GaP: A Factor Model for Discrete Data

John Canny
Computer Science Division
University of California, Berkeley
jfc@cs.berkeley.edu

ABSTRACT

We present a probabilistic model for a document corpus that combines many of the desirable features of previous models. The model is called "GaP" for Gamma-Poisson, the distributions of the first and last random variable. GaP is a factor model, that is it gives an approximate factorization of the document-term matrix into a product of matrices Λ and X. These factors have strictly non-negative terms. GaP is a generative probabilistic model that assigns finite probabilities to documents in a corpus. It can be computed with an efficient and simple EM recurrence. For a suitable choice of parameters, the GaP factorization maximizes independence between the factors. So it can be used as an independent-component algorithm adapted to document data. The form of the GaP model is empirically as well as analytically motivated. It gives very accurate results as a probabilistic model (measured via perplexity) and as a retrieval model. The GaP model projects documents and terms into a low-dimensional space of "themes," and models texts as "passages" of terms on the same theme.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Retrieval Models, Clustering*

General Terms

Algorithms, Experimentation, Measurement

Keywords

Probabilistic Models, Latent Semantic Analysis, EM Algorithm

1. INTRODUCTION

Probabilistic models for document corpora are a central concern for IR researchers. Among the applications for a probabilistic model are (i) accurate search and retrieval from the corpus (ii) detection of themes in the corpus and (iii) clustering of documents according to those themes. Intuitively, the model should also fit the data well without overfitting. While these desiderata are often considered separately, we present a model which addresses them simultaneously. The model is called "GaP" for Gamma-Poisson, the distributions of the first and last random variable. These distributions were chosen because of good empirical fit for text data, but also because they give a very clean and simple algorithm which is highly efficient. Informally, GaP models texts as contiguous "passages" of variable length, where each passage contains a fixed probability distribution of words.

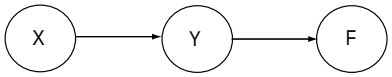

Figure 1: The GaP model. X and F have gamma and poisson distributions, respectively, and Y is a linear multiple of X by the matrix Λ.

The probabilistic model is shown in schematic form above. X is a matrix whose j^{th} column is a low-dimensional vector representing the contribution of "themes" in document j. Each X_{ij} is a *non-negative* random variable representing the contribution of theme i to document j measured as the total length of all passages on that theme. Y is a matrix whose j^{th} column is a vector (dimension equals number of terms) representing the *expected number of occurences* of term i in document j. Finally F is matrix where F_{ij} is the *actual* number of occurences of term i in document j. So F_{ij} is a discrete, non-negative random variable.

For empirical reasons, the X_{ij} are given a gamma distribution (more on this later). The relation between X and Y is a linear matrix

$$Y = \Lambda X$$

This follows because the text is modeled as a union of passages of length X_{ij} on theme i, and the total term frequencies are a sum over all such passages. Finally, if we assume independence of terms in specific locations in the document given the theme, then the F_{ij} will have poisson distributions.

The matrix Λ includes the theme information for the corpus. Each row of Λ encodes the term probabilities for a specific theme (which are non-negative). In the process of its execution, the GaP algorithm also computes the expected theme weights (matrix X) such that X_{ij} represents the estimated total passage length on theme i in document j. The matrix product ΛX is then an approximate factorization of the document-term matrix F.

SIGIR'04, July 25–29, 2004, Sheffield, South Yorkshire, UK.

GaP is a generative model which provides very good modeling of a text corpus, measured using perplexity. It also improves on some of the best current models for text retrieval (we compare it with a smoothed, KL-divergence algorithm). We give experimental results for both problems later in the paper. We believe GaP will give excellent results on clustering and other applications of factor models like LSA, but these await further studies. GaP factorization is computed using an EM-algorithm, derived in section 3. The algorithm has been designed to be as efficient as possible, and has complexity $O(Ndi)$ where N is the total number of terms in the corpus, d is the model dimension, or number of themes, and i is the number of iterations run (typically 20). Gigabyte datasets have been factored on a 2GHz desktop computer.

In the next section we discuss the relationship of other work to GaP. Then in section 3, we derive an EM-algorithm for computing a GaP factorization. In section 4 we discuss the implementation of the algorithm. This section includes an a explanation (section 3.6) of how the algorithm maximizes the statistical independence between factors.

2. RELATED WORK

Because it computes an approximate factorization ΛX of the document-term matrix, GaP is related to Latent Semantic Analysis [6, 7, 8]. It can be used for document retrieval (in either the low-dimensional theme space, or as a smoothing algorithm in term space). GaP can also be used for clustering by identifying each theme with a cluster and assigning documents to the cluster of highest weight in the vector x for that document. However, GaP factorization differs from LSA in several ways. LSA is based on Singular Value Decomposition which is equivalent to assuming a uniform distribution on X and a spherical gaussian distribution on F. Neither match the empirical distributions on real texts. Both the gamma prior and poisson distributions in GaP are a good match for empirical document data. LSA determines factors by orthogonality, which forces many components of the factorization to be negative. In GaP, both matrices Λ and X have non-negative entries and there is no orthogonality constraint. In fact we show that GaP can be used to maximize the independence of components, intuitively a more useful condition than orthogonality.

The factorization that GaP computes is a NMF (Nonnegative Matrix Factorization [10]) of the document corpus. NMF was shown in [10] to have extremely good performance in TREC clustering benchmarks. Note though, that the factorization derived from GaP will be different from [10], which is based on least-squares (implicit spherical gaussian distribution on F).

Because GaP is based on discrete output variables, it is a generative probabilistic model. That is, it assigns a finite probability to any specific document, and it can generate "random" documents from the corpus. Generative probabilistic models have been applied in IR before [3, 1], and GaP has particular similarities to LDA (Latent Dirichlet Allocation) [1, 2]. We will say more about their relationship during the derivation of the GaP model in the next section.

GaP is also related to ICA (Independent Component Analysis) algorithms [4] which also attempt to maximize the statistical independence of factors in a mixture distribution. There are two differences with ICA and GaP. First, ICA normally treats "pure" linear mixtures of factors. Since texts have discrete term frequencies, ICA is not directly applicable. GaP deals with this by interposing a poisson distribution after the factor mixture. Secondly, GaP uses known priors (gamma distributions) for the factors, based on empirical data. In ICA, the prior must be estimated which will generally require more data for a given degree of accuracy. GaP can be viewed as a customized ICA algorithm for document data, with the appropriate distributions wrapped around the linear factor model.

3. GAP MODEL DERIVATION

First we recall the form of the gamma distribution. A variable x has the gamma distribution with parameters a and b when its pdf is

$$p(x) = \frac{x^{(a-1)}\exp(-x/b)}{b^a\Gamma(a)}$$

The parameter b is called the scale parameter, and changing it is equivalent to scaling the variable x by b. The parameter a is called the shape parameter. When $a = 1$ the distribution reduces to an exponential distribution. When a is an integer, the gamma distribution is the distribution of a sum of a exponential variables. This summation property holds more generally for gamma variables with the same scale parameter. If x_1 and x_2 are gamma random variables with parameters (a_1, b) and (a_2, b) respectively, then their sum $x_1 + x_2$ is a gamma random variable with parameters $(a_1 + a_2, b)$. As a increases, by the central limit theorem, the gamma looks more and more like a normal distribution. The mean of a gamma random variable with parameters (a, b) is ab and its variance is ab^2.

To construct a language model, we let $x = (x_1, \ldots, x_k)$ be a vector of latent random variables representing the thematic content of a document. Each x_i encodes the total length of passages about topic i in the document. Then x_i is assumed to be an independent random variable with a gamma distribution with shape parameter a_i and scale parameter b_i. Independence is a natural assumption, in fact it by assuming it we are defining the kind of factorization we want - one where the themes in the document occur independently. Note that there is some necessary dependence between the x_i because of document length. Namely the expected value of the sum of the x_i will be the length of the document. However, when the theme space dimension is not too small (say 10 or greater), the effect of this constraint is minor. Since the mean value of x_i is $c_i = a_i b_i$ and c_i is easily observable, we use it to substitute for b_i in the parametrization. The prior pdf of x_i is then

$$p(x_i) = \frac{a_i(a_i x_i/c_i)^{a_i-1}\ \exp(-x_i a_i/c_i)}{c_i\Gamma(a_i)} \qquad \text{for } x_i > 0$$

and from the properties of the gamma distribution the mean $\mathrm{E}(x_i) = c_i$ and variance $\mathrm{VAR}(x_i) = c_i^2/a_i$.

From the random vector x we derive a vector y via:

$$y = \Lambda x$$

The vector y gives the expected frequencies of words in the document we are generating. Each term frequency in a document is modeled as a Poisson variable with expected value y_j. The column Λ_i of Λ is a vector of expected frequencies of terms that comprise theme i when the theme has unit weight.

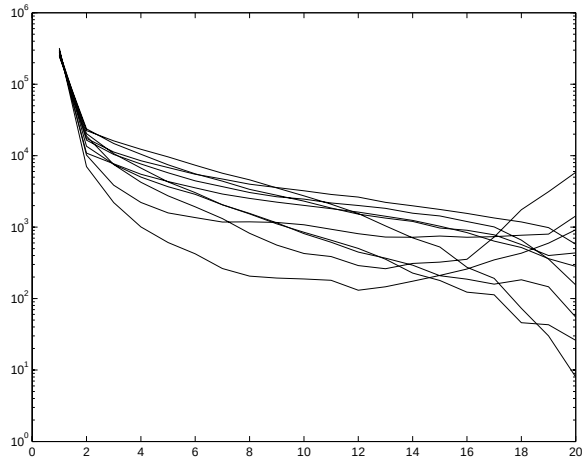

Figure 2: The probability distributions of some x_i's for the Cran corpus, computed using the EM recurrence with uniform prior

3.1 Empirical Justification

The model can be interpreted this way: Each document contains passages of various lengths (length = number of terms in the passage) on specific themes. Each x_i encodes a theme within the document such that the magnitude of x_i represents the *total* length of all passages on topic i in the document. Each theme i defines a probability distribution over the terms in the corpus, which is encoded as the i^{th} column of Λ. The probability of a term in a particular position in the document will then be a binomial distribution depending on the theme at that position. The ordering of the passages does not affect the distribution of the frequency of a term in the entire document, which will then be a poisson variable y_j. The expected value of y_j will be the sum over the various themes of the theme length x_i times the corresponding Λ probability for that term. Hence the linear form of the model, and the trailing poisson distribution.

The model so far is quite similar to Latent Dirichlet Allocation (LDA) developed by Jordan et al. [1, 2]. However, there are significant differences in the way the two models handle document length, and these lead to accuracy and performance differences. The differences arise due to the choice of prior distribution on x_i. Whereas the (intuitive) assumption of multiple themes dictates the GaP model so far (as well as LDA), it leaves completely open what the choice of distribution on x_i should be. It would be *tempting* at this point to assume a gamma distribution on x_i for analytic convenience as we will see. But a better way to proceed is to derive the mode from actual corpus data.

To do this, we replace the gamma prior with a uniform distribution on the x_i (i.e. no prior at all), and run an EM algorithm to derive the most likely Λ and X matrices. Figure 2 shows some of the X_{ij} pdf's for 300,000 TREC (Tipster 3) documents with a "no-prior GaP" factorization into 20 themes. The Y axis shows the (log scale) probability density for each x_i versus the value of that x_i. Almost all the curves show a good fit to the gamma signature: a long straight section over most of the range, with a distinct "spike" as x_i becomes very small. Note that the spikes are very similar, while the slopes vary. These two attributes correspond to the shape factor and scale factor respectively of the gamma distribution. The curves suggest that the shape factor is highly preditable while the scale factor varies significantly. In fact when we do a maximum-likelihood estimation of the gamma parameters this data, we find that all best-fit distributions have a shape factor of close to 0.19 with a standard deviation of 0.01. The scale factors however vary over a range of about $4:1$.

The gamma distribution has another very nice property when used with this model: namely that it has a scale factor, and a scaled gamma distribution is still a gamma distribution. This supports an equivalent model to the version of GaP just described: instead of the length of passages on topic i, we can scale x_i by one over the document length. The resulting $\overline{x_i}$ represents instead the *fraction* of the document on topic i. We confirmed this with a similar experiment. Namely, we computed a no-prior GaP model but where the x_i's were scaled by the reciprocal of each document's length. The plots were virtually identical to figure 2. In fact, we prefer to use this "normalized" version of the GaP model because it does not require the document length to be generated or explained by the model. The normalized version appears to perform slightly better in some situations, although it is often impossible to measure a difference.

To contrast further with LDA, in LDA the themes for each word in the document are chosen *independently* of themes for other words, according to a Dirichlet distribution. This is a more restrictive assumption than the assumption of passages (it is equivalent to assuming all passages are length 1). Our formulation could support many distributions on the x_i (length of passages). We tried several others empirically, including lognormal and Zipf, that cannot be modelled as indepedendent per-word processes. In fact, a more general model was not needed since the gamma does so well. Another way to view this is that GaP assumes that documents comprise long passages and can "scale". That is, the fractional mix of topics in a document has a similar distribution independent of document length (it depends on number of passages on the same topic in the document, not their length). LDA on the other hand, treats long documents as unions of many one-word documents with the same mixture parameters. The number of passages on the same theme per document is assumed to scale linearly with document length. The result is that the distribution of the fractional mix of topics changes with Dirichlet as the document length grows.

Note that while the fixed shape-factor gamma prior is strongly evident in TREC data, it may not be true of all document corpora. Intuitively, a fixed shape-factor GaP model should model well any documents where there are one or a few passages on the same theme in all the documents (there can of course be several passages on different themes in each document, GaP models those with the distinct x_i). In fact it strongly suggests that for TREC data there is usually only one passage per theme per document. For longer documents where there are several passages on the same theme, the distribution should be a sum of gamma-distributed variables. If the scale factors of these match (intuitively they should), then the result would be another gamma variable with a shape factor which is larger. Thus GaP models should also work well for documents containing multiple passages on the same theme. However, for such documents the shape factor for each theme should be learned for the corpus whereas for

corpora such as TREC this is not needed. This suggests an interesting as-yet unverified application of GaP models: The shape factor of the gamma for each theme provides information about the typical number of passages on that theme in each document the corpus. The larger the shape factor for a theme, the more passages on this same theme occur on average in a document.

3.2 An EM algorithm

Let f_j be the observed frequency of word j in a document. The probability distribution for the word count f_j is a Poisson variable:

$$\Pr[f_j = k] = \frac{y_j^k \exp(-y_j)}{k!} \qquad \text{for } k \geq 0$$

Notice that this same expression defines a normalized gamma pdf on the variable y_j with scale parameter 1 and shape parameter $k+1$ (because for integer k, $k! = \Gamma(k+1)$).

The likelihood for a document generated this way is

$$\prod_{j=1}^{n} \frac{y_j^{f_j} \exp(-y_j)}{f_j!} \prod_{i=1}^{k} \frac{x_i^{a_i-1} a_i^{a_i} \exp(-x_i a_i/c_i)}{c_i^{a_i}\Gamma(a_i)}$$

and the log likelihood is

$$\begin{aligned} l &= \textstyle\sum_{j=1}^{n} (f_j \log(y_j) - y_j - \log(f_j!)) \\ &+ \textstyle\sum_{i=1}^{k} ((a_i - 1)\log(x_i) - x_i a_i/c_i \\ &\qquad + a_i \log(a_i/c_i) - \log(\Gamma(a_i))) \end{aligned} \tag{1}$$

3.3 The E-step

Ideally, we would first compute the expected value of x. But a simpler approximation is the value of x which maximizes the likelihood. This is reasonable because whereas the gamma prior on x_i is highly asymmetric, the posterior for x_i conditioned on the observed y is much narrower and closer to a gaussian. Its peak and expected value are therefore very close. To find the maximum, we take the derivative wrt x_i giving:

$$\sum_{j=1}^{n} \left(\frac{f_j}{y_j} \Lambda_{ji} - \Lambda_{ji} \right) + \frac{a_i - 1}{x_i} - a_i/c_i = 0$$

As it turns out, there is a particularly efficient and simple iterative method to solve for the maximum likelihood x. Now the goal in EM is to compute the *expected*, not the maximum likelihood value. In many cases however, they will be very close. Figure 3 shows typical distributions of x variables given prior and posterior probabilities from GaP models of some TREC documents. While the gamma priors are highly asymmetric, the *summation* of likelihood contributions from all the y_j's that depend on a particular x_i lead to a narrow, gaussian-like distribution whose mean and maximum value are extremely close. The one exception to this rule is the distributions that include zero, since these are truncated at zero. It is clear that in those cases, the expected value of x_i should be "pushed away" from zero, as can be seen in the figure.

Before we describe that method, we proceed to derive the M-step. It will turn out that both E- and M-step can be computed using the same basic recurrence.

3.4 The M-step

Now that we know something about the distribution of x, we can work on the expected log likelihood. First we write

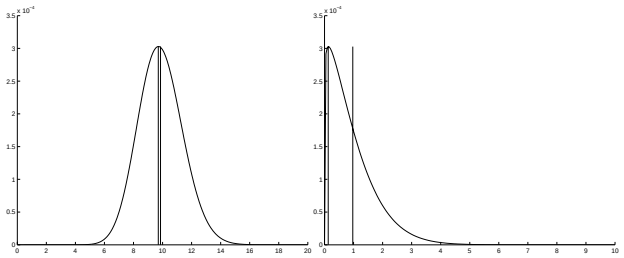

Figure 3: Sample computed pdfs for x_i given the document's observed term frequencies f. Vertical lines show mean and maximum likelihood values for x_i

Λ_j as the j^{th} row of Λ, so that $y_j = \Lambda_j x$. Next we expand each y_j about its expected value:

$$y_j = \overline{y_j}(1 + \Delta_j)$$

so that Δ_j is a random variable with mean zero. This will help us expand the log term in the expected log likelihood.

The next step is maximizing the expected log likelihood over Λ. Rewriting the log likelihood (1) with the above expansion, and ignoring the terms which are independent of Λ gives:

$$\sum_{j=1}^{n} (f_j \log(\overline{y_j}(1 + \Delta_j)) - \overline{y_j}(1 + \Delta_j)) \tag{2}$$

and taking expected values, using $E(\Delta_j) = 0$ and discarding terms of order 2 gives

$$\sum_{j=1}^{n} (f_j \log(\overline{y_j}) - \overline{y_j}) \tag{3}$$

now since $y_j = \Lambda_j X$, substituting for $\overline{y_j}$ into (3) gives:

$$\sum_{j=1}^{n} (f_j \log(\Lambda_j \overline{x}) - \Lambda_j \overline{x}) \tag{4}$$

which is now expressed in terms of the sufficient statistics $\overline{x}$. We can now differentiate the expected log likelihood $\overline{l}$ with respect to Λ_{ji} to find the maximum wrt Λ.

$$\frac{d\overline{l}}{d\Lambda_{ji}} = \frac{f_j}{\Lambda_j \overline{x}} \overline{x}_i - \overline{x_i} \tag{5}$$

and note that these equations are independent for distinct j. If we take the system of equations for all i, we obtain a self-contained system that defines the derivate wrt Λ_j:

$$g_{\Lambda_j} = \left(\frac{f_j}{\overline{y_j}} - 1 \right) \overline{x} \tag{6}$$

3.5 Recurrence Formulae

At this point, we observe that the equation for the log-likelihood (3) is similar to an entropy or KL-divergence measure. In [9], Lee and Seung considered a similar measure called "divergence" for non-negative matrix factorization. The measure they actually considered was

$$\sum_{j=1}^{n} - \left(f_j \log(\frac{\Lambda_j \overline{x}}{f_j}) - \Lambda_j \overline{x} + f_j \right) \tag{7}$$

while this measure is different from ours, if we differentiate it wrt Λ_j, we get

$$\frac{d\bar{l}}{d\Lambda_{ji}} = -\frac{f_j}{\Lambda_j \overline{x}}\overline{x}_i + \overline{x_i} \quad (8)$$

which is in fact the negative of the gradient of the log likelihood from equation (5). Since the derivatives agree up to sign, any algorithm which maximizes the log-likelihood of our model over Lambda is also minimizing the divergence between y and f. This is intuitively appealing. The frequencies f and the expected values y can be thought of as generalized probability distributions, except their sum is the document length rather than 1. The divergence above is minimized for the maximum likelihood model, and it naturally generalizes the KL-divergence for f and y.

Lee and Seung gave a simple, multiplicative recurrence for a matrix factorization $F = \Lambda X$ which minimizes the divergence. It can be applied directly to our M-step to compute Lambda. The recurrence for the **M-step** is

$$\Lambda_{ij} = \Lambda_{ij}\left(\sum_{k=1}^{n}\frac{F_{ik}}{\overline{Y}_{ik}}\overline{X}_{jk}\right) / \left(\sum_{k=1}^{n}\overline{X}_{jk}\right) \quad (9)$$

where F_{ik} and Y_{ik} are the actual and expected frequency of term term i in document k, and $\overline{X}_{jk}$ is the expected value of X_j for document $k \leq n$ total documents. Lee and Seung showed that this recurrence is globally convergent, and is guaranteed to find a unique minimum for Λ.

It turns out that our E-step can also be expressed as a minimization of divergence. Recall that the E-step should minimize:

$$\begin{aligned} l &= \textstyle\sum_{j=1}^{n}\left(f_j \log(\Lambda_j x) - \Lambda_j x\right) \\ &+ \textstyle\sum_{i=1}^{k}\left((a_i - 1)\log(x_i) - x_i a_i / c_i\right) \end{aligned} \quad (10)$$

which is equation (1) with substitutions for y_j and constant terms removed. And minimizing this expression over x is easily shown to be equivalent to minimizing the divergence in the following matrix factorization wrt x:

$$\begin{pmatrix} \Lambda \\ D_a D_c^{-1} \end{pmatrix} x = \begin{pmatrix} f \\ D_a - I \end{pmatrix} \quad (11)$$

where

$$D_a = \mathrm{diag}(a_1, \ldots, a_d) \qquad D_c = \mathrm{diag}(c_1, \ldots, c_d) \quad (12)$$

For a matrix factorization $F = \Lambda X$, Lee and Seung's formula for updates to X is:

$$X_{ik} = X_{ik}\left(\sum_{j=1}^{n}\frac{F_{jk}}{Y_{jk}}\Lambda_{ji}\right) / \left(\sum_{j=1}^{n}\Lambda_{ji}\right) \quad (13)$$

And when applied to the extended factorization (11), this becomes the **E-step**

$$X_{ik} = X_{ik}\left(\sum_{j=1}^{m}\frac{F_{jk}}{Y_{jk}}\Lambda_{ji} + \frac{a_i - 1}{X_{ik}}\right) / \left(\sum_{j=1}^{m}\Lambda_{ji} + \frac{a_i}{c_i}\right) \quad (14)$$

where X_{ik} is the magnitude of the i^{th} theme in document k, m is the number of terms, and other symbols are per their previous definitions. This recurrence is guaranteed to converge to a unique maximum for X. However, when the recurrences for E and M are combined, the iterations will converge but a global maximum is not guaranteed.

Notice that the numerator of (14) depends only on information about the current document, and that the number of terms in its sum is the number of actual terms in the document (not the total number of terms). The denominator on the other hand has a sum of Λ_{ji} over all terms, but the denominator is the same for all documents, so it only need be computed once. It follows that the total complexity of the E-step is $O(Nd)$ where N is the sum of the sizes of all documents (total corpus size) and d is the dimension, or number of themes.

The numerator of the M-step (9) also requires $O(Nd)$ steps to compute, as does its denominator. It would be tempting to alternate 1 E-step and 1 M-step, but there is more than arithmetic complexity at play. Because updating the X value for one document in the E-step requires only data that is local to one document, it has a very small memory footprint and easily fits in a CPU cache, usually in the L1 (fastest) cache. On the other hand, updating Λ requires a pass over the entire document set and is in practice much more expensive. We have found that it is economical to perform 8-20 E-steps (equation (14)) for each M-step (equation (9)). Finally, we note that the terms in the M-step formula are sums over n, the number of documents. Both numerator and denominator of the M-step can be computed incrementally during the E-steps for each document. Thus a combination of 8-20 E-steps and 1 M-step requires a single pass over the dataset.

If the corpus is small, it is possible to exploit a similar locality in the M-step recurrence, namely that an update to the row of Λ corresponding to a particular term depends only on the previous value of this row. However, this requires the matrix X of all X-values to be kept in memory. For large datasets, Λ is often smaller than X and if it isnt, the term set can be reduced by ignoring less-frequent terms in the model. Thus in our implementation, Λ resides in memory, and the X-values are saved on disk and read-written as a single pass.

3.6 Component Independence

So far we have described an EM algorithm for maximum likelihood estimation of Λ and X. We also argued that the GaP algorithm maximizes the independence between the factor components x_i. But why does this follow? The key is the choice of a well-fitting, highly-skewed prior on the x_i (the gamma distribution). We observed earlier that the shape parameter $a \geq 1$ for a gamma distribution is a measure of its skewed-ness. When $a = 1$, the distribution is an exponential distribution. As a increases, the distribution approaches a gaussian. From experiment, we have found that typical values for a are in the range 1.0-1.4.

Suppose that we explicitly knew identities of two independent components x_1 and x_2, meaning we knew the value of the corresponding columns of Λ from which we could derive the values x_1 and x_2 for all the documents. By our assumption (based on empirical data), x_1 and x_2 are well-fit by gamma distributions with a close to 1, and the same scale factor. Now consider any linear combination x_c of x_1 and x_2. It will be *less* skewed than either x_1 or x_2. For example, if it is a sum of x_1 and x_2 (which have the same scale factor), then x_c will have a scale factor greater than 2 which is much less skewed than x_1 or x_2. For a general combination, x_c can be decomposed into a gamma distribution with shape factor a, plus a residual distribution. When we compute the

likelihood for x_c using equation (1) with its gamma prior, it will be strictly lower than the likelihoods for either x_1 or x_2, whose distributions perfectly match the prior.

In general, any mixture of actual independent components with the specified gamma distributions will be less likely than the pure distributions. So the likelihood maximization will favor the most independent components. This is very similar to the popular technique of component analysis by kurtosis minimization from speech analysis [5], which has also been applied to ICA. Kurtosis is a general measure of "skewedness" of a distribution, and increases when signals are mixed. Minimizing it is a way of favoring unmixed components.

In practice, we do not know the actual shape factors a, nor are the priors on x_i's perfectly gamma-shaped. We could estimate the a's during the EM algorithm, but this is not a good idea. Suppose the algorithm estimates the a's at higher values (less skewed) than the independent components. Then it will be easy for the optimization to find a Λ which mixes several components in order to produce components that approximate the less-skewed distributions. In other words, the algorithm can make a bad guess but find a model that supports this guess.

A better approach is to introduce an independence "bias" by setting the shape parameters a to be deliberately smaller than their actual values. For instance, if actual shape parameters are in the range 1.2-1.4, model shape parameters are set at 1.1 (this has worked well in experiments). The more skewed the actual x_i distributions, the better they will fit the $a = 1.1$ priors. And the most skewed distributions will also be the most independent.

The negative effects of this deception are small. The components X_i during the algorithm will always be mixtures of actual independent components. There is no degree of freedom in the shape of those distributions per se, that the algorithm needs to find. Rather, it only needs to find the best (most independent) linear combinations. The deception causes a slight decrease in the log likelihood. But assuming average document length is much longer than the theme space dimension, then the output (poisson) term in the likelihood will dominate anyway.

4. IMPLEMENTATION

The GaP algorithm has been coded in both Matlab and native C++ code. Its performance is dominated by matrix operations, specifically matrix inversion, multiply and back-substitution. For efficiency, these operations should be handled by cache-aware, optimized Lapack routines. In Matlab, these operations are built-in. The Matlab GaP code has been heavily vectorized so its performance is dominated by the Lapack matrix primitives. The C++ version of GaP uses the Intel MKL (Math Kernel Library) implementation of Lapack. These libraries have very similar performance, and the result is that both Matlab and C++ versions of the GaP code have running times within a few percent of each other on most examples.

The main difference between C++ and Matlab versions is their use of memory. The Matlab code requires all matrices to be memory-resident. The C++ code allows the document-term and X matrices to be disk files, and makes multiple passes over them. The C++ code can handle a much larger number of documents. It also allows the set of terms processed to be restricted to terms with specified minimum frequency, which also reduces memory demand and allows the C++ version to handle arbitrarily large datasets.

Both Matlab and Intel MKL are proprietary software. If an open-source matrix library is desired, it is possible to use the Lapack routines from NIST (`gams.nist.gov`) which are available in both Fortran and C. However, most of these are not cache-aware and on modern computer hardware, one should expect an order of magnitude (factor of 10) or more loss in performance. There are some open-source cache-aware routines available. We experimented with Atlas (available from NIST) which did much better than naive Lapack routines, but was still much slower than MKL or Matlab. However, we did not tune the Atlas routines for the cache hierarchy of the PCs we were using which might have made up this difference. Another automatic-tuning library is PhiPac (Portable High Performance ANSI C), which is at `http://www.icsi.berkeley.edu/~bilmes/phipac/`.

5. EXPERIMENTS

We studied the performance of GaP in two dimensions: First as a language model, by measuring its fit (via perplexity) to a test corpus. A good point of comparison was to LDA (Latent Dirichlet Allocation) [1] which used the Cranfield dataset, and which included numerical perplexity values for various model dimensions. Perplexity is a standard measure of fit and is defined as

$$\text{Perplexity} = \exp\left(\sum_{\text{documents } d} \log p(d) \; / \sum_{\text{documents } d} N_d \right)$$

where d is a document from the corpus and N_d is its number of terms.

We tested two datasets: CRAN and the TREC queries 50-100 on the 300,000 document Tipster dataset. CRAN, the Cranfield dataset has 1400 documents and 3763 distinct terms. In [1], the corpus was divided randomly into training and test sets. We followed this approach, and our training set comprised 1300 documents while the test set comprised the remaining 100. A GaP model was created from the training set, and the model was used to evaluate document probabilities and hence perplexities for the remaining test documents. Both models were evaluated at various dimensions, as shown in figure 4. GaP exhibits lower perplexity (better fit) across the dimension scale, especially at low dimensions.

In a second experiment, we compared retrieval performance of GaP with two standard retrieval methods, tfidf and KL-divergence with Dirichlet smoothing on a unigram model [11]. The implementation of both methods came from the Lemur toolkit from CMU `http://www.cs.cmu.edu/~lemur`. The test dataset was again the Cranfield corpus

To apply GaP as a retrieval model, KL-divergence was used to measure distance from a query to each document. GaP can be used as a smoothing algorithm because it assigns non-zero probabilities to many terms not appearing in a document. In our experiments, we found that smoothing with GAP was sufficient by itself for large corpora (such as TREC data). Either Jelinek-Mercer or Dirichlet smoothing [11] can be used. We tested Jelinek-Mercer in our experiments.

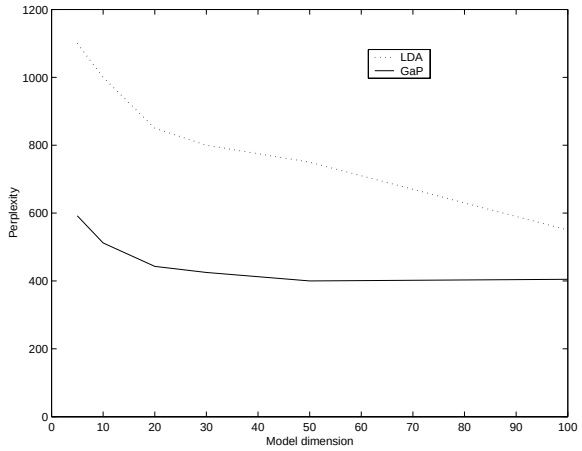

Figure 4: Perplexity results for Cranfield corpus

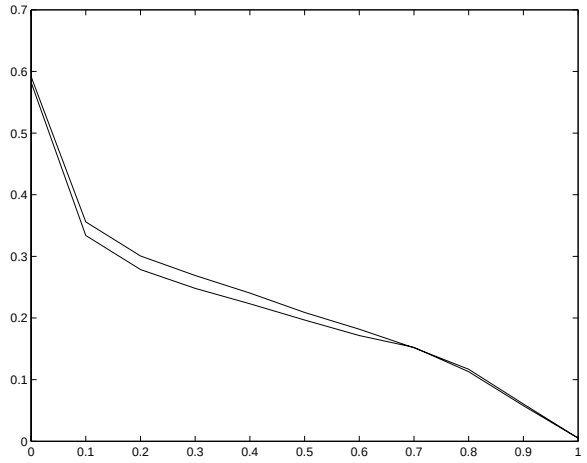

Figure 5: Retrieval results for TREC topics 51-100

The best results with TREC data were for a combination:

$$\sum_q n_q \log(p1(q) + 0.2 * p2(q))$$

where $p1$ is the observed probability of the term in the document (i.e. the term's relative frequency), and $p2$ is the GAP probability.

Figure 5 shows results for TREC topics 51-100 on the 300,000-document Tipster2 dataset and a 100-dimensional GAP model. The baseline curve uses Jelinek-Mercer smoothing with the same blend factor. That is, it uses the corpus probability for $p2$ instead of the GAP probability. The overall improvement using GAP is about 5%. The improvement was higher at 100 dimensions than at lower dimensions. We did not try models with more than 100 dimensions to see if this improvement continued. The running time to generate this model was about 4 hours on a 2.66 GHz Pentium.

For small corpora such as CRAN, it was necessary to use a secondary smoothing term from the probability computed from the GaP model, and relative frequency of the term in the corpus. For Cran, a good choice of probabilities

$$\sum_q n_q \log(p1(q) + 0.5 * p2(q) + 0.5 * p3(q))$$

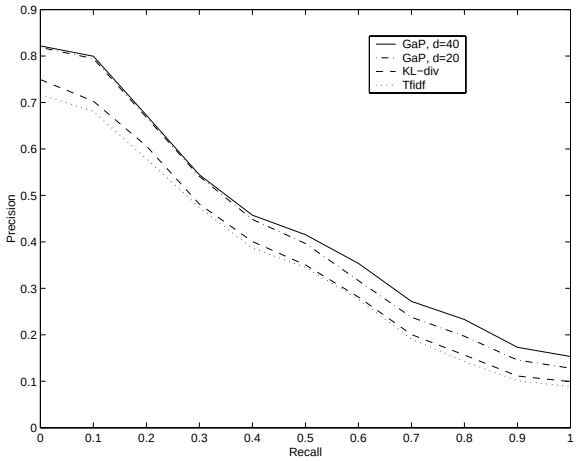

Figure 6: Retrieval results for Cranfield corpus

where q is a term appearing in the query, n_q is the number of occurences in the query, $p1$ is the term's empirical probability (relative frequency) in the document, $p2$ is the term's probability as computed by GaP, and $p3(q)$ is the probability of the term in the corpus (relative frequency in the corpus).

The results are shown in figure 6. The upper two plots are the results with GaP models of dimension 20 and 40 respectively. The lower two plots are KL-divergence with Dirichlet smoothing and tfidf. GaP gives consistently better results with either choice of dimension. Dimension 40 gives best results, and its average precision (0.42) is about 15% higher than KL-divergence or tfidf. Higher dimensions did not improve retrieval, as would be expected given the short document lengths in the Cranfield corpus.

The running time to compute a 40-dimensional GaP factorization for the Cranfield corpus was about 30 seconds on a 2.66 GHz Pentium. As we explained in the sections on E- and M-steps, the complexity is the product of corpus size, model dimension, and number of iterations. That is if the total corpus size is N (sum of number of terms in all documents), the model dimension is d, and the number of iterations is i, then the complexity is $O(Ndi)$.

6. FUTURE WORK

GaP shows promise for a variety of IR tasks. We believe it has applications outside of IR as well. In fact, GaP's raison d'etre is for activity analysis from logs of many users' actions when online. Specifically, email, IM, document and web access are all actions that users perform as part of collaborative knowledge work activities that extend across time and space. The goal of activity analysis is to recognize these longitudinal threads of activity, to cluster them and make predictions from them. GaP was developed to model such online activities, and ultimately real-world activities discovered from sensor data.

It is no accident that it performs well as a document analysis tool. The relationship between texts and "activity records" is a deep one which has been recognized by many influential linguists and literary theorists. Various versions of this connection appear in the works of Vygotsky, Leontev, Bakhtin and Burke. Burke's title "Language as Sym-

bolic Action" captures the idea well. A text is canonically a description of *sequences of actions* by various actors, in various times and places. The "activities" we seek to discover in activity logs are more than analogous to the "themes" we seek to discover in texts.

The great potential for GaP is where activity analysis and document analysis meet: in context-aware retrieval. GaP can be used simultaneously to mine discrete user actions involving texts (sending mail or reading documents), and the content of those texts themselves. The resulting term-based decomposition can be used to disambiguate search terms or even for pro-active (system initiated) retrieval.

6.1 Clustering

In the short term, we plan to test GaP as a clustering algorithm. Other researchers [10] have reported extremely good results with non-negative matrix factorization (NMF) of document corpora for clustering. GaP is such a model. It may have advantages over earlier NMF algorithms for reasons we have discussed earlier.

Acknowledgements

This work was supported in part by the National Science Foundation under NSF grant IIS-0222745.

7. REFERENCES

[1] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet Allocation. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*. MIT Press, 2002.

[2] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, Jan. 2003.

[3] T. Hofmann. Probabilistic Latent Semantic Indexing. In *ACM SIGIR Conference*, pages 50–57, 1999.

[4] A. Hyvärinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Trans. Neural Networks*, 10(3):626–634, 1999.

[5] A. Hyvärinen and E. Oja. Independent component analysis: Algorithms and applications. *Neural Networks*, 13(4–5):411–430, 2000.

[6] T. K. Landauer and S. T. Dumais. A solution to Plato's problem: The Latent Semantic Analysis theory of the acquisition, induction, and representation of knowledge. *Psychological Review*, 104:211–240, 1997.

[7] T. K. Landauer, P. W. Foltz, and D. Laham. Introduction to Latent Semantic Analysis. *Discourse Processes*, 25:259–284, 1998.

[8] T. K. Landauer, D. Laham, and P. W. Foltz. Learning human-like knowledge by singular value decomposition: A progress report. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10, pages 45–51. Cambridge: MIT Press, 1998.

[9] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *NIPS*, pages 556–562, 2000.

[10] W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *Proc. ACM SIGIR*, pages 267–273, August 2003.

[11] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Research and Development in Information Retrieval*, pages 334–342, 2001.