{x_j}, y) \neq 0$.

Now we can describe our variation on GIS. Basically, instead of updating all λ's simultaneously, we will loop over each indicator function, and compute an update for that indicator function, in turn. In particular, the first change we make is that we exchange the outer loops over training instances and indicator functions. Notice that in order to do this efficiently, we also need to rearrange our data structures: while we previously assumed that the training data was stored as a sparse matrix of indicator functions with non-zero values for each instance, we now assume that the data is stored as a sparse matrix of instances with non-zero values for each indicator. The size of the two matrices is obviously the same.

The next change we make is to update each λ_i near the inner loop, immediately after *expected*$[i]$ is computed, rather than after *expected* values for all features have been computed. If we update the features one at a time, then the meaning of $f^{\#}$ changes. In the original version of GIS, $f^{\#}$ is the largest total of all features. However, $f^{\#}$ only needs to be the largest total of all the features being updated, and in

[1] Many published versions of the GIS algorithm require inclusion of a "slack" indicator function so that the same number of constraints always applies. In practice it is only necessary that the total of the indicator functions be bounded by $f^{\#}$, not necessarily equal to it. Alternatively, one can see this as including the slack indicator, but fixing the corresponding λ to 0, and not updating it, so that it can be ommitted from any equations; the proofs that GIS improves at each iteration and that there is a global optimum still hold.

$z[1..I] = Y$
$s[1..I, 1..Y] = 0$
for each feature f_i
  expected $= 0$
  for each output y
    for each instance j such that $f_i(\overline{x_j}, y) \neq 0$
      expected $+= f_i(\overline{x_j}, y) \times e^{s[j,y]}/z[j]$
  $\delta_i = \frac{1}{\max_{j,y} f_i(\overline{x_j}, y)} \log \frac{\textit{observed}[i]}{\textit{expected}[i]}$
  $\lambda_i += \delta_i$
  for each output y
    for each instance j such that $f_i(\overline{x_j}, y) \neq 0$
      $z[j] -= e^{s[j,y]}$
      $s[j, y] += \delta_i$
      $z[j] += e^{s[j,y]}$

Figure 2: One Iteration of Sequential Conditional Generalized Iterative Scaling (SCGIS)

this case, there is only one such feature. Thus, instead of $f^{\#}$, we use $\max_{j,y} f_i(\overline{x_j}, y)$. In many maxent applications, the f_i take on only the values 0 or 1, and thus, typically, $\max_{j,y} f_i(\overline{x_j}, y) = 1$. Therefore, instead of slowing by a factor of $f^{\#}$, there may be no slowing at all!

We make one last change in order to get a speedup. Rather than recompute for each instance j and each output y, $s[j, y] = \sum_i \lambda_i \times f_i(\overline{x_j}, y)$, and the corresponding normalizing factors $z = \sum_y e^{s[j,y]}$ we instead keep these arrays computed as invariants, and incrementally update them whenever a λ_i changes. With this important change, we now get a substantial speedup. The code for this transformed algorithm is given in Figure 2.

The space of models in the form of Equation 1 is convex, with a single global optimum. Thus, GIS and SCGIS are guaranteed to converge towards the same point. For convergence proofs, see Darroch and Ratcliff (1972), who prove convergence of the algorithm for joint models.

2.1 Time and Space

In this section, we analyze the time and space requirements for SCGIS compared to GIS. The space results depend on Y, the number of output classes. When Y is small, SCGIS requires only a small amount more space than GIS. Note that in Section 3, we describe a technique that, when there are many output classes, uses clustering to get both a speedup and to reduce the number of outputs, thus alleviating the space issues.

Typically for GIS, the training data is stored as a sparse matrix of size T of all non-zero indicator functions for each instance j and output y. The transposed matrix used by SCGIS is the same size T.

In order to make the relationship between GIS and SCGIS clearer, the algorithms in Figures 1 and 2 are given with some wasted space. For instance, the matrix $s[j, y]$ of sums of λs only needs to be a simple array $s[y]$ for GIS, but we wrote it as a matrix so that it would have the same meaning in both algorithms. In the space and time analyses, we will assume that such space-wasting techniques are optimized out before coding.

Now we can analyze the space and time for GIS. GIS requires the training matrix, of size T, the λs, of size F, as well as the *expected* and *observed* arrays, which are also size F. Thus, GIS requires space $O(T + F)$. Since T must be at least as large as F (we can eliminate any indicator functions that don't appear in the training data), this is $O(T)$.

SCGIS is potentially somewhat larger. SCGIS also needs to store the training data, albeit in a different form, but one that is also of size T. In particular, the matrix is interchanged so that its outermost index is over indicator functions, instead of training data. SCGIS also needs the *observed* and λ arrays, both of size F, and the array $z[j]$ of size I, and, more importantly, the full array $s[j, y]$, which is of size IY. In many problems, Y is small – often 2 – and IY is negligible, but in problems like language modeling, Y can be very large (60,000 or more). The overall space for SCGIS, $O(T+IY)$, is essentially the same as for GIS when Y is small, but much larger when Y is large – but see the optimization described in Section 3.

Now, consider the time for each algorithm to execute one iteration. Assume that for every instance and output there is at least one non-zero indicator function, which is true in practice. Notice that for GIS, the top loops end up iterating over all non-zero indicator functions, for each output, for each training instance. In other words, they examine every entry in the training matrix T once, and thus require time T. The bottom loops simply require time F, which is smaller than T. Thus, GIS requires time $O(T)$.

For SCGIS, the top loops are also over each non-zero entry in the training data, which takes time $O(T)$. The bottom loops also require time $O(T)$. Thus, one iteration of SCGIS takes about as long as one iteration of GIS, and in practice in our im-

plementation, each SCGIS iteration takes about 1.3 times as long as each GIS iteration. The speedup in SCGIS comes from the step size: the update in GIS is slowed by $f^{\#}$, while the update in SCGIS is not. Thus, we expect SCGIS to converge by up to a factor of $f^{\#}$ faster. For many applications, $f^{\#}$ can be large.

The speedup from the larger step size is difficult to analyze rigorously, and it may not be obvious whether the speedup we in fact observe is actually due to the $f^{\#}$ improvement or to the caching. Note that without the caching, each iteration of SCGIS would be O($f\#$) times slower than an iteration of GIS; the caching is certainly a key component. But with the caching, each iteration of SCGIS is still marginally slower than GIS (by a small constant factor). In Section 4, we in fact empirically observe that fewer iterations are required to achieve a given level of convergence, and this reduction is very roughly proportional to $f\#$. Thus, the speedup does appear to be because of the larger step size. However, the exact speedup from the step size depends on many factors, including how correlated features are, and the order in which they are trained.

Although we are not aware of any problems where maxent training data does not fit in main memory, and yet the model can be learned in reasonable time, it is comforting that SCGIS, like GIS, requires sequential, not random, access to the training data. So, if one wanted to train a model using a large amount of data on disk or tape, this could still be done with reasonable efficiency, as long as the s and z arrays, for which we need random access, fit in main memory.

All of these analyses have assumed that the training data is stored as a precomputed sparse matrix of the non-zero values for f_i for each training instance for each output. In some applications, such as language modeling, this is not the case; instead, the f_i are computed on the fly. However, with a bit of thought, those data structures also can be rearranged.

Chen and Rosenfeld (1999) describe a technique for smoothing maximum entropy that is the best currently known. Maximum entropy models are naturally maximally smooth, in the sense that they are as close as possible to uniform, subject to satisfying the constraints. However, in practice, there may be enough constraints that the models are not nearly smooth enough – they overfit the training data. Chen and Rosenfeld describe a technique whereby a Gaussian prior on the parameters is assumed. The models no longer satisfy the constraints exactly, but work much better on test data. In particular, instead of attempting to maximize the probability of the training data, they maximize a slightly different objective function, the probability of the training data times the prior probability of the model:

$$\arg\max_{\overline{\lambda}} \prod_{j=1}^{J} P_{\overline{\lambda}}(y_j|\overline{x_j}) P(\overline{\lambda}) \tag{3}$$

where $P(\overline{\lambda}) = \prod_{i=1}^{I} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\lambda_i^2}{2\sigma^2}}$. In other words, the probability of the λs is a simple normal distribution with 0 mean, and a standard deviation of σ.

Chen and Rosenfeld describe a modified update rule in which to find the updates, one solves for δ_i in

$$\textit{observed}[i] = \textit{expected}[i] \times e^{\delta_i f^{\#}} + \frac{\lambda_i + \delta_i}{\sigma^2}$$

SCGIS can be modified in a similar way to use an update rule in which one solves for δ_i in

$$\textit{observed}[i] = \textit{expected}[i] \times e^{\delta_i \max_{j,y} f_i(\overline{x_j},y)} + \frac{\lambda_i + \delta_i}{\sigma^2}$$

3 Previous Work

Although sequential updating was described for joint probabilities in the original paper on GIS by Darroch and Ratcliff (1972), GIS with sequential updating for conditional models appears previously unknown. Note that in the NLP community, almost all maxent models have used conditional models (which are typically far more efficient to learn), and *none* to our knowledge has used this speedup.[2]

There appear to be two main reasons this speedup has not been used before for conditional models. One issue is that for joint models, it turns out to be more natural to compute the sums $s[\overline{x}]$, while for conditional models, it is more natural to compute the λs and not store the sums s. Storing s is essential for our speedup. Also, one of the first and best known uses of conditional maxent models is for language modeling (Rosenfeld, 1994), where the number of output classes is the vocabulary size, typically 5,000-60,000 words. For such applications, the array $s[j, y]$ would be of a size at least 5000 times the number of training instances: clearly impractical (but see below for

[2]Berger et al. (1996) use an algorithm that might appear sequential, but an examination of the definition of $f^{\#}$ and related work shows that it is not.

a recently discovered trick). Thus, it is unsurprising that this speedup was forgotten.

There have been several previous attempts to speed up maxent modeling. Best known is the work of Della Pietra et al. (1997), the Improved Iterative Scaling (IIS) algorithm. Instead of treating $f^{\#}$ as a constant, we can treat it as a function of $\overline{x_j}$ and y. In particular, let $f^{\#}(\overline{x}, y) = \sum_i f_i(\overline{x}, y)$ Then, solve numerically for δ_i in the equation

$$\textit{observed}[i] = \sum_{j,y} P_{\overline{\lambda}}(y|\overline{x_j}) \times f_i(\overline{x_j}, y) \times exp(\delta_i f^{\#}(\overline{x_j}, y)) \quad (4)$$

Notice that in the special case where $f^{\#}(\overline{x}, y)$ is a constant $f^{\#}$, Equation 4 reduces to Equation 2. However, for training instances where $f^{\#}(\overline{x_j}, y) < f^{\#}$, the IIS update can take a proportionately larger step. Thus, IIS can lead to speedups when $f^{\#}(\overline{x_j}, y)$ is substantially less than $f^{\#}$. It is, however, hard to think of applications where this difference is typically large. We only know of one limited experiment comparing IIS to GIS (Lafferty, 1995). That experiment showed roughly a factor of 2 speedup. It should be noted that compared to GIS, IIS is much harder to implement efficiently. When solving Equation 4, one uses an algorithm such as Newton's method that repeatedly evaluates the function. Either one must repeatedly cycle through the training data to compute the right hand side of this equation, or one must use tricks such as bucketing by the values of $f^{\#}(\overline{x_j}, y)$. The first option is inefficient and the second adds considerably to the complexity of the algorithm.

Note that IIS and SCGIS can be combined by using an update rule where one solves for

$$\textit{observed}[i] = \sum_{j,y} P_{\overline{\lambda}}(\overline{x_j}, y) \times f_i(\overline{x_j}, y) \times exp(\delta_i f_i(\overline{x_j}, y)) \quad (5)$$

For many model types, the f_i take only the values 1 or 0. In this case, Equation 5 reduces to the normal SCGIS update.

Brown (1959) describes Iterative Scaling (IS), applied to joint probabilities, and Jelinek (1997, page 235) shows how to apply IS to conditional probabilities. For binary-valued features, without the caching trick, SCGIS is the same as the algorithm described by Jelinek. The advantage of SCGIS over IS is the caching – without which there is no speedup – and because it is a variation on GIS, it can be applied to non-binary valued features. Also, with SCGIS, it is clear how to apply other improvements such as the smoothing technique of Chen and Rosenfeld (1999).

Several techniques have been developed specifically for speeding up conditional maxent models, especially when Y is large, such as language models, and space precludes a full discussion here. These techniques include unigram caching, cluster expansion (Lafferty et al., 2001; Wu and Khudanpur, 2000), and word clustering (Goodman, 2001). Of these, the best appears to be word clustering, which leads to up to a factor of 35 speedup, and which has an additional advantage: it allows the SCGIS speedup to be used when there are a large number of outputs.

The word clustering speedup (which can be applied to almost any problem with many outputs, not just words) works as follows. Notice that in both GIS and in SCGIS, there are key loops over all outputs, y. Even with certain optimizations that can be applied, the length of these loops will still be bounded by, and often be proportional to, the number of outputs. We therefore change from a model of the form $P(y|\overline{x})$ to modeling $P(\textit{cluster}(y)|\overline{x}) \times P(y|\overline{x}, \textit{cluster}(y))$. Consider a language model in which y is a word, $\overline{x}$ represents the words preceding y, and the vocabulary size is 10,000 words. Then for a model $P(y|\overline{x})$, there are 10,000 outputs. On the other hand, if we create 100 word clusters, each with 100 words per cluster, then for a model $P(\textit{cluster}(y)|\overline{x})$, there are 100 outputs, and for a model $P(y|\overline{x}, \textit{cluster}(y))$ there are also 100 outputs. Thus, instead of training one model with a time proportional to 10,000, we train two models, each with time proportional to 100. Thus, in this example, there is a 50 times speedup. In practice, the speedups are not quite so large, but we do achieve speedups of up to a factor of 35. Although the model form learned is not exactly the same as the original model, the perplexity of the form using two models is actually marginally lower (better) than the perplexity of the form using a single model, so there does not seem to be any disadvantage to using it.

The word clustering technique can be extended to use multiple levels. For instance, by putting words into superclusters, such as their part of speech, and clusters, such as semantically similar words of a given part of speech, one could use a three level model. In fact, the technique can be extended to up to $log_2 Y$ levels with two outputs per level, meaning that the space requirements are proportional to 2 instead of to the original Y. Since SCGIS works

by increasing the step size, and the cluster-based speedup works by increasing the speed of the inner loop (whchi SCGIS shares), we expect that the two techniques would complement each other well, and that the speedups would be nearly multiplicative. Very preliminary language modeling experiments are consistent with this analysis.

There has been interesting recent unpublished work by Minka (2001). While this work is very preliminary, and the experimental setting somewhat unrealistic (dense features artificially generated), especially for many natural language tasks, the results are dramatic enough to be worth noting. In particular, Minka found that a version of conjugate gradient descent worked extremely well – much faster than GIS. If the problem domain resembles Minka's, then conjugate gradient descent and related techniques are well worth trying, and it would be interesting to try these techniques for more realistic tasks.

SCGIS turns out to be related to boosting. As shown by Collins et al. (2002), boosting is in some ways a sequential version of maxent. The single largest difference between our algorithm and Collins' is that we update each feature in order, while Collins' algorithms select a (possibly new) feature to update. That algorithm also require more storage than our algorithm when data is sparse: fast implementations require storage of both the training data matrix (to compute which feature to update) and the transpose of the training data matrix (to perform the update efficiently.)

4 Experimental Results

In this section, we give experimental results, showing that SCGIS converges up to an order of magnitude faster than GIS, or more, depending on the number of non-zero indicator functions, and the method of measuring performance.

There are at least three ways in which one could measure performance of a maxent model: the objective function optimized by GIS/SCGIS; the entropy on test data; and the percent correct on test data. The objective function for both SCGIS and GIS when smoothing is Equation 3: the probability of the training data times the probability of the model. The most interesting measure, the percent correct on test data, tends to be noisy.

For a test corpus, we chose to use exactly the same training, test, problems, and feature sets used by Banko and Brill (2001). These problems consisted of trying to guess which of two confusable words, e.g. "their" or "there", a user intended. Banko and Brill chose this data to be representative of typical machine learning problems, and, by trying it across data sizes and different pairs of words, it exhibits a good deal of different behaviors. Banko and Brill used a standard set of features, including words within a window of 2, part-of-speech tags within a window of 2, pairs of word or tag features, and whether or not a given word occurred within a window of 9. Altogether, they had 55 feature types. That is, there were many thousands of features in the model (depending on the exact model), but at most 55 could be "true" for a given training or test instance.

We examine the performance of SCGIS versus GIS across three different axes. The most important variable is the number of features. In addition to trying Banko and Brill's 55 feature types, we tried using feature sets with 5 feature types (words within a window of 2, plus the "unigram" feature) and 15 feature types (words within a window of 2, tags within a window of 2, the unigram, and pairs of words within a window of 2). We also tried not using smoothing, and we tried varying the training data size.

In Table 1, we present a "typical" configuration, using 55 feature types, and 10 million words of training, and smoothing with a Gaussian prior. The first two columns show the different confusable words. Each column shows the ratio of how much longer (in terms of elapsed time) it takes GIS to achieve the same results as 10 iterations of SCGIS. An "XXX" denotes a case in which GIS did not achieve the performance level of SCGIS within 1000 iterations. (XXXs were not included in averages.)[3] The "objec" column shows the ratio of time to achieve the same value of the objective function (Equation 3); the "ent" column show the ratio of time to achieve the same test entropy; and the "cor" column shows the ratio of time to achieve the same test error rate. For all three measurements, the ratio can be up to a factor of 30, though the average is somewhat lower, and in two cases, GIS converged faster.

In Table 2 we repeat the experiment, but without smoothing. On the objective function – which with no smoothing is just the training entropy – the increase from SCGIS is even larger. On the other

[3] On a 1.7 GHz Pentium IV with 10,000,000 words training, and 5 feature types it took between .006 and .24 seconds per iteration of SCGIS, and between .004 and .18 seconds for GIS. With 55 feature types, it took between .05 and 1.7 seconds for SCGIS and between .03 and 1.2 seconds for GIS. Note that many experiments use much larger datasets or many more feature types; run time scales linearly with training data size.

		objec	ent	cor
accept	except	31.3	38.9	32.3
affect	effect	27.8	10.7	6.4
among	between	30.9	1.9	XXX
its	it's	26.8	18.5	11.1
peace	piece	33.4	0.3	XXX
principal	principle	24.1	XXX	0.2
then	than	23.4	37.4	24.4
their	there	17.3	31.3	6.1
weather	whether	21.3	XXX	8.7
your	you're	36.8	9.7	19.1
	Average	**27.3**	**18.6**	**13.5**

Table 1: Baseline: standard feature types (55), 10 million words, smoothed

		objec	ent	cor
accept	except	39.3	4.8	7.5
affect	effect	46.4	5.2	5.1
among	between	48.7	4.5	2.5
its	it's	47.0	3.2	1.4
peace	piece	46.0	0.6	XXX
principal	principle	43.9	5.7	0.7
then	than	48.7	5.6	1.0
their	there	46.8	8.7	0.6
weather	whether	44.7	6.7	2.1
your	you're	49.0	2.0	29.6
	Average	**46.1**	**4.7**	**5.6**

Table 2: Same as baseline, except no smoothing

		objec	ent	cor
accept	except	6.0	4.8	3.7
affect	effect	3.6	3.6	1.0
among	between	5.8	1.0	0.7
its	it's	8.7	5.6	3.3
peace	piece	25.2	2.9	XXX
principal	principle	6.7	18.6	1.0
then	than	6.9	6.7	9.6
their	there	4.7	4.2	3.6
weather	whether	2.2	6.5	7.5
your	you're	7.6	3.4	16.8
	Average	**7.7**	**5.7**	**5.2**

Table 3: Small feature set (5 feature types)

		objec	ent	cor
accept	except	10.8	10.7	8.3
affect	effect	12.4	18.3	6.8
among	between	7.7	14.3	9.0
its	it's	7.4	XXX	5.4
peace	piece	14.6	4.5	9.4
principal	principle	7.3	XXX	0.0
then	than	6.5	13.7	11.0
their	there	5.9	11.3	2.8
weather	whether	10.5	29.3	13.9
your	you're	13.1	8.1	9.8
	Average	**9.6**	**13.8**	**7.6**

Table 4: Medium feature set (15 feature types)

criteria – test entropy and percentage correct – the increase from SCGIS is smaller than it was with smoothing, but still consistently large.

In Tables 3 and 4, we show results with small and medium feature sets. As can be seen, the speedups with smaller features sets (5 feature types) are less than the speedups with the medium sized feature set (15 feature types), which are smaller than the baseline speedup with 55 features.

Notice that across all experiments, there were no cases where GIS converged faster than SCGIS on the objective function; two cases where it coverged faster on test data entropy; and 5 cases where it converged faster on test data correctness. The objective function measure is less noisy than test data entropy, and test data entropy is less noisy than test data error rate: the noisier the data, the more chance of an unexpected result. Thus, one possibility is that these cases are simply due to noise. Similarly, the four cases in which GIS never reached the test data entropy of SCGIS and the four cases in which GIS never reached the test data error rate of SCGIS might also be attributable to noise. There is an alternative explanation that might be worth exploring. On a different data set, 20 newsgroups, we found that early stopping techniques were helpful, and that GIS and SCGIS benefited differently depending on the exact settings. It is possible that effects similar to the smoothing effect of early stopping played a role in both the XXX cases (in which SCGIS presumably benefited more from the effects) and in the cases where GIS beat SCGIS (in which cases GIS presumably benefited more.) Additional research would be required to determine which explanation – early stopping or noise – is correct, although we suspect both explanations apply in some cases.

We also ran experiments that were the same as the baseline experiment, except changing the training data size to 50 million words and to 1 million words. We found that the individual speedups were often different at the different sizes, but did not appear to

be overall higher or lower or qualitatively different.

5 Discussion

There are many reasons that maxent speedups are useful. First, in applications with active learning or parameter optimization or feature set selection, it may be necessary to run many rounds of maxent, making speed essential. There are other fast algorithms, such as Winnow, available, but in our experience, there are some problems where smoothed maxent models are better classifiers than Winnow. Furthermore, many other fast classification algorithms, including Winnow, do not output probabilities, which are useful for precision/recall curves, or when there is a non-equal tradeoff between false positives and false negatives, or when the output of the classifier is used as input to other models. Finally, there are many applications of maxent where huge amounts of data are available, such as for language modeling. Unfortunately, it has previously been very difficult to use maxent models for these types of experiments. For instance, in one language modeling experiment we performed, it took a month to learn a single model. Clearly, for models of this type, any speedup will be very helpful.

Overall, we expect this technique to be widely used. It leads to very significant speedups – up to an order of magnitude or more. It is very easy to implement – other than the need to transpose the training data matrix, and store an extra array, it is no more complex than standard GIS. It can be easily applied to any model type, although it leads to the largest speedups on models with more feature types. Since models with many interacting features are the type for which maxent models are most interesting, this is typical. It requires very few additional resources: unless there are a large number of output classes, it uses about as much space as standard GIS, and when there are a large number of output classes, it can be combined with our clustering speedup technique (Goodman, 2001) to get both additional speedups, and to reduce the space requirements. Thus, there appear to be no real impediments to its use, and it leads to large, broadly applicable gains.

Acknowledgements

Thanks to Ciprian Chelba, Stan Chen, Chris Meek, and the anonymous reviewers for useful comments.

References

M. Banko and E. Brill. 2001. Mitigating the paucity of data problem. In *HLT*.

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

P. Brown, S. DellaPietra, V. DellaPietra, R. Mercer, A. Nadas, and S. Roukos. Unpublished. Translation models using learned features and a generalized Csiszar algorithm. IBM research report.

D. Brown. 1959. A note on approximations to probability distributions. *Information and Control*, 2:386–392.

S.F. Chen and R. Rosenfeld. 1999. A gaussian prior for smoothing maximum entropy models. Technical Report CMU-CS-99-108, Computer Science Department, Carnegie Mellon University.

Michael Collins, Robert E. Schapire, and Yoram Singer. 2002. Logistic regression, adaboost and bregman distances. *Machine Learning*, 48.

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

Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. 1997. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, April.

Joshua Goodman. 2001. Classes for fast maximum entropy training. In *ICASSP 2001*.

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

J. Lafferty, F. Pereira, and A. McCallum. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*.

John Lafferty. 1995. Gibbs-markov models. In *Computing Science and Statistics: Proceedings of the 27th Symposium on the Interface*.

Thomas Minka. 2001. Algorithms for maximum-likelihood logistic regression. Available from `http://www-white.media.mit.edu/~tpminka/papers/learning.html`.

Adwait Ratnaparkhi. 1998. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. Ph.D. thesis, University of Pennsylvania.

J. Reynar and A. Ratnaparkhi. 1997. A maximum entropy approach to identifying sentence boundaries. In *ANLP*.

Ronald Rosenfeld. 1994. *Adaptive Statistical Language Modeling: A Maximum Entropy Approach*. Ph.D. thesis, Carnegie Mellon University, April.

J. Wu and S. Khudanpur. 2000. Efficient training methods for maximum entropy language modeling. In *ICSLP*, volume 3, pages 114–117.