Mixture Models for Learning Low-dimensional Roles in High-dimensional Data

Manas Somaiya
University of Florida
Gainesville, FL, USA
manas@acm.org

Christopher Jermaine
Rice University
Houston, TX, USA
cjermain@rice.edu

Sanjay Ranka
University of Florida
Gainesville, FL, USA
ranka@cise.ufl.edu

ABSTRACT

Archived data often describe entities that participate in multiple roles. Each of these roles may influence various aspects of the data. For example, a register transaction collected at a retail store may have been initiated by a person who is a woman, a mother, an avid reader, and an action movie fan. Each of these roles can influence various aspects of the customer's purchase: the fact that the customer is a mother may greatly influence the purchase of a toddler-sized pair of pants, but have no influence on the purchase of an action-adventure novel. The fact that the customer is an action move fan and an avid reader may influence the purchase of the novel, but will have no effect on the purchase of a shirt.

In this paper, we present a generic, Bayesian framework for capturing exactly this situation. In our framework, it is assumed that multiple roles exist, and each data point corresponds to an entity (such as a retail customer, or an email, or a news article) that selects various roles which compete to influence the various attributes associated with the data point. We develop robust, MCMC algorithms for learning the models under the framework.

Categories and Subject Descriptors

I.5.1 [**Pattern Recognition**]: Models—*Statistical*

General Terms

Algorithms, Theory

Keywords

High-Dimensional Data, MCMC, Mixture Models

1. INTRODUCTION

Collected data often describe entities that are generated via complex interactions among multiple aspects of each entity, or roles that an entity may play. For example, consider a data set of customer transactions at a movie rental store. In such a domain, we would expect a customer to belong to multiple and possibly overlapping customer classes such as *male*, *female*, *teenager*, *adult*, *parent*, *action-movies-fan*, *comedy-movies-fan*, *horror-movies-fan*, etc. Membership in each customer class affects the movie rentals selected by the customer, and the effect of each customer class is more or less significant, depending on the data attribute (movie) under consideration. Consider a customer who is both a *parent* and an *action-movies-fan*. One can imagine that the *parent* class is more influential than the *action-movies-fan* class when the customer decides whether or not to rent the animated movie *Teenage Mutant Ninja Turtles*.

Modeling Overlapping Relationships. Unfortunately, most models for learning patterns from data do not respect the fact that various unseen or hidden roles of each entity can interact in such a complex and overlapping fashion, as described above.

One of the common ways to model such multi-class data is a so-called *mixture model* [14, 15]. A classical mixture model for our movie-rental example would view the data set as being generated by a mixture of customer classes, with each class modeled as a multinomial component in the mixture. Under such a model, when a customer enters a store she chooses one of the customer classes by performing a multinomial trial according the mixture proportions, and then a random vector generated using the selected class would produce the actual rental record.

The problem with such a model in many application domains is that it only allows one model component (customer class) to generate a data point (rental record), and thus does not account for the underlying data generation mechanism that a customer belongs to multiple classes. In this paper, we propose a new class of mixture models called *Power models*, as well as associated learning algorithms. In a Power model, an arbitrary set of components can contribute to the generation of a single data point, and each component can have a varying degree of influence on different data attributes. We choose the name "Power model" for two reasons. First, "Power" is a descriptive acronym: *PrObabilistic Weighted Ensemble of Roles*. Second, each data point under a Power model is produced by a mixture that is itself sampled from the power set of all of the model components, as we discuss now.

Data Generation Under Power Model. As in a classic mixture model, each class in a Power model has a unique appearance probability that indicates the prevalence of this class in the data set. However, rather than class appearance being modeled as a multinomial process, it is controlled via a Bernoulli process, so that multiple classes can influence a single data point.

Under the Power generative process, when our customer enters the rental store, she chooses some of the customer classes by flipping a biased coin for each of the customer classes – the bias is controlled by the class' associated appearance probability. A heads result on the i^{th} trial means that the i^{th} customer class is associated

KDD'10, July 25–28, 2010, Washington, DC, USA.

with the customer. We refer to these selected classes as the *active* classes, and the customer's action is controlled via a mixture of the active classes.

Assume that the various coin flips produce a customer who is an *action-movies-fan*, a *horror-movies-fan*, and a *parent*. Now, let us assume that she is trying to decide if she wants to rent the movie *Teenage Mutant Ninja Turtles*. To determine which of active classes is used to make this rental decision, we perform a weighted multinomial trial using weight parameters for this movie. Each of the active customer classes will supply a weight, which indicates this class' "desire" to control whether or not the customer rents the movie. The total weight that a class may distribute to the various attributes is finite, so that the class must decide which attributes are most important to it.

Assume that the weights of the active customer classes for this movie are $w_{tmnt,action}$, $w_{tmnt,horror}$, and $w_{tmnt,parent}$ respectively. Then (for example) the class *action-movies-fan* has a

$$\frac{w_{tmnt,action}}{w_{tmnt,action} + w_{tmnt,horror} + w_{tmnt,parent}}$$

probability of being selected as the generating class for this movie. The *dominant* class for this movie is then selected via a multinomial trial. Assume that the customer class *parent* is selected. Then the final decision for renting this movie will be based on the probability that a customer who is a parent will pick *Teenage Mutant Ninja Turtles* as a rental.

This type of model has several advantages. Compared to classic mixture models, this model allows multiple components to act together in generation of a data point. This allows for very generic and intuitive classes (such as *horror-movie-fan*, *action-movie-fan*, *cartoon-fan*, etc., in our example) while still allowing us to precisely model specific data points. In other words, our model can naturally (and realistically) describe someone who simultaneously rents movies as diverse as *Scooby Doo* and *The Ring* – this person is a *horror-movie-fan* and also a *parent*. A classic mixture model would need to use a single class ("horror-movie-fan-parent") to describe such a person, leading to overly complex classes.

The Power framework is generic, and not tied to a specific application. In fact, the framework prescribes only how the various model components interact with one another and compete to influence the data attributes. Application-specific distributions or stochastic processes for generating the actual data attributes can be chosen as appropriate. Also, since the model assumes that data attributes are independent of each other, it trivially allows mixing of various data types and associated distributions.

Paper Organization. The next section describes the Power framework in detail. Section 3 discusses a Gibbs Sampler that can be used to learn a Power model from a data set. Section 4 discusses how we speed up certain parts of our Gibbs Sampler. Section 5 details some example applications of the Power model, Section 6 discusses related work, and Section 7 concludes the paper.

2. MODEL

In this section, we formally describe the Power model. Let $X = \langle x_1, x_2, \cdots, x_n \rangle$ be the dataset, where each x_a takes the form $x_a = \langle x_{a,1}, x_{a,2}, \cdots, x_{a,d} \rangle$. The value $x_{a,j}$ is assumed to have been sampled from a random variable A_j corresponding to the j^{th} "attribute", having a parameterized probability density function f_j. "Attribute" is in quotations here because it is possible for A_j to be vector-valued to facilitate correlated attribute values.

The proposed model consists of a mixture of k components $C = \{C_1, C_2, \cdots, C_k\}$. Associated with each component C_i is an appearance probability α_i. Each component C_i has an associated d-dimensional parameter vector Θ_i that parameterizes the probability density function f_i. Each component specifies the strength of its influence on various data attributes using a vector of positive real numbers $W_i = \langle w_{i,1}, w_{i,2}, \cdots, w_{i,d} \rangle$. We call these the "parameter weights"; and $\sum_j w_{i,j} = 1$.

2.1 Generative Process

Given this setup, each data point x_a is generated by the following, three-step process:

- First, one or more of the k components are marked as "active" by performing a Bernoulli trial, where the probability of C_i being active is α_i.
- Second, for each attribute a "dominant" component is selected by performing a weighted multinomial trial among the active components. If C_i is active, then the probability that C_i is dominant for attribute j is proportional to $w_{i,j}$.
- Finally, each data attribute is generated. If C_i is dominant for attribute j, then the j^{th} attribute is generated according to density function f_i.

Since we use Bernoulli trials for selection of active components, there is a non-zero probability that none of the components will become active. To ensure that at least one component is always present and to provide a background probability distribution for the mixture model, we make one of the k components a special "default" component. The default component is active for all data points; that is, the appearance probability of the default component is set to be 1. The default component provides the background parameters that are used when no other, interesting component is actually present. Going back to our video-store example, the background component might correspond to a generic customer who enters the store, but who is not a *parent* or an *action-movies-fan* or a member of any other of the classes.

Furthermore, we want the default component to become a dominant component for any attribute only when no other component is active. Assume that C_i is the default component. This can be achieved by forcing $w_{i,j_1} = w_{i,j_2}$ for every j_1 and j_2; that is, the background component is not allowed to favor any particular attribute. Furthermore, each $w_{i,j}$ is set to be ϵ, for very small constant ϵ. By increasing or decreasing the value of ϵ, the user can make default's influence stronger or weaker as compared to other active components, and thus limit or strengthen its role in the model.

2.2 Bayesian Framework

To allow for a learning algorithm, the model parameters are generated in a hierarchical Bayesian fashion. A plate diagram showing the outline of the relationships of all the variables and their explanation can be seen in Figure 1. For each component other than the default, we start by assigning a Beta prior with user-defined parameters u and v for each of the appearance probabilities α_i associated with component C_i:

$$\alpha_i | u, v \quad \sim \quad \beta(\cdot | u, v) \quad i = 1 \cdots k$$

The parameter weights W_i in the model are simulated by normalizing positive real numbers called *mask values* and denoted by M_i. We assign a Gamma prior with user-defined parameters q and r for the mask vector values $m_{i,j}$:

$$\begin{aligned} m_{i,j} | q, r \quad &\sim \quad \gamma(\cdot | q, r) \quad i = 1 \cdots k,\ j = 1 \cdots d \\ w_{i,j} \quad &= \quad \frac{m_{i,j}}{\sum_j m_{i,j}} \end{aligned}$$

Variable	Explanation
n	number of data points
d	number of data attributes
k	number of components in the model
α	appearance probability of a component
u, v	user defined prior parameters for α
m	mask values
q, r	user defined prior parameters for m
w	parameter weights
c	indicator variable for active components
g	indicator variable for dominant component
μ	mean values for the parameterized p.d.f.
μ_a, μ_b	user defined prior parameters for μ
σ	std dev values for the parameterized p.d.f
σ_a, σ_b	user defined prior parameters for σ
x	data variable

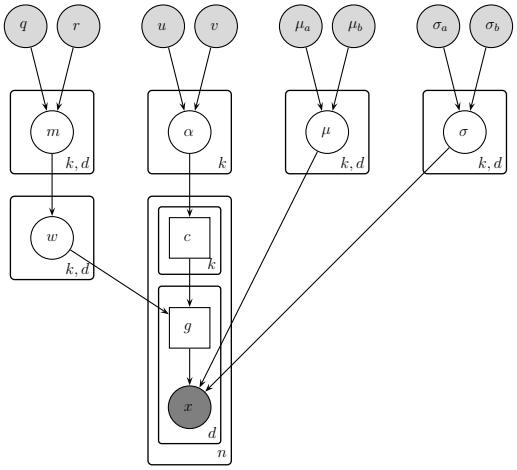


Figure 1: Generation process for the Power family of models. Light gray nodes are user-defined constants. Unshaded nodes are hidden random variables, while dark gray shaded nodes are observed random variables. A circular node indicates a continuous value, while a square node indicates a discrete value. An explanation of the various random variables and constants can be found in the table.

To generate a data point, first one or more of the k components are marked as "active" by performing a Bernoulli trial with their appearance probabilities. Let $\overrightarrow{c_a}$ be the hidden random variable that indicates active components for data point x_a. Then,

$$c_{a,i}|\alpha_i \quad \sim \quad Bernoulli(\cdot|\alpha_i) \quad i = 1 \cdots k$$

Next, for each attribute a "dominant" component is selected by performing a weighted multinomial trial (using the parameter weights) amongst active components. Let $e_{a,j}$ be the sum of weights, and let $g_{a,j}$ indicates the selected dominant component for the jth attribute from the active components for data point x_a. We have,

$$\begin{aligned} e_{a,j} &= \sum_{i=1}^{k} c_{a,i} \cdot w_{i,j} \quad a = 1 \cdots n,\ j = 1 \cdots d \\ f_{a,j,i} &= \frac{c_{a,i} \cdot w_{i,j}}{e_{a,j}} \quad a = 1 \cdots n,\ j = 1 \cdots d,\ i = 1 \cdots k \\ g_{a,j} &\sim Multinomial(1, \overrightarrow{f_{a,j}}) \quad a = 1 \cdots n,\ j = 1 \cdots d \end{aligned}$$

For the ease of explanation, we will assume throughout the rest of the paper that all data attributes are generated by a normal probability density function (i.e. Gaussian) generators. However, in general our framework is data-type agnostic, and one can use any probabilistic data generator. So, in the final step of data generation, each data attribute is generated using the parameterized normal distribution by using the parameters from its dominant component:

$$x_{a,j} \quad \sim \quad N(\cdot|\mu_{g_{a,j},j}, \sigma_{g_{a,j},j}) \quad a = 1 \cdots n,\ j = 1 \cdots d$$

In the normal case, the mean and the standard deviation parameters can be assigned a non-informative inverse gamma priors with parameters μ_a and μ_b, and σ_a and σ_b respectively.

$$\mu_{i,j} \quad \sim \quad IG(\cdot|\mu_a, \mu_b) \quad i = 1 \cdots k,\ j = 1 \cdots d$$

$$\sigma_{i,j} \quad \sim \quad IG(\cdot|\sigma_a, \sigma_b) \quad i = 1 \cdots k,\ j = 1 \cdots d$$

3. LEARNING THE MODEL

Bayesian inference for the proposed model can be accomplished via a Gibbs sampling algorithm. Gibbs sampling is a very widely used method to generate samples from a joint probability distribution of many random variables. It is particularly useful when it is hard to sample from the joint probability distribution itself but very easy to sample from the conditional distributions of the random variables. Starting from a random initialization, Gibbs sampling is an iterative process, where in each iteration, we consecutively update the value of each random variable by drawing a sample from its conditional distribution w.r.t. all other random variables. Thus, Gibbs Sampler is actually a Monte Carlo Markov Chain, and it is generally accepted that after numerous iterations, the chain reaches the steady state where the samples actually closely approximate the joint probability distribution of the random variables. For a detailed formal description and analysis of Gibbs Sampling we direct the reader to the excellent textbook by Robert and Casella [20].

Applying a Gibbs sampling algorithm requires derivation of conditional distributions for the random variables. Next, we outline this derivation for all the random variables in the proposed model.

3.1 Appearance Probability α

Starting with Bayes rule, the conditional distribution for appearance probability is given by

$$F(\alpha|X, g, m, \mu, \sigma, c) = \frac{F(\alpha, X, g, m, \mu, \sigma, c)}{F(X, g, m, \mu, \sigma, c)}$$

which can be reduced to,

$$F(\alpha|X, g, m, \mu, \sigma, c) \propto F(c|\alpha) \cdot F(\alpha)$$

Hence, it is clear that the value of the appearance probability α_i can be updated by just using $c_{\star,i}$ and the prior $F(\alpha)$.

$$F(\alpha_i|X, g, m, \mu, \sigma, c) \propto \beta(\alpha_i|u, v) \cdot \prod_a F(c_{a,i}|\alpha_i)$$

Now, $F(c_{a,i}|\alpha_i) = \alpha_i$ if component C_i is active for data point x_a, and $F(c_{a,i}|\alpha_i) = 1 - \alpha_i$ otherwise. Hence, if $nactive_i$ is the count of all data points for which C_i is active, then $n - nactive_i$ is the count of all data points for it is inactive. Using these values, we can further simply the conditional to be

$$F(\alpha_i|X, g, m, \mu, \sigma, c) \propto \beta(\alpha_i|a, b) \cdot \alpha_i^{nactive_i} \cdot (1-\alpha_i)^{n-nactive_i}$$

Based on this conditional distribution, it is fairly straight forward to setup a rejection sampling scheme for α_i.

3.2 Active Components Indicator Variable c

Starting with Bayes rule, the conditional distribution for the active components indicator variable is given by

$$F(c|X,g,m,\sigma,\mu,\alpha) = \frac{F(c,X,g,m,\sigma,\mu,\alpha)}{F(X,g,m,\sigma,\mu,\alpha)}$$

which can be reduced to,

$$F(c|X,g,m,\sigma,\mu,\alpha) \propto F(g|c,m) \cdot F(c|\alpha)$$

Hence, the active component indicator variable $c_{a,i}$ can be updated based on values of dominant component indicator variables $g_{a,\star}$, mask values m, and appearance probability α_i. We observe that for a particular dimension j, the value of $g_{a,j}$ depends not only any single $c_{a,i}$ but on all of them. Hence, we need to perform block updates for all $c_{a,\star}$. Block updates [12, 13] are typically performed to update a set of correlated variables together.

Note that there are only two possible values for $c_{a,i}$ – either 1(active) or 0(inactive). If any $g_{a,j} = i$ i.e. the i^{th} component generated the $x_{a,j}$ then we can conclude that it is active.

$$\text{if} \quad \exists j,\, g_{a,j} = i \quad \text{then} \quad c_{a,i} = 1$$

However, if there is no such $g_{a,j} = i$ for all j, then we have to look at both the possibilities—the component was active but did not dominate any single attribute or the component was inactive,

$$\begin{aligned} F(c_{a,i}=0|X,g,m,\sigma,\mu,\alpha) &\propto F(c_{a,i}=0|\alpha_i) \\ &\quad \cdot \prod_j F(g_{a,j}|c_{a,\star}, c_{a,i}=0, m) \\ F(c_{a,i}=1|X,g,m,\sigma,\mu,\alpha) &\propto F(c_{a,i}=1|\alpha_i) \\ &\quad \cdot \prod_j F(g_{a,j}|c_{a,\star}, c_{a,i}=1, m) \end{aligned}$$

where,

$$\begin{aligned} F(c_{a,i}=1|\alpha_i) &= \alpha_i \\ F(c_{a,i}=0|\alpha_i) &= 1-\alpha_i \\ F(g_{a,j}|c_{a,\star},m) &= \frac{w_{g_{a,j},j} \cdot I(c_{a,g_{a,j}}=1)}{\sum_i w_{i,j} \cdot I(c_{a,i}=1)} \\ w_{i,j} &= \frac{m_{i,j}}{\sum_j m_{i,j}} \end{aligned}$$

Here, $I()$ is an indicator function that evaluates to 1 if the logical argument provided to it is true, and 0 otherwise. After evaluating the posterior distributions, we can update $c_{a,i}$ by performing a Bernoulli trial based on those values.

3.3 Dominant Component Indicator Var g

Starting with Bayes rule, the conditional distribution for dominant component indicator variable is given by

$$F(g|X,c,m,\sigma,\mu,\alpha) = \frac{F(g,X,c,m,\sigma,\mu,\alpha)}{F(X,c,m,\sigma,\mu,\alpha)}$$

which can be reduced to,

$$F(g|X,c,m,\sigma,\mu,\alpha) \propto F(X|g,\mu,\sigma) \cdot F(g|c,m)$$

Hence, it is clear that the dominant component indicator variable $g_{a,j}$ can be updated based on values of active component indicator variables $c_{a,\star}$, mean and standard deviation parameters μ and σ, and mask values m.

$$\begin{aligned} F(g_{a,j}|X,c,m,\sigma,\mu,\alpha) &\propto F(x_{a,j}|g_{a,j},\mu,\sigma) \\ &\quad \cdot F(g_{a,j}|c_{a,\star},m) \\ F(g_{a,j}=i|X,c,m,\sigma,\mu,\alpha) &\propto N(x_{a,j}|\mu_{g_{a,j},j},\sigma_{g_{a,j},j}) \\ &\quad \cdot F(g_{a,j}=i|c_{a,\star},m) \end{aligned}$$

where,

$$\begin{aligned} F(g_{a,j}=i|c_{a,\star},m) &= \frac{w_{i,j} \cdot I(c_{a,i}=1)}{\sum_i w_{i,j} \cdot I(c_{a,i}=1)} \\ w_{i,j} &= \frac{m_{i,j}}{\sum_j m_{i,j}} \end{aligned}$$

So this becomes a simple multinomial trial with probabilities proportional to posterior distribution for each possible value of $g_{a,j}$.

3.4 Mask Value m

Starting with the Bayes rule, the conditional distribution for mask values is given by

$$F(m|X,g,\mu,\sigma,c,\alpha) = \frac{F(m,X,g,\mu,\sigma,c,\alpha)}{F(X,g,\mu,\sigma,c,\alpha)}$$

which reduces to,

$$F(m|X,g,\mu,\sigma,c,\alpha) \propto F(g|c,m) \cdot F(m)$$

Hence, mask value $m_{i,j}$ can be updated based on values of dominant component indicator variables g, active component indicator variables c, all other mask values m, and the prior $F(m)$. Note, that changing any mask value $m_{i,j}$ has an impact on all parameter weights $w_{i,\star}$, and hence the dependence on all g and c random variables. Based on this, we can write:

$$\begin{aligned} F(m_{i,j}|X,c,m,\theta,\alpha) &\propto \gamma(m_{i,j}|q,r) \\ &\quad \cdot \prod_a \prod_j \frac{w_{g_{a,j},j} \cdot I(c_{a,g_{a,j}}=1)}{\sum_i w_{i,j} \cdot I(c_{a,i}=1)} \end{aligned} \tag{1}$$

where,

$$w_{i,j} = \frac{m_{i,j}}{\sum_j m_{i,j}}$$

Based on this conditional distribution, it is fairly straight forward to setup a rejection sampling scheme for $m_{i,j}$. Updating the corresponding parameter weights $w_{i,\star}$ is then a simple arithmetic exercise.

3.5 Mean and StdDev Parameters μ and σ

It is fairly straightforward to derive conditional distributions and setup the rejection sampling for both the Gaussian parameters. We skip the details because of space constraints.

4. SPEEDING UP THE MASK UPDATES

Because conjugate priors are not available for the Power model, our Gibbs sampler implementation makes use of a rejection sampler [16] to sample from each of the marginals.

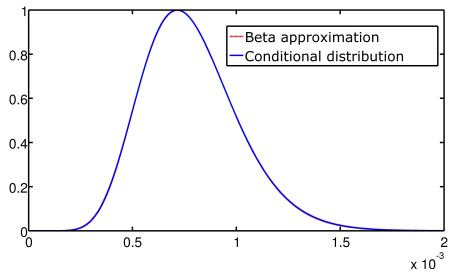
Unfortunately, our initial parallelized C implementation was very slow, despite of running on a 32-core machine. For example, it took five hours to run each iteration over the Reuters data set (described in the next section). This data set is not trivial (it has tens of thousands of observations and a thousand attributes), but five hours per iteration is too long. Coupled with the fact that one might run a Gibbs sampler for a few thousand iterations to allow for a "burn

Table 1: Details of the datasets used for qualitative testing of the beta approximation. n is number of data points, d is number of attributes, and k is number of components in the model.

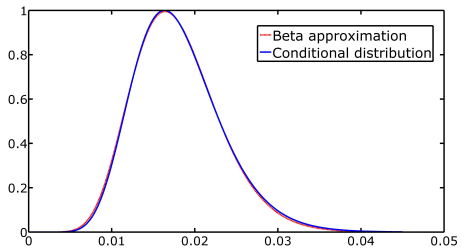
Id	Type	Generators	n	d	k
1	Real-life	Normal	500	80	5
2	Synthetic	Normal	1000	4	5
3	Synthetic	Bernoulli	2000	4	5
4	Real-life	Bernoulli	2800	41	10

Table 2: Quantitative testing of the beta approximation. $k\#$ and $d\#$ are respectively the component and data attribute under consideration.

Id	Iteration #	k #	d #	KL quad	KL discrete
1	10	2	3	0.000000	0.066571
2	50	3	1	0.000000	0.234133
3	20	2	4	0.000000	0.000001
4	5	7	21	0.000009	0.363120



(a) Dataset 1



(b) Dataset 4

Figure 2: Comparison of the PDFs for the conditional distribution of the weight parameter with its beta approximation for two real-life datasets. Each chart is normalized for easy comparison and has been zoomed-in to the region where the mass of the PDFs are concentrated. Details about the datasets can be found in Tables 1 and 2.

in" [18], this equates to roughly 15000 hours (520 days) of compute time.

A profile of our code determined it spent the vast majority of its time updating the mask values (that is, the various $m_{i,j}$ values). To understand where the computational difficulty in updating the mask values lies, consider the conditional distribution for $m_{i,j}$, given as Equation 1. Evaluating the right-hand-side (RHS) of the equation is an $O(n \cdot d)$ operation. The RHS must be evaluated at least once per iteration of the rejection sampler. If an average of z iterations are required to run the rejection sampler, given that there are $k \cdot d$ mask values, the overall complexity for updating all mask values is $O(z \cdot n \cdot k \cdot d^2)$. We find that typically, $z \approx 50$. We also note that n, k and d are fixed properties of the model and the data, but z is not. Thus, reducing z seemed to be a very obvious way to reduce the overall running time.

We accomplish this via an approximation of the conditional for $m_{i,j}$. Since $F(m_{i,j})$ measures the (non-normalized) weight that component i assigns to its j^{th} attribute, it was very natural to approximate this conditional using a Beta distribution. The Beta is the usual distribution used to model unknown probabilities (or weights), and it allows for a wide variety of distributions (bi-modal, uniform, very peaked). Furthermore, assuming a Beta, we need only three computations of the conditional in order to fit values for each of the three Beta parameters—the two shape parameters and one scale parameter.

To obtain these three parameters, we sample the value of the conditional distribution and obtain three different $F(m_{i,j}|\cdot)$ values and solve for the three Beta parameters analytically. We can then sample $m_{i,j}$ directly from the approximate conditional, with the only real cost being three evaluations of the conditional to obtain the approximation. If we fail to obtain a valid Beta distribution (that is, the resulting shape parameters are invalid), then we default to rejection sampling. In practice, this whole process speeds up our implementation by a factor of ten.

Of course, this can introduce approximation error. A detailed analytic or experimental study of the resulting error is beyond the scope of the paper, but we do outline a brief qualitative study of the approximation using two synthetic and two real-life datasets. As outlined in Table 1, the synthetic datasets consist of four dimensional, real-valued and zero-one data; we also considered a real-life data set that tracks California river levels, and one that tracks the movement of the S&P 500 (see the next section for details). We have examined a large number of the conditional approximations obtained over these data sets, and found an almost perfect match each time. In Figure 2, we graphically compare one of the Beta approximations from the two real-life data sets with the true conditional. In Table 2, we show the KL-divergence [11] between the two distributions, computed using both Matlab's built-in quadrature, and a simple discretization scheme. In each of these cases, the approximation is very accurate.

5. EXPERIMENTS

In this section, we show the learning capabilities of our model both on synthetic and real-world data. The learning algorithm for the real-life data sets was written in C, and was run on a workstation with eight quad-core AMD Opteron processors operating at 1.8GHz with 128GB RAM. Parts of the code were parallelized to make use of multiple cores.

5.1 Synthetic Data Set

In this section we outline a simple experiment where we apply the Power model's learning algorithm to synthetic data. The goal of this experiment is to perform a sanity check on our learning algorithms. To this end, we generate a simple data set using known parameters, and see if the learning algorithms can accurately recover the parameters used for data generation.

Experimental setup. We generated a 1000-record, four-component, four-attribute data set under the Power generative model, as outlined in Table 3, and then ran the MCMC learning algorithms over the resulting data. To initialize the learner, all the mean and standard deviation parameters were initialized to the mean and the standard deviation of the data set. The weight for each attribute was initialized to $\frac{1}{4}$, so as provide same influence over all data attributes.

Table 3: The four components used to generate the synthetic data set. The generator for each attribute is expressed as a triplet of parameter values, given as (Mean, Standard deviation) [Weight]. α is the appearance probability of a component.

α	A1	A2	A3	A4
0.2492	(300,20)	(600,20)	(900,20)	(1200,20)
	[0.4]	[0.1]	[0.1]	[0.4]
0.2528	(600,20)	(900,20)	(1200,20)	(300,20)
	[0.1]	[0.4]	[0.4]	[0.1]
0.2328	(900,20)	(1200,20)	(300,20)	(600,20)
	[0.4]	[0.1]	[0.4]	[0.1]
0.2339	(1200,20)	(300,20)	(600,20)	(900,20,0.4)
	[0.1]	[0.4]	[0.1]	[0.4]

Table 4: Parameter values learned from the data set after 1000 Gibbs iterations.

α	A1	A2	A3	A4
0.3284	(298,20.4)	(600,19.4)	(900,19.7)	(1201,19.6)
	[0.37]	[0.11]	[0.10]	[0.43]
0.3658	(600,19.8)	(901,20.3)	(1200,19.0)	(299,19.2)
	[0.10]	[0.42]	[0.38]	[0.10]
0.3286	(898,20.8)	(1197,20.8)	(303,21.2)	(599,19.6)
	[0.45]	[0.09]	[0.33]	[0.13]
0.3201	(1201,20.4)	(300,19.7)	(598,19.1)	(900,20.5)
	[0.12]	[0.40]	[0.10]	[0.39]

The weight for the default component ϵ was set to be one-hundredth of the initial weight for each attribute. The parameters u and v controlling the prior for the appearance probability were set to 100 and 300 respectively. The prior parameters q, r, μ_a, mu_b, σ_a, and σ_b were all initialized to 1 so as to be non-informative. We ran our Gibbs Sampling procedure for 1000 iterations, allowing for a burn-in of 900 iterations. The average values of the model parameters over the last 100 iterations are shown in Table 4.

Discussion. It would be fair to say that the learning algorithm has successfully recovered all model parameters in this simple experiment.

One slight anomaly is that the learned values for appearance probability are all somewhat higher than what is found in the original generators. The explanation for this is that many (or even most) data points that were in fact generated using the default component could also be explained as having come from all of the the four generators working together, since the range of attribute values that are covered by the four components together nicely spans set of attribute values that could be produced by the default. Because some data points that were actually created by the default are instead assumed by the learner to have been created by all four components working together, each individual component is assigned a slightly higher appearance probability than it actually deserves.

5.2 NIPS Papers Data Set

In this subsection, we show how we can use the Power model to learn patterns in high-dimensional, real-life data. Specifically, we consider the widely-used NIPS papers data set available from the UC Irvine Machine Learning Repository [3]. The selection of this data set was motivated by the fact that correlations amongst words in NIPS subareas are intuitive, easy to understand, and well-studied. Thus, it would be easy to observe and discuss the patterns found by the model.

Experimental setup. The NIPS full papers data set consists of words collected from 1500 papers. The vocabulary covers 12419 words, and a total of approximately 6.4 million words can be found in the papers. We considered the top 1000 non-trivial words. Each paper was converted to a row of zeros and ones corresponding to the absence and presence of the word, respectively. Thus, we obtain a 0/1 matrix of size 1500 by 1000. This kind of data is naturally modeled using components that are arrays of Bernoulli variables, with each Bernoulli corresponding to a word.

We attached a weak beta prior $\beta(1, 1)$ to each Bernoulli component. The prior parameters u, v, q and r that control the appearance probability and parameter weights were set to 1 so as to be non-informative. We set the number of components in the model to be 21, and the weight for the default component ϵ was set to be the same as the initial weight for each attribute i.e. $\frac{1}{1000}$. We ran the Gibbs Sampling procedure for 2000 iterations. Allowing for a burn-in period of 1000 iterations, we collected samples over the last 1000 iterations. The average for all model parameters over these iteration can be found in a separate technical report [22], and we do not report them here for lack of space.

On our 32-core machine, each iteration of the Gibbs sampler took approximately 65 seconds to complete, leading to an overall learning time of 36 hours.

Post-processing. One of the key aspects of our model is that each component assigns an importance to each of the data set's attributes—there are certain attributes that it has a very strong influence on, and others where its influence is quite weak. Ideally, one would like to be able to look at a particular component (each of which has 1000 dimensions in this experiment), and rank all of the attributes according to the attribute's level of importance for the component. Looking at only the top few attributes in each component should give a very good idea of the sort of roles that are present in the data.

To accomplish this, one could simply rank the various attributes based upon the weights assigned to each attribute by the component, but this would not necessarily tell the whole story. Simply because a component C_i weighs a particular attribute lightly, does not mean that the attribute is not vital to component C_i. If all other components weigh the attribute far *more* lightly than component C_i, and if active it will still have the final say in the generation of the attribute. It does not make sense for component C_i to "waste" any of its total weight budget trying to increase its influence on that attribute, when the component C_i's influence on the attribute is almost dominant anyway.

Thus, when visualizing a component, we rank the various attributes as follows. For component C_i, let $p_{in}(j)$ denote the probability that word j is present if C_i is active, and let $p_{ex}(j)$ denote the probability that word j is present if C_i is not active. These probabilities can be computed easily using a simple Monte Carlo process. Note that relative values of $p_{in}(j)$ and $p_{ex}(j)$ are a very good measure of how important word j is to component C_i. For example, $p_{ex}(j) = 0.5$ but $p_{in}(j) = 0.9$ clearly indicates that the presence of this component drastically increases the chance of word j appearing in a document. However, if $p_{ex}(j) = 0.5$ but $p_{in}(j) = 0.51$, then it indicates that the presence of this component does not drastically alter the chances of this word appearing in a document. Then, to determine the relative importance of word j to component C_i, we compute the KL-divergence between the two Bernoulli distributes induced by $p_{in}(j)$ and $p_{ex}(j)$:

$$kl(j) = p_{in}(j) \cdot \log \frac{p_{in}(j)}{p_{ex}(j)} + (1 - p_{in}(j)) \cdot \log \frac{(1 - p_{in}(j))}{(1 - p_{ex}(j))}$$

Note that this ranking does not apply only to Bernoulli models. It can be extended to any generative model by computing the KL-

Table 5: The most highly-ranked words for some of the components learned from the NIPS dataset. Plain text indicates that the component tries to enhance the likelihood that the word appears—that is, it assigns a high importance to the word, as well as a high Bernoulli probability. Bold text indicates that the component tries to suppress the word—it assigns a high importance but a low Bernoulli probability.

id	Words
1	arbitrary, assume, asymptotic, bound, case, consider, define, exist, implies, proof, theorem, theory
3	acoustic, amplitude, auditory, channel, filter frequency, noise, signal, sound, speaker, speech
5	activity, brain, cortex, excitatory, firing, inhibition, membrane, neuron, response, spike, stimuli, synapse
6	activation, backpropagation, feedforward, hidden, input, layer, network, neural, output, perceptron, training
7	adaptive, control, dynamic, environment, exploration, motor, move, positioning, robot, trajectory, velocity
9	class, classifier, clustering, data, dimensionality, features, label, table, testing, training, validation
13	dot, edges, field, horizontal, images, matching, object, orientation, perception, pixel, plane, projection, retina, rotation, scene, shape, spatial, vertical, vision, visual
14	bayesian, conditional, covariance, density, distribution, estimate, expectation, gaussian, inference, likelihood, mixture, model, parameter, posterior, prior, probability
17	analog, bit, chip, circuit, design, diagram, digital, gate, hardware, implement, integrated, output, Power, processor, pulse, source, transistor, vlsi, voltage
19	**acknowledgement**, **department**, **foundation**, **grant**, **institute**, **research**, **support**, **thank**, **university**

divergence between the attribute distribution induced when C_i is present and the attribute distribution induced when C_i is absent.

It is impractical for us to report $kl(j)$ for each component-word pair, but these values can be found in the technical report [22]. We report the most important words for some selected components in Table 5.

Discussion. There is a very clear and intuitive meaning associated with each of the learned components, and the components give a very nice tour through the various topics or sections that can be found in a NIPS paper.

For example, component 1 concerns itself with words related to theory and proofs. Component 3 is concerned with speech processing. Words related to the brain and nervous system can be seen in component 5. Component 6 is concerned with neural networks. Component 7 is concerned with robotics and moving objects. Component 9 is associated with classification and data mining. Component 13 is responsible for words related to computer vision and image processing. Words related to statistical methods (particularly Bayesian methods) are captured in component 14. Component 17 is concerned with electronics and electrical systems.

Not all components chose to induce the presence of words in a paper. Component 19 is particularly interesting because the presence of this component tends to suppress acknowledgements and words related to academia and grants.

5.3 Reuters Data Set

In order to test our model with a larger data set, we collected news stories from the Reuters website.

Experimental setup. For each day in 2007, we sampled 10% of the stories from the Reuters archive [19]. Over the entire corpus of the documents we removed stop words and trivial words. We pushed all of the remaining words through Snowball English stemmer [17] to obtain word roots. We picked the most frequent 1000 resulting words, and each story was converted to a row of zeros and ones corresponding to the absence and presence of these words. We dropped all the stories that did not contain at least 100 of the final 1000 words. Based on all of this processing, we finally obtained a data set that consisted of 22,429 stories, and each story was represented by a 0/1 vector of length 1000, indicating absence/presence of the words.

Table 6: The most highly-ranked words for some of the components learned from the Reuters dataset. Plain text indicates that the component tries to enhance the likelihood that the word appears, while bold text indicates that the component tries to suppress the word.

id	Words
2	asia, beij, china, chinese, global, hong, kong, **mexico**
3	armi, attack, bomb, border, camp, civilian, dead, fight forc, govern, kill, milit, militari, polic, provinc, rebel region, soldier, troop, violence, war, worker, wound
4	album, film, game, hollywood, love, make, movi, music, night, play, realli, song, sport, star, stori, tour
5	case, charge, claim, committe, congress, court, feder, file, hear, investig, judg, law, legal, prison, request, senat
7	britain, brussel, euro, europ, european, franc, french, german, germani, **iraq**, london, uk, **washington**
12	acquisit, bid, buy, chairman, chief, compani, deal, equiti, execut, firm, hold, invest, merger, offer, partner, privat, share, sharehold, stake, stock, takeov, valu, ventur
13	bank, benchmark, bond, currency, dollar, economi, gain, index, inflat, investor, market, price, ralli, session, trade
15	cancer, care, clinic, death, diseas, drug, effect, food, health, heart, hospit, human, journal, medic, patient, pharamaceut, prevent, safeti, studi, treatment, trial
16	**asia**, **beij**, **china**, **chines**, **india**, **japan**, **japanes**, **tokyo**, **world**
17	alli, bush, contri, diplomat, elect, foreign, govern, hama, iran, iraq, isra, leader, minist, palestinian, peace, war
19	advertis, broadcast, communic, content, devic, digit, electron, internet, launch, mobil, network, onlin, phone, softwar, technolg, televis, user, video, web, wireless
20	carbon, climat, coal, demand, effici, electr, emiss, energi, environment, fuel, gas, generat, miner, natur, oil, output plant, Power, produc, suppli, warm, water, world

The experimental settings were exactly the same as in the NIPS data set. We ran our algorithm for 2000 iterations, including 1000 iterations for the burn-in. We show some of the components with some of their important word roots in Table 6—"important" is again computed using KL-divergence, as in the case of the NIPS data set.

The total running time for the learning was 510 hours, at an average of 15 minutes per iteration.

Discussion. Despite large number of stories and high dimensionality, the learning algorithm again gives a good idea of the standard topics that were represented in the resulting model. Components 2 and 16 are very interesting. Component 2 tends to create a news story that is concerned with China (and, interestingly, *not* with Mexico), while component 16 suppresses words that have anything to do with Asia. The presence of component 3 causes the resulting article to be concerned with war and fighting. Words related to sports and entertainment industry are dominated by component 4. Component 5 causes the presence of words related to court cases. Component 7 tends to influence the presence of words related to Europe. At the same time, it tends to suppress the words Iraq and Washington—perhaps this is as a result of the chilly relationship

between Europe and the United States in 2007, as well as the lack of European involvement in the Iraq war. Component 12 is associated with articles that give company profiles, and component 13 is associated with articles covering the financial markets. Component 15 is associated with health-care articles, and component 17 with the middle east. Technology sector words are influenced by component 19. Component 20 is associated with words related to global warming and energy sector.

5.4 S&P 500 Data Set

As a final experiment, we applied the Power model to 17 years of data tracking the 500 stocks listed on the Standard and Poor's 500.

Experimental setup. The data we used included trading from January 8,1995 through September 8, 2002. For each stock on each day, we record whether the stock fluctuated up or down during the day's trading, resulting in a one or a zero, respectively. In this way, there is a 500-dimensional vector that summarizes each day's trading. We again learned a 21-component Power model for the data. We ran our algorithm for 2000 iterations, including 1000 iterations for the burn-in. We show some of the components with some of the key stocks in Table 7—the "key stocks" are again computed using KL-divergence.

The total running time for the learning was 33 hours, at an average of 48 seconds per iteration.

Discussion. All of the stocks at the top of the learned components have an obvious interpretation. The very strongest learned pattern was represented by the three mining companies from component 13, which tend to move in lock-step with one another. Component 3 consists entirely of financial stocks. Component 4 is drug and biotech companies. Component 5 contains oil and oil services companies. Component 10 consists of communication companies (interestingly, all of the companies except for Knight-Ridder are content delivery companies, whereas Knight-Ridder is a content provider). Component 15 consists of tech companies—the list of tech companies at the top of this component was actually much longer than what we reproduce here. Component 16 is entirely electricity companies. Component 17 is purely oil services companies.

Components 1, 2, and 19 are interesting in that they contain anti-correlations. Component 1 consists of food companies plus Bear Stearns, whose stock tends to go down when the food companies' stocks go up. Likewise, component 2 consists of large retailers, plus five commodities organizations that are anti-correlated with the retailers. Finally, component 19 consists of nine companies that provide communication and networking hardware, plus 14 other companies that are anti-correlated. These anti-correlated companies are mostly consumer-related, but also include Fannie Mae and Freddie Mac (the two mortgage giants), plus United Health.

Table 7: The most highly-ranked stocks in selected components. Plain text indicates that the component tries to enhance the likelihood that the stock increases in value, while bold text indicates that the component tries to decrease the stock value.

id	Stocks
1	Coke, **Bear Stearns**, Conagra Foods, Gen. Mills, Heinz, Hershey, Kellogg, Proctor & Gamble, Sysco, Winn-Dixie, Wrigley
2	**Boise Cascade**, Dollar General, **Dow Chem.**, Household Intl., Kohls, May Dept., **Mead West**, Target, **Temple-Inland**, **US Steel**, Wal-Mart
3	Bank New York, Bank America, BB&T, Comerica, Fifth Third, Nat'l City, PNC Financial, Suntrust, Synovus, Union Planters, US Bancorp
4	Abbott Labs, Amgen, Baxter, Bristol-Myer, Eli Lilly, J&J, Merck, Pfizer, P&G, Schering-Plough, Wyeth
5	Amerada Hess, Anadarko Petrol, Apache, Chevron-Tex., Kerr-McGee, Marathon Oil, Phillips Pete, RDS, Unocal
10	AllTel, AT&T, BellSouth, Knight-Ridder, SBC, Sprint, Verizon, Viacom
13	Barrick Gold, Placer Dome, Newmont Mining
15	AMD, Altera, Analog Devices, Apple, Applied Material, BMC, Cisco, Dell, EMC, Intel, KLA-Tenco, Linear Tech, LSI, Maxim, Micron, Microsoft, Motorola, Nat'l Semi., Novell, Oracle, PMC-Sierra, Sun, Tellabs, Teradyne, TI, Xilinx
16	Allegheny, Am. Elec., Cinergy, Con. Ed., Constellation, Dominion, DTE Energy, Duke Energy, Edison, Entergy, Exelon, FPL Group, Nisource, PG&E, Pinnacle West, PPL, Progress, Pub Serve, Reliant, Southern Co., Teco Engergy, TXU
17	Baker Hughes, BJ Servies, Halliburton, Kerr-McGee, Nabors, Noble Corp., Rowan, Schlumberger, Transocean
19	Applied Micro, Boradcom, Ciena, Clear Channel, **Coke**, **Colgate Palmolive**, **Conagra**, **Fannie Mae**, **Freddie Mac**, **Gen. Mills**, **Gillette**, **Heinz**, **Pepsico**, **Phillip Morris**, Power-One, Qwest, Rational, **Sara Lee**, **Unilever**, **United Health**, **Wrigley**

6. RELATED WORK

The closest related work is likely our own prior work defining the MOS or "Mixture of Subsets" model [21]. Like the Power model, in the MOS model various components compete to influence each data attribute. However, the MOS model suffers from a complicated and rather cumbersome mechanism for allowing the various components to choose which attributes they will influence. In our opinion, this limits the practical applicability of the MOS methods. Another key difference is that the MOS methodology makes use of an intricate Monte Carlo expectation-maximization algorithm [7] to perform the learning, rather than the Bayesian approach advocated in this paper.

Our work can be viewed as having some of the same goals as the now-classic papers on subspace clustering [2, 1]. However, a key difference is that we have taken a statistical approach to the problem, where we attempt to learn a generative model. This has many implications, such as the fact that our learned model has a clear, statistical interpretation; and that our framework is data-type agnostic—our framework is applicable to categorical data, numerical data restricted to positive integers, and so on, as long as one substitutes a distribution that is appropriate for the data in question.

Another popular approach for multi-class membership is the Latent Dirichlet Allocation (LDA) topic model [4]. Under the LDA model, each "document" in a text corpus is generated based on a mixture of "topics" drawn from a common Dirichlet distribution. Each "word" in the document is then generated by the topic selected via a multinomial draw from this mixture of topics. It is easy to observe that there is some similarity between our approach and the LDA model. However, the most significant difference is how a topic expresses itself in a given document in our model. Our model is specifically designed so that a role (or "topic" in LDA parlance) may choose to influence only a small subset of the data attributes. In the LDA model, the probability of a particular topic controlling the generation of a word is the same for all words in the document. There is no way for a topic to indicate a set of words that are particularly important to it. Going back to our motivating example, this would mean that under LDA, a customer's decision of whether or not to rent the movies *Cinderella*, *Iron Man*, and *Mamma Mia* will all have the same probability of being generated based on the preferences of the class *action-movies-fan*, when only *Iron Man* is an action movie.

The inability of LDA to allow topics to show a preference for certain words should not be confused with a topic's ability under LDA to supply strong/weak probability of generating a particular word. An LDA *action-movies-fan* topic can indeed choose to assign a low probability to a children's movie such as *Cinderella*, but the result is that the presence of the *action-movies-fan* topic will actively suppress the selection of *Cinderella*. Practically speaking, this means that the LDA model will tend to discount the possibility of a "document" corresponding to a renter who is an *action-movies-fan* but also a *small-child*.

In contrast, in our proposed model a mixture component can choose to favor a particular word or a set of words by significantly increasing the corresponding parameter weights, while at the same time ignoring a particular word or a set of words by decreasing the corresponding parameter weights. This allows each topic to cede words that are not important to it to other topics in the model.

Mixture models [14] have a long history in the KDD literature [5, 6]. However, there have been only a few other mixture models in the KDD and machine learning literature that allow a generative process where multiple components influence the generation of a single data point. For example, Griffiths and Ghahramani [8] have derived a prior distribution for Bayesian generative models called the *Indian Buffet Process* that allows each data point to belong to potentially any combination of the infinitely many classes. However, they initially did not define how these classes combine to generate the actual data. In follow-up work, Heller and Ghahramani [9] and later Heller, Williamson, and Ghahramani [10] propose taking the product of the various distributions that influence a data point in order to handle the case of multiple generating classes. Unlike our own framework, there is no way in which classes express a preference or affinity for certain attributes as opposed to others.

7. CONCLUSIONS

In this paper, we have introduced a new class of mixture models called Power models, and defined a generic probabilistic framework to enable learning of Power models. The key novelty of Power modeling framework is that it allows multiple components in the mixture model to influence a data point simultaneously, and also provides a framework for each component to choose varying degree of influence on the data attributes. Our modeling framework is data-type agnostic, and can be used for any data that can be modeled using a parameterized probability density function. We have derived a learning algorithm that is suitable for learning this class of probabilistic models, and proposed an qualitatively accurate approximation that speeds up parts of our learning algorithm. We have also shown the usefulness of our modeling framework using experiments on several real-life high-dimensional datasets.

8. ACKNOWLEDGEMENTS

This work was supported by the National Science Foundation under grant number 0612170, as well as an Alfred P. Sloan Foundation Fellowship.

9. REFERENCES

[1] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park. Fast algorithms for projected clustering. In *SIGMOD*, pages 61–72, New York, NY, USA, 1999. ACM Press.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD*, pages 94–105, New York, NY, USA, 1998. ACM Press.

[3] A. Asuncion and D. Newman. UCI machine learning repository. `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 2007.

[4] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003.

[5] I. Cadez, S. Gaffney, and P. Smyth. A general probabilistic framework for clustering individuals and objects. In *KDD*, pages 140–149, 2000.

[6] I. Cadez, P. Smyth, and H. Mannila. Probabilistic modeling of transaction data with applications to profiling, visualization, and prediction. In *KDD*, pages 37–46, 2001.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of Royal Statistical Society*, B-39:1–39, 1977.

[8] T. Griffiths and Z. Ghahramani. Infinite latent feature models and the indian buffet process. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *NIPS 18*, pages 475–482. MIT Press, Cambridge, MA, 2006.

[9] K. Heller and Z. Ghahramani. A nonparametric bayesian approach to modeling overlapping clusters. In *AISATS*. The Society for Artificial Intelligence and Statistics, 2007.

[10] K. A. Heller, S. Williamson, and Z. Ghahramani. Statistical models for partial membership. In *ICML*, pages 392–399, 2008.

[11] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[12] J. S. Liu. The collapsed gibbs sampler in bayesian computations with applications to a gene regulation problem. *Journal of the American Statistical Association*, 89(427):958–966, 1994.

[13] J. S. Liu, W. H. Wong, and A. Kong. Covariance structure of the Gibbs sampler with applications to the comparisons of estimators and augmentation schemes. *Biometrika*, 81(1):27–40, 1994.

[14] G. J. McLachlan and K. E. Basford. *Mixture Models: Inference and Applications to Clustering*. Marcel Dekker, New York, 1988.

[15] G. J. McLachlan and D. Peel. *Finite Mixture Models*. Wiley, New York, 2000.

[16] J. V. Neumann. Various techniques used in connection with random digits. *Applied Math Series*, 1951.

[17] M. F. Porter. Snowball: A language for stemming algorithms. `http://www.snowball.tartarus.org/texts/introduction.html`, 2001.

[18] A. E. Raftery and S. Lewis. How many iterations in the gibbs sampler? In *Bayesian Statistics*, volume 4, pages 763–773. Oxford University Press, 1992.

[19] Reuters news and video archive. `http://www.reuters.com/resources/archive/us/index.html`.

[20] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer, 2005.

[21] M. Somaiya, C. Jermaine, and S. Ranka. Learning correlations using the mixture-of-subsets model. *ACM Trans. Knowl. Discov. Data*, 1(4):1–42, 2008.

[22] M. Somaiya, C. Jermaine, and S. Ranka. Various experiments using POWER models. `http://www.cise.ufl.edu/~ranka/power/`, 2010.