ACCAMS: Additive Co-Clustering to Approximate Matrices Succinctly

Alex Beutel
Computer Science, CMU
Pittsburgh, PA
abeutel@cs.cmu.edu

Amr Ahmed
Google ST
Mountain View, CA
amra@google.com

Alexander J. Smola
CMU MLD and Google ST
Pittsburgh, PA
alex@smola.org

ABSTRACT

Matrix completion and approximation are popular tools to capture a user's preferences for recommendation and to approximate missing data. Instead of using low-rank factorization we take a drastically different approach, based on the simple insight that an additive model of co-clusterings allows one to approximate matrices efficiently. This allows us to build a concise model that, per bit of model learned, significantly beats all factorization approaches in matrix completion. Even more surprisingly, we find that summing over small co-clusterings is more effective in modeling matrices than classic co-clustering, which uses just one large partitioning of the matrix.

Following Occam's razor principle, the fact that our model is more concise and yet just as accurate as more complex models suggests that it better captures the latent preferences and decision making processes present in the real world. We provide an iterative minimization algorithm, a collapsed Gibbs sampler, theoretical guarantees for matrix approximation, and excellent empirical evidence for the efficacy of our approach. We achieve state-of-the-art results for matrix completion on Netflix at a fraction of the model complexity.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Data mining; H.3.m [**Information Storage and Retrieval**]: Miscellaneous; I.5.3 [**Computing Methodologies**]: Clustering

Keywords

Recommender Systems; Collaborative Filtering; Clustering

1. INTRODUCTION

Given users' ratings of movies or products, how can we model a user's preferences for different types of items and recommend other items that the user will like? This problem, often referred to as the Netflix problem, has generated a flurry of research in collaborative filtering, with a variety of proposed matrix factorization models and inference methods. Top recommendation systems have used thousands of factors per item and per user, as was the case in the winning submissions in the Netflix prize [10]. Recent state-of-the-art methods have relied on learning even larger, more complex factorization models, often taking nontrivial combinations of multiple submodels [16, 13]. Such complex models are increasingly difficult to interpret, use large amounts of memory, and are often difficult integrate into larger systems.

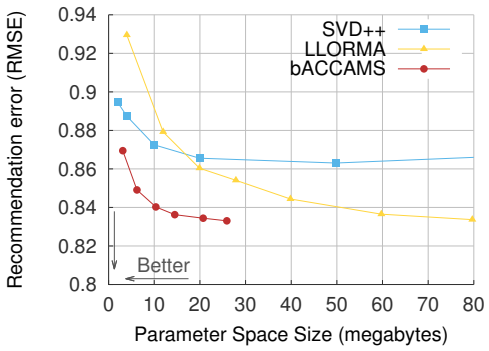


Figure 1: Accuracy of ACCAMS on Netflix, compared to [10] and [13]. Note that our model achieves state of the art accuracy at a fraction of the model size.

1.1 Linear combinations of attributes

Our approach is drastically different from previous collaborative filtering research. Rather than start with the assumptions of a matrix factorization model, we make *co-clustering* effective for high quality matrix completion and approximation. Co-clustering finds a clustering of the rows and columns of a matrix $\mathbf{R}$ so as to partition $\mathbf{R}$ into blocks that are highly similar. It has been well studied [2, 21] but was not previously competitive in large behavior modeling and matrix completion problems. To achieve state of the art results, we use an *additive model of simple co-clusterings* that we call stencils, rather than building a large single co-clustering. The result is a model that is conceptually simple, has a small parameter space, has interpretable structure, and still achieves the best published accuracy for matrix completion on Netflix, as seen in Figure 1.

Using a linear combination of co-clusterings corresponds to a rather different interpretation of user preferences and movie properties. Matrix factorization assumes that a movie preference is based on a weighted sum of preferences for

WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741091.

different genres, with the movie properties being represented in vectorial form. Therefore, even if a movie is a comedy and the user likes comedies, the model still also offsets by how scary the movie is and does the user like scary movies.

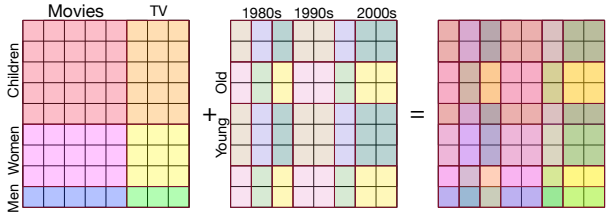
Co-clustering on the other hand assumes there exists some "correct" partitioning of movies (and users). For instance, a user might be part of a group that likes all comedies but does not like romantic movies. Correspondingly, all romantic comedies might be split out into a separate cluster. If a group of users likes new movies but not old ones then each genre may be further partitioned by decade. This quickly leads to a combinatorial explosion.

By taking a linear combination of co-clusterings we benefit from both perspectives: by modeling the discrete nature of attributes we can avoid the cost of high-dimensional factorization models, and by adding the preferences for different attributes we can avoid the large models necessary to cover all combinations of attributes. Rather, through backfitting, we create a more powerful, hierarchical representation. For instance, a movie may be {funny, scary, sad}, it was made in some era, it has a certain age rating, it may contain a certain group of actors or be shot in a certain visual style. However, if a user likes comedies but doesn't like scary movies, it is generally unlikely that this preference will suddenly flip depending on the decade the movie was produced. Therefore, by taking linear combinations of co-clusterings we can efficiently take all of these attributes into account.

1.2 Succinct Stencils

The mathematical challenge that motivated this work is, how can we make a *succinct* model of user behavior that still provides high quality predictions? Designing a succinct model is difficult because it requires making assumptions and restrictions while not decreasing the model's accuracy. Factorization models are very flexible. In order to encode a rank-k matrix by a factorization, we need k numbers per row (and column) respectively. Rather, consider a stencil - a small $k \times k$ template of a matrix and its mapping to the row and column vectors respectively. We only need $\log_2 k$ bits per row (and column) plus $O(k^2)$ floating point numbers regardless of the size of the matrix.

Taking a linear combination of stencils we can model quite complex matrices. This is best understood by the example below: assume that we have two simple stencils containing 3×2 and 2×3 co-clusterings. Their linear combination yields a rather nontrivial 9×8 matrix of rank 5. Alternatively, classic co-clustering would require a $(3\cdot2)\times(2\cdot3)$ partitioning to match this structure. When we have S stencils of size $k\times k$, this would require a single co-clustering of size $k^S \times k^S$.



By design our model has a parameter space that is an order of magnitude smaller than competing methods, requiring only $S \log_2 k$ bits per user and per movie and Sk^2 floating point numbers, where k is generally quite small. While ACCAMS is more restrictive than classic factorization, we demonstrate that our assumptions do not increase the generalization error, achieving even better prediction accuracy than more complex models.

Finding succinct models for binary matrices, e.g. by minimizing the minimum description (MDL), has been the focus of significant research and valuable results in the data mining community [14, 11]. That said, these models are quite different. To the best of our knowledge, ours is the first work aimed at finding a parsimonious model for general (real-valued) matrix completion and approximation.

1.3 Contributions

Our paper makes a number of contributions to the problem of finding sparse representations of matrices.

- **Optimization algorithm:** We present ACCAMS, an iterative k-means style algorithm that minimizes the approximation error by backfitting the residuals of previous approximations.
- **Theoretical bounds:** We provide linear approximation rates exploiting the geometry of rows and columns of rating matrix.
- **Bayesian model:** We present a generative Bayesian non-parametric model and devise a collapsed Gibbs sampling algorithm, bACCAMS, for efficient inference.
- **State-of-the-art results:** Experiments confirm the efficacy of our approach, offering the best published results for matrix completion on Netflix, an interpretable hierarchy of content, and succinct matrix approximations for ratings, image, and network data.

We believe that these contributions offer a promising new direction for behavior modeling and matrix approximation.

Outline. We begin by discussing related work from recommendation systems, non-parametric Bayesian models, co-clustering, and minimum description length. We subsequently introduce the simple k-means style co-clustering and its approximation properties in Section 3. Subsequently, in Section 4.1 we define our Bayesian co-clustering model and collapsed Gibbs sampler for a single stencil. In Section 4.4 we extend our Bayesian model to multiple stencils. Section 5 reports our experimental results and we conclude with a discussion of future directions for the work.

2. RELATED WORK

Recommender Systems. Closest to our work is the research on behavior modeling and recommendation. Matrix factorization approaches, such as Koren's SVD++ [9], have enjoyed great success in recommender systems. Recent models such as DFC [16] and LLORMA [13] have focused on using ensembles of factorizations to exploit local structure.

More closely related to our model are Bayesian non-parametric approaches. For instance, [6, 5] use the Indian Buffet Process (IBP) for recommendation. In doing so they assume that each user (and movie) has certain preferential binary attributes. It can be seen as an extreme case of ACCAMS where the cluster size $k = 2$, while using a somewhat different strategy to handle cluster assignment and overall similarity within a cluster. Following a similar intuition as ACCAMS but different perspective and focus, [17] extended the IBP to handle $k > 2$ for link prediction tasks on binary graphs. Our work differs in its focus on general, real-valued matrices, its application of co-clustering, and its significantly simpler parameterization.

Co-clustering was incorporated into a factorization approach to recommendation in [3]. While the co-clustering improved modeling accuracy, it did not reduce the model complexity of the underlying factorization. Finally, [19] proposed a factorization model based on a Dirichlet process over users and columns. All these models are closely related to the mixed-membership stochastic blockmodels of [1].

Co-clustering. The technique was originally used for understanding the clustering of rows and columns of a matrix rather than for matrix approximation or completion [8]. This formulation was well suited for biological tasks but evolved to cover a wider variety of objectives [2]. [18] defined a soft co-clustering objective akin to a factorization model. Recent work has defined a Bayesian model for co-clustering focused on matrix modeling [21]. [24] focuses on exploiting co-clustering ensembles, but do so by finding a single consensus co-clustering. As far as we know, ours is the first work to use an additive combination of co-clusterings.

Matrix Approximation. There exists a large body of work on matrix approximation in the theoretical computer science community. They focus mainly on efficient low-rank approximations, e.g. by projection or by interpolation [7, 4]. Essentially one aims to find a general low-rank approximation of the matrix, similar to recommender models.

A more parsimonious strategy is to seek *interpolative* decompositions, approximating columns of a matrix by a linear combination of a subset of other columns [15]. Nonetheless this requires us to store at least one, possibly more scaling coefficients per column. Also note the focus on column interpolations — this can easily be extended to row and column interpolations. To the best of our knowledge, the problem of approximating matrices with piecewise constant block matrices as we propose here is not the focus of research in TCS.

Succinct modeling. The data mining community has focused on finding succinct models of data, often directly optimizing the model size described by the minimum description length (MDL) [20]. This approach has led to valuable results in pattern and item-set mining [23, 14] as well as graph summarization [11]. However, these approaches typically focus on modeling databases of discrete items rather than real-valued datasets with missing values.

3. MATRIX APPROXIMATION

We begin by defining our notation and the problem. We will denote matrices by bold capital letters, vectors by bold lowercase letters, and scalars by non-bold symbols. Unless otherwise specified, subscripts on matrices (or vectors) denote indices into the matrix (or vector), e.g. $\mathbf{R}_{u,m}$ is the scalar in the uth row and mth column of $\mathbf{R}$. Using : selects all indices, such that $\mathbf{R}_{u,:}$ represents the entire uth row of $\mathbf{R}$ and $\mathbf{R}_{:,m}$ is the entire mth column. Superscripts, as in $\mathbf{T}^{(\ell)}$, denote different matrices for different superscripts. A full list of the symbols used can be found in Table 1.

We consider now the problem of matrix approximation:

Problem Definition 1 ***(Matrix Approximation)*** ***Given:*** *a sparse matrix $\mathbf{R} \in \mathbb{R}^{N \times M}$ with indicator matrix $\mathbf{I}$* ***Find:*** *a model $\mathcal{M}$ with parameters θ, such that the size of θ is small, $|\theta| \ll \mathbf{R}$, and $\mathcal{M}(\theta)$ approximates $\mathbf{R}$ well:*

$$\underset{\theta}{\text{minimize}} \sum_{u=1}^{N} \sum_{m=1}^{M} \mathbf{I}_{u,m} (\mathbf{R}_{u,m} - \mathcal{M}(\theta)_{u,m})^2 \qquad (1)$$

Symbol	Definition
N, M	Number of rows (users) and columns (movies)
$\mathbf{R}$	Data matrix $\in \mathbb{R}^{N \times M}$ (with missing values)
$\mathbf{I}$	Indicator matrix $\in \{0,1\}^{N \times M}$ for $\mathbf{R}$
S	Number of stencils
$k_n^{(\ell)}, k_m^{(\ell)}$	Number of user and movie clusters in stencil ℓ
$\mathbf{T}^{(\ell)}$	Matrix $\in \mathbb{R}^{k_n^{(\ell)} \times k_m^{(\ell)}}$ for stencil ℓ
$\mathbf{c}^{(\ell)}$	Vector of user assignments $\in \{1, \ldots, k_n^{(\ell)}\}^N$
$\mathbf{d}^{(\ell)}$	Vector of movie assignments $\in \{1, \ldots, k_m^{(\ell)}\}^M$
$\mathcal{S}(\mathbf{T}, \mathbf{c}, \mathbf{d})$	$\in \mathbb{R}^{N \times M}$ defined by $\mathcal{S}(\mathbf{T}, \mathbf{c}, \mathbf{d})_{u,m} = \mathbf{T}_{\mathbf{c}_u, \mathbf{d}_m}$
$\mathbf{n}_c$	Number of users in cluster c
$\mathbf{n}_c^{(-u)}$	Number of users in cluster c, ignoring user u
$\mathbf{N}_{c,d}$	Number of observations in block (c, d)
$\mathbf{r}_{c,d}$	Vector of observed ratings in block (c, d)

Table 1: Symbols used throughout this paper.

As stated above, we assume $\mathbf{R}$ is a sparse matrix where there is no data for many values in the matrix. The indicator matrix $\mathbf{I}$ denotes this information, where $\mathbf{I}_{i,j} = 1$ when there is an observed value for $\mathbf{R}_{i,j}$ and $\mathbf{I}_{i,j} = 0$ otherwise. Without loss of generality we assume that our data matrix, $\mathbf{R}$, has more rows than columns, i.e. $\mathbf{R} \in \mathbb{R}^{N \times M}$ with $N \geq M$.

The concept of a small model in the context of behavior modeling has typically been captured by the rank of a factorization. We generalize this concept and define a small model by the number of bits required to store it, more commonly known as the minimum description length (MDL).

3.1 Proposed Model

A stencil assigns each user u to a user cluster c, each movie to a movie cluster d, and for each (user cluster c, movie cluster d) combination there is a *block* of ratings which are predicted to have value $\mathbf{T}_{c,d}$. Formally:

Definition 1 (Stencil) *A stencil $\mathcal{S}(\mathbf{T}, \mathbf{c}, \mathbf{d})$ is a matrix $\mathcal{S} \in \mathbb{R}^{N \times M}$ with the property that $\mathcal{S}(\mathbf{T}, \mathbf{c}, \mathbf{d})_{u,m} = \mathbf{T}_{\mathbf{c}_u, \mathbf{d}_m}$ for a template $\mathbf{T} \in \mathbb{R}^{k_n \times k_m}$ and discrete index vectors $\mathbf{c} \in \{1, \ldots, k_n\}^N$ and $\mathbf{d} \in \{1, \ldots, k_m\}^M$ respectively.*

Therefore, our goal is to find a stencil $\mathcal{S}(\mathbf{T}, \mathbf{c}, \mathbf{d})$ with a small approximation error $\mathbf{R} - \mathcal{S}(\mathbf{T}, \mathbf{c}, \mathbf{d})$ and small cost for storing $\theta = \{\mathbf{T}, \mathbf{c}, \mathbf{d}\}$.

Lemma 2 (Compression) *Stencil $\mathcal{S}(\mathbf{T}, \mathbf{c}, \mathbf{d})$ can be stored with $N \log_2 k_n$ bits for row cluster assignments $\mathbf{c}$, $M \log_2 k_m$ bits for column cluster assignments $\mathbf{d}$ and $32 k_n k_m$ bits to store a 32-bit floating point number for each block in $\mathbf{T}$:*

$$\text{Bits}(\{\mathbf{T}, \mathbf{c}, \mathbf{d}\}) = N \log_2 k_n + M \log_2 k_m + 32 k_n k_m \qquad (2)$$

Note, it is trivial to get zero approximation error by setting $k_n = N$ and $k_m = M$, but this creates a very large model (the size of the original data) that is not useful.

As mentioned above, we can efficiently improve the approximation accuracy by using *multiple* stencils:

$$\underset{\{\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)}\}}{\text{minimize}} \sum_{u=1}^{N} \sum_{m=1}^{M} \mathbf{I}_{u,m} \left(\mathbf{R}_{u,m} - \sum_{\ell=1}^{S} \mathcal{S}\left(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)}\right)_{u,m} \right)^2$$

That is, we would like to find an additive model of S stencils that minimizes the approximation error to $\mathbf{R}$.

Algorithm 1 Matrix Approximation

Require: matrix $\mathbf{R}$, indicator matrix $\mathbf{I}$, clusters k_n, k_m, max stencils S

1: $\hat{\mathbf{R}} \leftarrow \mathbf{R}$
2: **for** $\ell = 1$ **to** S **do**
3: $(\mathbf{c}^{(\ell)}, \mathbf{V}^{(\text{row})}, \mathbf{L}) \leftarrow \text{CLUSTER}(\hat{\mathbf{R}}, k_n, \mathbf{I})$ {Rows}
4: $(\mathbf{d}^{(\ell)}, \cdot, \cdot) \leftarrow \text{CLUSTER}([\mathbf{V}^{(\text{row})}]^\top, k_m, \mathbf{L}^\top)$ {Columns}
5: **for all** $a, b \in \{1, \dots k_n\} \times \{1, \dots k_m\}$ **do**
6: $\mathbf{T}^{(\ell)}_{a,b} \leftarrow \text{mean}\left\{\hat{\mathbf{R}}_{u,m} | \mathbf{c}^{(\ell)}_u = a \text{ and } \mathbf{d}^{(\ell)}_m = b\right\}$
7: **end for**
8: $\hat{\mathbf{R}} \leftarrow \hat{\mathbf{R}} - \mathcal{S}(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})$ {Backfit on residuals}
9: **end for**
10: **return** $\{\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)}\}_{\ell=1}^{S}$

Algorithm 2 CLUSTER$(\mathbf{M}, k, \mathbf{W})$

Require: matrix $\mathbf{M} \in \mathbb{R}^{N_1 \times N_2}$, weights $\mathbf{W} \in \mathbb{R}^{N_1 \times N_2}$, number of clusters k

1: Draw k rows from $\mathbf{M}$ at random without replacement and copy them to $\mathbf{V} = \{\mathbf{v}_1, \dots, \mathbf{v}_k\} \in \mathbb{R}^{k \times N_2}$.
2: **while** not converged **do**
3: $\mathbf{L} \leftarrow \mathbf{0} \in \mathbb{R}^{k \times N_2}$ and $\mathbf{Y} \leftarrow \mathbf{0} \in \mathbb{R}^{k \times N_2}$
4: **for** $i = 1$ **to** N_1 **do**
5: $\mathbf{c}_i \leftarrow \text{argmin}_c \sum_j \mathbf{W}_{i,j}(\mathbf{M}_{i,j} - \mathbf{V}_{c,j})^2$
6: $\mathbf{Y}_{\mathbf{c}_i,:} \leftarrow \mathbf{Y}_{\mathbf{c}_i,:} + \mathbf{M}_{i,:}$ {Increment statistics}
7: $\mathbf{L}_{\mathbf{c}_i,:} \leftarrow \mathbf{L}_{\mathbf{c}_i,:} + \mathbf{I}_{i,:}$ {Increment counts}
8: **end for**
9: **for** $c = 1$ **to** k **do**
10: $\mathbf{V}_{c,:} \leftarrow \mathbf{Y}_{c,:}/\mathbf{L}_{c,:}$ {New cluster center}
11: **end for**
12: **end while**
13: **return** cluster assignments $\mathbf{c}$, clusters $\mathbf{V}$, counts $\mathbf{L}$

Given $\mathbf{R}$, it is our goal to *find* such stencils $\mathcal{S}(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})$ with good approximation properties. Unfortunately, finding linear combinations of co-clusterings is NP-hard. It is easy to see this by reducing co-clustering, which is NP-hard, to our problem by setting $S = 1$. We describe below two algorithms to learn our stencils that offer good approximation guarantees and work well in practice.

3.2 Algorithm

We consider a simple iterative procedure to learn stencils to approximate our data $\mathbf{R}$. Algorithm 1 gives the high level algorithm of learning each stencil one at a time. In learning each stencil we use the CLUSTER algorithm, similar to k-means, as given in Algorithm 2.

Row clustering. We first perform k-means clustering of the rows. That is, we aim to find an approximation of $\mathbf{R}$ that replaces all rows by a small subset thereof. Algorithm 2 is essentially a generalization of k-means clustering. By calling the algorithm with $\mathbf{M} = \mathbf{R}$, $k = k_n$, and $\mathbf{W} = \mathbf{1}^{N \times M}$, we find that the algorithm simplifies significantly to classic k-means. The cluster assignment in Line 5 is simply

$$\mathbf{c}_u \leftarrow \underset{c}{\text{argmin}} \|\mathbf{R}_{u,:} - \mathbf{v}_c\|_2^2 \tag{3}$$

and $\mathbf{L}$ stores the number of rows in each row cluster.

Column clustering. Once we have run the clustering algorithm on the rows, we now cluster the columns of previously learned row clusters, $\mathbf{V}^{(\text{row})}$. In this case, we need to weight each row of $\mathbf{V}^{(\text{row})}$ (corresponding to a row cluster) by the number of rows it represents (the number of rows in that cluster). As a result, rather than choose a column cluster assignment by the Euclidean distance, we use the Mahalanobis distance. Still, Algorithm 2 is a generalization of this concept. We use the assignment (for Line 5):

$$\mathbf{d}_m \leftarrow \underset{d}{\text{argmin}} \left(\mathbf{V}^{(\text{row})}_{:,m} - \mathbf{V}_{:,d}\right)^\top \mathbf{D} \left(\mathbf{V}^{(\text{row})}_{:,m} - \mathbf{V}_{:,d}\right) \tag{4}$$

where $\mathbf{V} \in \mathbb{R}^{k_n \times k_m}$ is the matrix obtained by stacking $\mathbf{V}_{:,d} = \mathbf{v}_d^\top$ and $\mathbf{D}$ would be the diagonal matrix of counts, i.e. $\mathbf{D}_{cc}$ is the number of rows of $\mathbf{R}$ in cluster c.

Missing entries. In many cases, however, $\mathbf{R}$ itself is incomplete. This is addressed quite easily by using the assignment shown in Line 5 of Algorithm 2. Therefore, in finding a good cluster for the row $\mathbf{R}_{u,:}$ we restrict ourselves to the coordinates in $\mathbf{V}_{c,:}$ where $\mathbf{R}_{u,m}$ exists (and also where $\mathbf{V}_{c,m}$ has been initialized).

For the purpose of obtaining the column clusters, we now need to weight each coordinate in $\mathbf{V}^{(\text{row})}$ by how many elements in $\mathbf{R}$ contributed to it. Correspondingly denote by $\mathbf{L}_{c,m} := \sum_{(u,m) \in \mathbf{R}: \mathbf{c}_u = c} 1$ the number of entries mapped into coordinate $\mathbf{V}^{(\text{row})}_{c,m}$. Then the assignment for column clusters is obtained via

$$\mathbf{d}_m \leftarrow \underset{d}{\text{argmin}} \sum_c \mathbf{L}_{c,m} \left(\mathbf{V}^{(\text{row})}_{c,m} - \mathbf{V}_{c,d}\right)^2 \tag{5}$$

We observe that Algorithm 2 handles both row and column clustering correctly, by calling it appropriately as shown in Algorithm 1.

Backfitting. The outcome of running the row and column clustering described above on $\mathbf{R}$ is a single stencil $\mathcal{S}(\mathbf{T}, \mathbf{c}, \mathbf{d})$ consisting of the clusters obtained by first row and then column clustering. It may be desirable to alternate between row and column clustering for further refinement.

However, as noted above, additional stencils only reduce the objective function further - convergence to a local minimum is assured, with the same caveat on solution quality as in k-means clustering. Therefore, we take the residual $\hat{\mathbf{R}} = \mathbf{R} - \mathcal{S}(\mathbf{T}, \mathbf{c}, \mathbf{d})$ and use it as the starting point to learn a new stencil. By repeatedly learning new stencils on the residuals of the previous stencils, as shown in Algorithm 1, we obtain an additive model of co-clusterings that minimizes the approximation error to $\mathbf{R}$.

3.3 Approximation Guarantees

A key question is how well any given matrix $\mathbf{R}$ can be approximated by an appropriate stencil. For the sake of simplicity we limit ourselves to the case where all entries of the matrix are observed. Using covering numbers and the spectral properties of $\mathbf{R}$, we can obtain approximation guarantees for co-clustering. Denote by $\mathbf{\Sigma} = \text{diag}(\sigma_1, \dots \sigma_n)$ the singular values of $\mathbf{R}$.

Theorem 3 (Approximation Guarantees) *Using k clusters for both rows and columns, the matrix $\mathbf{R}$ can be approximated with error at most*

$$\left\|\mathbf{R} - \mathbf{R}'\right\|_\infty \leq 2 \|\mathbf{R}\|^{\frac{1}{2}} \epsilon_k \left(\mathbf{\Sigma}^{\frac{1}{2}}\right)$$

$$\left\|\mathbf{R} - \mathbf{R}'\right\|_2 \leq (\sqrt{N} + \sqrt{M}) \|\mathbf{R}\|^{\frac{1}{2}} \epsilon_k \left(\mathbf{\Sigma}^{\frac{1}{2}}\right)$$

where

$$\epsilon_k(\mathbf{\Sigma}) \leq 6 \sup_{j \in \mathbb{N}} \left(\frac{1}{k} \prod_{i=1}^{j} \sigma_i \right)^{\frac{1}{j}} \leq 6\epsilon_k(\mathbf{\Sigma})$$

PROOF. We omit the proof of Theorem 3 due to space constraints. The complete proof can be found at http://arxiv.org/abs/1501.00199. □

Note that the above is a statement of *existence* rather than a constructive prescription. However, the main purpose of the above analysis is to obtain theoretical upper bounds on the rate of convergence. In practice, the results can be considerably better, as we show in Section 5.

4. GENERATIVE MODEL

As many frequentist algorithms have a Bayesian counterpart, we now devise a Bayesian counterpart to ACCAMS, which we will refer to as bACCAMS. We begin with the single stencil case, describing the model in Section 4.1 and a collapsed Gibbs sampler in Section 4.2. We then extend the model and sampler to many stencils in Section 4.4.

4.1 Co-Clustering with a Single Stencil

We begin with a simple Bayesian model of co-clustering. This is the basic template for single-stencil inference, and our model of additive co-clusters will use the same idea. Our model can be broken into two parts: (1) generating block values and (2) generating cluster assignments. We will go through each part of the model and then the model as a whole, as shown in Figure 2.

Block values. We begin by considering how we generate the prediction $\mathbf{T}_{c,d}$ for a particular block, corresponding to ratings from users in cluster c to movies in cluster d. Previously, as shown in Line 6 of Algorithm 1, each block merely took the average of the values that fell in that block. Subsequently, we would use that block mean directly as the prediction for all values in the block.

In our Bayesian model, we consider each $\mathbf{T}_{c,d}$ to come from a Gaussian distribution, centered at 0 and with variance τ^2. On top of this, each value in the matrix $\mathbf{R}_{u,m}$ is generated by a Gaussian with mean $\mathbf{T}_{\mathbf{c}_u,\mathbf{d}_m}$ and variance σ^2 (alternatively stated, with additive noise $\epsilon_{u,m} \sim \mathcal{N}(0,\sigma^2)$). As such we have

$$\mathbf{T}_{c,d} \sim \mathcal{N}(0,\tau^2) \qquad \mathbf{R}_{u,m} \sim \mathcal{N}\left(\mathbf{T}_{\mathbf{c}_u,\mathbf{d}_m}, \sigma^2\right) \tag{6}$$

Because τ^2 and σ^2 are not yet defined and are data dependent, they too are sampled from the conjugate prior distribution, specifically the Inverse Gamma distribution:

$$\tau^2 \sim \text{IG}(\gamma) \qquad \sigma^2 \sim \text{IG}(\eta) \tag{7}$$

Cluster assignments. In Algorithm 2, a user u (or movie m) is assigned to a particular cluster $\mathbf{c}_u$ ($\mathbf{d}_m$ for movies) based purely on the distance to the cluster center. As with most frequentist algorithms, there is no prior on the cluster assignments.

In our Bayesian model, we put a Dirichlet prior on the cluster assignments. More simply understood, we believe cluster assignments are generated by a Chinese Restaurant Process (CRP). One big advantage of using a CRP is that it allows for an unrestricted number of clusters, where new clusters can always be created with small probability. To understand the Chinese Restaurant Process, consider the following metaphor. A user u is at a Chinese restaurant and trying to pick a table (cluster) at which to sit. Each table is chosen with probability proportional to the number of users already at that table (in that cluster), and a new table is started with some small probability, proportional to α. To be concrete, we assume there are N users, and cluster c has $\mathbf{n}_c$ users in it; $\mathbf{n}_c^{(-u)}$ is the number of users in cluster c not including user u. The probability that user u will go to a particular cluster c is given by:

$$p(\mathbf{c}_u = c | \mathbf{c}^{(-u)}, \alpha) = \begin{cases} \frac{\mathbf{n}_c^{(-u)}}{\alpha+N-1} & \text{if } \mathbf{n}_c^{(-u)} > 0 \\ \frac{\alpha}{\alpha+N-1} & \text{new cluster } c \end{cases} \tag{8}$$

An analogous expression is available for movies: $p(\mathbf{d}_m = d | \mathbf{d}^{(-m)}, \beta)$. Under this model, large values of α and β encourage the formation of larger numbers of clusters.

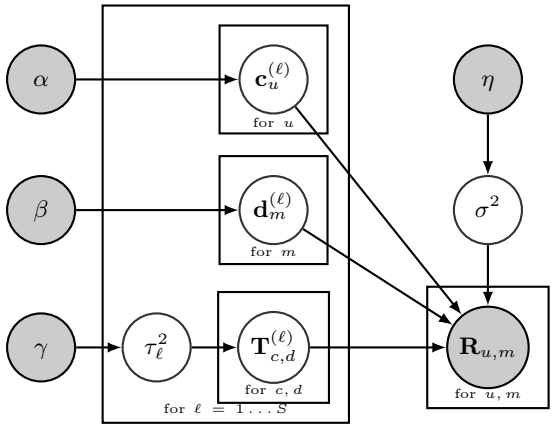


Figure 2: Generative model for recommdnation and matrix approximation (bACCAMS). For each stencil, as indexed by ℓ, row and cluster memberships $\mathbf{c}^{(\ell)}$ and $\mathbf{d}^{(\ell)}$ are drawn from a Chinese Restaurant Process. The values for the template $\mathbf{T}^{(\ell)}$ are drawn from a Normal Distribution. The observed ratings $\mathbf{R}_{u,m}$ are sums over the stencils $\mathcal{S}(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})$.

Complete model. Putting these two elements together, we can now describe our complete Bayesian model for co-clustering:

$$\mathbf{c}_u \sim \text{CRP}(\alpha) \qquad \mathbf{d}_m \sim \text{CRP}(\beta) \tag{9a}$$

$$\mathbf{T}_{c,d} \sim \mathcal{N}(0,\tau^2) \qquad \mathbf{R}_{u,m} \sim \mathcal{N}\left(\mathbf{T}_{\mathbf{c}_u,\mathbf{d}_m}, \sigma^2\right) \tag{9b}$$

$$\tau^2 \sim \text{IG}(\gamma) \qquad \sigma^2 \sim \text{IG}(\eta) \tag{9c}$$

Consequently the joint probability distribution over all ratings, given the variances, is given by

$$p\left(\mathbf{R}, \mathcal{S}(\mathbf{T}, \mathbf{c}, \mathbf{d}) | \alpha, \beta, \sigma^2, \tau^2\right) = \text{CRP}(\mathbf{c}|\alpha)\text{CRP}(\mathbf{d}|\beta) \times \tag{10}$$
$$\prod_{c,d} \frac{1}{\sqrt{2\pi\tau^2}} \exp\left(\frac{-\mathbf{T}_{c,d}^2}{2\tau^2}\right) \prod_{(u,m)} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(\mathbf{R}_{u,m} - \mathbf{T}_{\mathbf{c}_u,\mathbf{d}_m}\right)^2}{2\sigma^2}\right)$$

This is an extremely simple model similar to [21], akin to a decision stump. The rationale for picking such a primitive model is that we will be taking linear combinations thereof to obtain a very flexible tool. We will now show that, by design, the model can be effeciently sampled.

4.2 Collapsed Gibbs Sampler

We now dive into the details of efficiently learning our model through a collapsed Gibbs sampler. There are three main parts of the model that need to be sampled: (1) the cluster assignments $\mathbf{c}$ and $\mathbf{d}$, (2) the block values $\mathbf{T}_{c,d}$, and (3) the variances τ^2 and σ^2. By design, in particular the use of conjugate priors, our model is efficient to sample. For sampling cluster assignments, we find that we can collapse out $\mathbf{T}_{c,d}$ so that sequential samples of cluster assignments are not based on stale values of $\mathbf{T}_{c,d}$ and we achieve a significantly faster sampler.

4.2.1 Inferring Cluster Assignments

We begin with the challenge of sampling the user cluster assignments $\mathbf{c}$ given only the data $\mathbf{R}$, the movie cluster assignments $\mathbf{d}$, and the priors α, σ^2, τ^2. To do this, we must collapse out the block values of $\mathbf{T}$.

In particular, at each step, we check how the likelihood of the data $\mathbf{R}$ changes by assigning a user (or movie) to a new cluster:

$$p(\mathbf{c}_u = c|\mathbf{R}, \mathbf{d}, \alpha, \sigma^2, \tau^2) \propto \mathrm{CRP}(c|\alpha) \frac{p(\mathbf{R}|\mathbf{c}^{(-u)}, \mathbf{c}_u = c, \mathbf{d}, \sigma^2, \tau^2)}{p(\mathbf{R}|\mathbf{c}^{(-u)}, \mathbf{d}, \sigma^2, \tau^2)}$$

We denote the observations for block (c, d), the ratings from users in cluster c to movies in cluster d, by the vector $\mathbf{r}_{c,d}$; the updated ratings in the block after the assignment is given by $\mathbf{r}'_{c,d}$. With this, the calculation above simplifies to:

$$p(\mathbf{c}_u = c|\mathbf{R}, \mathbf{d}, \alpha, \sigma^2, \tau^2) \propto \mathrm{CRP}(c|\alpha) \prod_d \frac{p(\mathbf{r}'_{c,d}|\sigma^2, \tau^2)}{p(\mathbf{r}_{c,d}|\sigma^2, \tau^2)}$$

As we shall see, this is easily calculated by keeping simple linear statistics of the ratings. Moreover, by integrating out $\mathbf{T}$ we avoid the problem of having to instantiate a new value whenever a new cluster is added.

For a given block (c, d) with associated $\mathbf{r}_{c,d}$, the distribution of ratings is Gaussian with mean 0 and with covariance matrix $\mathbf{\Sigma} = \sigma^2\mathbf{1} + \tau^2 11^\top$ (due to the independence of the variances and the additive nature of the normal distribution). Here we use $\mathbf{1}$ to denote the identity matrix and 1 to denote the vector of all 1. Denote by $\mathbf{N}_{c,d}$ the number of rating pairs (u, m) for which $\mathbf{c}_u = c$ and $\mathbf{d}_m = d$. Hence, the likelihood of the block (c, d), as observed in $\mathbf{r}_{c,d}$, is

$$p(\mathbf{r}_{c,d}|\sigma^2, \tau^2) = \frac{\exp\left[-\frac{1}{2}\mathbf{r}_{c,d}^\top \mathbf{\Sigma}^{-1}\mathbf{r}_{c,d}\right]}{(2\pi)^{\frac{\mathbf{N}_{c,d}}{2}} |\mathbf{\Sigma}|^{\frac{1}{2}}}$$

In computing the above expression we need to compute the determinant of $\mathbf{\Sigma}$, a diagonal matrix with rank-1 update, and the inverse of said matrix. For the former, we use the matrix-determinant lemma, and for the latter, the Sherman-Morrison-Woodbury formula:

$$\mathbf{r}_{c,d}^\top \mathbf{\Sigma}^{-1}\mathbf{r}_{c,d} = \frac{1}{\sigma^2}\|\mathbf{r}_{c,d}\|^2 - \frac{\tau^2}{\sigma^2} \cdot \frac{\left(1^\top \mathbf{r}_{c,d}\right)^2}{\sigma^2 + \mathbf{N}_{c,d}\tau^2}$$

$$\log|\mathbf{\Sigma}| = (\mathbf{N}_{c,d} - 1)\log\sigma^2 + \log\left[\sigma^2 + \mathbf{N}_{c,d}\tau^2\right]$$

This allows us to assess whether it is beneficial to assign a user u or a movie m to a different or a new cluster efficiently, since the only statistics involved in the operation are sums of ratings and of their squares, $1^\top \mathbf{r}_{c,d}$ and $\|\mathbf{r}_{c,d}\|^2$ respectively.

We denote by $\mathbf{N}'_{c,d}$ the new cluster count and by $\mathbf{r}'_{c,d}$ the new set of ratings, after having assigned user u to cluster c. Let

$$\Delta := \frac{\mathbf{N}'_{c,d} - \mathbf{N}_{c,d}}{2}\left[\log(2\pi) + \log\sigma^2\right] + \frac{1}{2\sigma^2}\left[\|\mathbf{r}'_{c,d}\|^2 - \|\mathbf{r}_{c,d}\|^2\right]$$

be a constant offset, in log-space, that only depends on the additional ratings that are added to a cluster. That is, Δ is *independent* of the cluster that the additional scores are assigned to and can be safely ignored. The result is:

$$p(\mathbf{c}_u = c|\mathbf{R}, \mathbf{d}, \alpha, \sigma^2, \tau^2) \propto \frac{\mathbf{n}_c^{(-u)}}{\alpha + N - 1} \prod_d \left[\frac{\sigma^2 + \mathbf{N}_{c,d}\tau^2}{\sigma^2 + \mathbf{N}'_{c,d}\tau^2}\right]^{\frac{1}{2}} \times \exp\left[\frac{\tau^2}{2\sigma^2}\sum_d \left[\frac{\left(1^\top \mathbf{r}'_{c,d}\right)^2}{\sigma^2 + \mathbf{N}'_{c,d}\tau^2} - \frac{\left(1^\top \mathbf{r}_{c,d}\right)^2}{\sigma^2 + \mathbf{N}_{c,d}\tau^2}\right]\right] \quad (11)$$

For a new cluster this can be simplified since there is no data, hence $\mathbf{N}_{c,d} = 0$ and $\mathbf{r}_{c,d} = []$.

$$p(\mathbf{c}_u = c_{\text{new}}|\mathbf{R}, \mathbf{d}, \alpha, \sigma^2, \tau^2) \propto \frac{\alpha}{\alpha + N - 1} \prod_d \left[\frac{\sigma^2}{\sigma^2 + \mathbf{N}'_{c,d}\tau^2}\right]^{\frac{1}{2}} \times \exp\left[\frac{\tau^2}{2\sigma^2}\sum_d \frac{\left(1^\top \mathbf{r}'_{c,d}\right)^2}{\sigma^2 + \mathbf{N}'_{c,d}\tau^2}\right] \quad (12)$$

The above expression is fairly straightforward to compute: we only need to track $\mathbf{N}_{c,d}$, i.e. the number of ratings assigned to a particular (user cluster, movie cluster) combination and $1^\top \mathbf{r}_{c,d}$, i.e. the sum of the ratings for this block.

4.2.2 Inferring Block Values

For the purpose of recommendation and for a subsequent combination of several matrices, we need to instantiate the block values $\mathbf{T}_{c,d}$. By checking (10) we see that $\mathbf{T}_{c,d}|\text{rest}$ is given by

$$\mathbf{T}_{c,d}|\text{rest} \sim \mathcal{N}\left(\frac{1^\top \mathbf{r}_{c,d}}{\rho\mathbf{N}_{c,d}}, \frac{\sigma^2}{\rho\mathbf{N}_{c,d}}\right) \text{ where } \rho = \left[1 + \frac{1}{\mathbf{N}_{c,d}}\frac{\sigma^2}{\tau^2}\right] \quad (13)$$

Here too sampling $\mathbf{T}_{c,d}$ only requires having the number of ratings $\mathbf{N}_{c,d}$ and sum of ratings in the block $1^\top \mathbf{r}_{c,d}$.

4.2.3 Inferring Variances

Last, we consider how to sample the variances for the priors, σ^2 and τ^2. Both σ^2 and τ^2 are generated by the Inverse Gamma distribution:

$$p(x|a, b) = b^a \Gamma^{-1}(a) x^{-a-1} e^{-\frac{b}{x}} \quad (14)$$

Denote by E the total number of observed values in $\mathbf{R}$. In this case, σ^2 is drawn from an Inverse Gamma prior with parameters (η'_a, η'_b):

$$\eta'_a \leftarrow \eta_a + \frac{E}{2} \text{ and } \eta'_b \leftarrow \eta_b + \sum_{(u,m)} \mathbf{I}_{u,m}(\mathbf{R}_{u,m} - \mathbf{T}_{\mathbf{c}_u, \mathbf{d}_m})^2 \quad (15)$$

Analogously, we draw τ^2 from an Inverse Gamma with parameters

$$\gamma'_a \leftarrow \gamma_a + \frac{k_n k_m}{2} \text{ and } \gamma'_b \leftarrow \gamma_b + \sum_{c,d} \mathbf{T}_{c,d}^2 \quad (16)$$

k_n and k_m denote the number of user and movie clusters.

4.3 Efficient Implementation

With these inference equations we can implement an efficient sampler, as seen in Algorithm 3. The key to efficient sampling is to cache the per-cluster sums of ratings $1^\top \mathbf{r}_{c,d}$. Then reassigning a user (or movie) to a different

Algorithm 3 StencilSampler($\mathbf{M}, \mathbf{T}, \mathbf{c}, \mathbf{d}$)

Initialize row-index and column-index of data in $\mathbf{M}$
Initialize statistics for each partition
$\mathbf{N}_{c,d} := |\{(u,m) : \mathbf{c}_u = c, \mathbf{d}_m = d\}|$ and $\mathbf{Y}_{c,d} := \sum_{(u,m):\mathbf{c}_u=c,\mathbf{d}_m=d} \mathbf{M}_{u,m}$
while sampler not converged **do**
  for all users u **do**
    For all movie clusters $\mathbf{d}$ compute the incremental changes
    $\mathbf{l}_d^{(u)} := |\{(u,m) : \mathbf{d}_m = d\}|$ and $\mathbf{y}_d^{(u)} := \sum_{(u,m):\mathbf{d}_m=d} \mathbf{M}_{u,m}$
    Remove u from their cluster
    $\mathbf{N}_{\mathbf{c}_u,:} \leftarrow \mathbf{N}_{\mathbf{c}_u,:} - \mathbf{l}^{(u)}$ and $\mathbf{Y}_{\mathbf{c}_u,:} \leftarrow \mathbf{Y}_{\mathbf{c}_u,:} - \mathbf{y}^{(u)}$
    Sample new user cluster $\mathbf{c}_u$ using (11) and (12).
    Update statistics
    $\mathbf{N}_{\mathbf{c}_u,:} \leftarrow \mathbf{N}_{\mathbf{c}_u,:} + \mathbf{l}^{(u)}$ and $\mathbf{Y}_{\mathbf{c}_u,:} \leftarrow \mathbf{Y}_{\mathbf{c}_u,:} + \mathbf{y}^{(u)}$
  end for
  for all movies m **do**
    Sample movie cluster assignments analogously.
  end for
  for all (c,d) cluster partitions **do**
    Resample $\mathbf{T}_{c,d}$ using (13) and $\mathbf{N}_{c,d}, \mathbf{Y}_{c,d}$.
  end for
  Resample σ^2 and τ^2 using (15) and (16).
end while
return $\mathbf{T}, \mathbf{c}, \mathbf{d}$

(or new) cluster is just a matter of checking the amount of change that this would effect. Hence each sampling pass costs $O(k_n \cdot k_m \cdot (N+M) + E)$ operations. It is linear in the number of ratings and of partitions.

Note that once $\mathbf{y}^{(u)}$ and $\mathbf{l}^{(u)}$ are available for all users (or all movies), it is cheap to perform additional sampling sweeps at comparably low cost. It is therefore beneficial to iterate over all users (or all movies) more than once, in particular in the initial stages of the algorithm. Also note that the algorithm can be used on datasets that are being streamed from disk, provided that an index and an inverted index of $\mathbf{M}$ can be stored: we need to be able to traverse the data when ordered by users and when ordered by movies. It is thus compatible with solid state disks.

4.4 Additive Combinations of Stencils

As before, we find that using a linear combination of stencils is far more powerful than just a single stencil. Therefore, we enumerate the stencils by $\mathcal{S}(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})$, where stencil index ℓ ranges from 1 to S. Correspondingly we now need to sample from a set of $\mathcal{S}(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})$ and τ_ℓ^2 *per* stencil. However, we keep the additive noise term $\mathcal{N}(0, \sigma^2)$ unchanged. This is the model of Figure 2. The additivity of Gaussians makes inference easy:

$$\mathbf{R}_{u,m} \sim \mathcal{N}\left(\sum_\ell \mathcal{S}(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})_{u,m}, \sigma^2\right). \quad (17)$$

Note, though, that estimating $\mathcal{S}$ jointly *for all* indices ℓ is not tractable since various clusterings $(\mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})$ overlap and intersect with each other, hence the joint normal distribution over all variables would be expensive to factorize.

Instead, we sample over one stencil at a time, as shown in Algorithm 4. This algorithm only requires repeated passes

Algorithm 4 bACCAMS

initialize residuals $\hat{\mathbf{R}} \leftarrow \mathbf{R}$ and $\mathbf{T}^{(\ell)} = 0\ \forall \ell$
while sampler not converged **do**
  for all stencils $\ell = 1 \dots S$ **do**
    $\hat{\mathbf{R}} \leftarrow \hat{\mathbf{R}} + \mathcal{S}(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})$ {Without stencil ℓ}
    $(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)}) \leftarrow$ StencilSampler($\hat{\mathbf{R}}, \mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)}$)
    $\hat{\mathbf{R}} \leftarrow \hat{\mathbf{R}} - \mathcal{S}(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})$ {With stencil ℓ}
  end for
end while

through the dataset. Moreover, it can be modified into a backfitting procedure by fitting one matrix at a time and then fixing the outcome. Capacity control can be enforced by modifying α and β such that the probability of a new cluster decreases for larger ℓ, i.e. by decreasing α and β. As a result following the analysis in the single stencil case, each sampling pass costs $O(S \cdot (k_n \cdot k_m \cdot (N+M) + E))$ operations. It is linear in the number of ratings, in the number of partitions and in the number of stencils.

5. EXPERIMENTS

We evaluate our method based on its ability to perform matrix completion, matrix approximation and to give interpretable results. Here we describe our experimental setup and results on real world data, such as the Netflix ratings.

5.1 Implementation

We implemented both ACCAMS, the k-means-based algorithm, as well as bACCAMS, the Bayesian model. Unless specified otherwise, we run Algorithm 2 for up to $T = 50$ iterations. Our system can also iterate over the stencils multiple times, such that earlier stencils can be re-learned after we have learned later ones. In practice, we observe this only yields small gains in accuracy, hence we generally do not use it.

We implemented bACCAMS using Gibbs sampling (Section 4.4) and used the k-means algorithm ACCAMS for the initialization of each stencil. Following standard practice, we bound the range of σ by $\sigma_{\max}$ from above. This rejection sampler avoids pathological cases. For the sake of simplicity, we set $k = k_n = k_m$ to be the maximum number of clusters that can be generated in each stencil. When inferring the cluster assignments for a given stencil, we run three iterations of the sampler before proceeding to the next stencil. As common in MCMC algorithms, we use a burn-in period of at least 30 iterations (each with three sub-iterations of sampling cluster assignments) and then average the predictions over many draws. Code for both ACCAMS and bACCAMS is available at http://alexbeutel.com/accams.

5.2 Experimental Setup

Netflix. We run our algorithms on data from a variety of domains. Our primary testing dataset is the ratings dataset from the Netflix contest. The dataset contains 100 million ratings from 480,189 users and 17,770 movies. Following standard practice for testing recommendation accuracy, we average over three different random 90:10 splits for training and testing.

CMU Face Images. To test how well we can approximate arbitrary matrices, we use image data from the CMU Face

Images dataset[1]. It contains black and white images of 20 different people, each in 32 different positions, for a total of 640 images. Each image has 128×120 pixel resolution; we flatten this into a matrix of 640×15360, i.e. an image by pixel matrix.

AS Peering Graph. To assess our model's ability to deal with graph data we consider the AS graph[2]. It contains information on the peering information of 13,580 nodes. It thus creates a binary matrix of size $13,580 \times 13,580$ with 37k edges. Since our algorithm is not designed to learn binary matrices, we treat the entries $\{0, 1\}$ as real valued numbers.

Parameters. For all experiments, we set the hyperparameters in bACCAMS to $\alpha = \beta = 10$, $\eta_\alpha = 2$, $\eta_\beta = 0.3$, $\gamma_\alpha = 5$, and $\gamma_\alpha = 0.3$. Depending on the task, we compare ACCAMS against SVD++ using the GraphChi [12] implementation, SVD from Matlab for full matrices, and previously reported state-of-the-art results.

Model complexity. Since our model is structurally quite different from factorization models, we compare them based on the number of bits in the model and prediction accuracy. For factorization models, we consider each factor to be a 32 bit `float`. Hence the complexity of a rank r SVD++ model of N users and M movies is $32 \cdot r(N + M)$ bits.

For ACCAMS with S stencils and $k \times k$ co-clusters in each stencil, the cluster assignment for a given row or column is $\log_2 k$ bits and each value in the stencil is a `float`. As such, the complexity of a model is $S((N+M)\log_2 k + 32 \cdot k^2)$ bits.

In calculating the parameter space size for LLORMA, we make the very conservative estimate that each row and column is on average part of two factorizations, even though the model contains more than 30 factorizations that each row and column could be part of.

5.3 Matrix Completion

Since the primary motivation of our model is collaborative filtering we begin by discussing results on the classic Netflix problem; accuracy is measured in RMSE. To avoid divergence we set $\sigma_{\max} = 1$. We then vary both the number of clusters k and the number of stencils S.

A summary of recent results as well as results using our method can be found in Table 2. Using GraphChi we run SVD++ on our data. We use the reported values from LLORMA [13] and DFC [16], which were obtained using the same protocol as reported here.

As can be seen in Table 2, bACCAMS achieves the *best* published result. We achieve this while using a very different model that is significantly simpler both conceptually and in terms of parameter space size. We also did not use any of the temporal and contextual variants that many other models use to incorporate prior knowledge.

As shown in Figure 1, we observe that per bit our model achieves much better accuracy at a fraction of the model size. In Figure 3(a) we compare different configurations of our algorithm. As can be seen, classic co-clustering quickly overfits the training data and provides a less fine-grained ability to improve prediction accuracy than ACCAMS. Since ACCAMS has no regularization, it too overfits the training data. By using a Bayesian model with bACCAMS, we do not overfit the training data and thus can use more stencils for prediction, greatly improving the prediction accuracy.

[1]http://www.cs.cmu.edu/~tom/faces.html
[2]http://topology.eecs.umich.edu/data.html

Method	Parameters	Size	Test RMSE
SVD++ [12]	$R = 25$	49.8MB	0.8631
DFC-NYS [16]	Not reported		0.8486
DFC-PROJ [16]	Not reported		0.8411
LLORMA [13]	$R = 1$	3.98MB	0.9295
LLORMA [13]	$R = 5$, $a > 30$	19.9MB	0.8604
LLORMA [13]	$R = 10$, $a > 30$	39.8MB	0.8444
LLORMA [13]	$R = 20$, $a > 30$	79.7MB	0.8337
ACCAMS	$k = 10$, $s = 13$	2.69MB	0.8780
ACCAMS	$k = 100$, $s = 5$	2.27MB	0.8759
bACCAMS	$k = 10$, $s = 50$	10.4MB	0.8403
bACCAMS	$k = 10$, $s = 70$	14.5MB	0.8363
bACCAMS	$k = 10$, $s = 125$	**25.9MB**	**0.8331**

Table 2: bACCAMS achieves an accuracy for matrix completion on Netflix better than or on-par with the best published results, while having a parameter space a fraction of the size of other methods. a denotes the number of anchor points for LLORMA and sizes listed are the parameter space size.

5.4 Matrix Approximation

In addition to matrix completion, it is valuable to be able to approximate matrices well, especially for dimensionality reduction tasks. To test the ability of ACCAMS to model matrix data we analyze both how well our model fits the training data from the Netflix tests above as well as on image data from the CMU Faces dataset and a binary matrix from the AS peering graph. (Note, for Netflix we now use the training data from one split of the dataset.) For each of these of datasets we compare to the SVD (or SVD++ to handle missing values). We also use our algorithm to perform classic co-clustering by setting $S = 1$ and varying k.

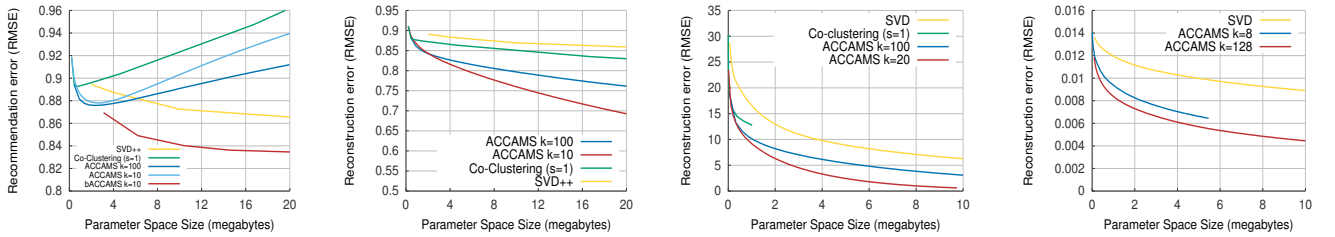
As can be seen in Figure 3(b-d), ACCAMS models the matrices from all three domains much more compactly than SVD (or SVD++ in the case of the Netflix matrix, which contains missing values). In particular, we observe on the CMU Faces matrix that ACCAMS uses in some cases under $\frac{1}{4}$ of the bits as SVD for the same quality matrix approximation. Additionally, we observe that using a linear combination of stencils is more efficient to approximate the matrices than performing classic co-clustering where we have just one stencil. Ultimately, although the method was not designed specifically for image or network data, we observe that our method is effective for succinctly modeling the data.

5.5 Interpretability

In any model the structure of factors makes assumptions about the form of user preferences and decision making. The fact that our model uses a smaller parameter space while achieving an *improvement* in the generalization error suggests that our modeling assumptions better match the underlying data generation (how people make decisions). One advantage of our model being compact and conceptually simple is that we can understand our learned parameters.

To test the model's interpretability we use ACCAMS to model the Netflix data with $S = 20$ stencils and $k^2 = 100$ clusters (a model of similar size to a rank-3 matrix factorization). Here we look at two ways to interpret the results.

First we view the cluster assignments in stencils as inducing a hierarchy on the movies. That is, movies are split in the first level based on their cluster assignments in the first



(a) Netflix Prediction Error (b) Netflix Training Error (c) Faces Approximation (d) AS Graph Approximation

Figure 3: On images, ratings, and binary graphs, ACCAMS approximates the matrix more efficiently than SVD, SVD++, or classic co-clustering.

stencil. At the second level, we split movies based on their cluster assignments in the second stencil, etc. In Figure 4 we observe the hierarchy of TV shows induced by the first three stencils learned by ACCAMS (we only include shows where there is more than one season of that show in the leaf and we pruned small partitions due to space restrictions).

As can be seen in the hierarchy, there are branches which clearly cluster together shows more focused on male audiences, female audiences, or children. However, beyond a first brush at the leaf nodes, we can notice some larger structural differences. For example, looking at the two large branches coming from the root, we observe that the left branch generally contains more recent TV shows from the late 1990s to the present, while the right branch generally contains older shows ranging from the 1960s to the mid 1990s. This can be most starkly noticed by "Friends," which shows up in both branches; Seasons 1 to 4 of "Friends" from 1994-1997 fall in the older branch, while Seasons 5 to 9 from 1998-2002 fall in the newer branch. Of course the algorithm does not know the dates the shows were released, but our model learns these general concepts just based on the ratings. From this it is clear the stencils can be useful for breaking down content in a meaningful structured way, something that is not possible under classic factorization approaches.

While the hierarchy demonstrates that our stencils are learning meaningful latent factors, it may be difficult to always understand individual clusters. Rather, to use knowledge from *all* of the stencils, we can look to the use case of "Users who watched X also liked Y," and ask given a movie or TV show to search, can we find other similar items? We do this by comparing the set of cluster assignments from the given movie to the set of cluster assignments of other items. We measure similarity between two movies using the Hamming distance between cluster assignments.

As can be seen in Table 5, we find the combination of clusters for different movies and TV shows can be used to easily find similar content. While we see some obvious cases where the method succeeds, e.g. "Sex and the City" returns six more seasons of "Sex and the City," we also notice the method takes into account more subtle similarities of movies beyond genre. For example, while the first season of "Seinfeld" returns the subsequent seasons of "Seinfeld," it is followed by three seasons of "Curb Your Enthusiasm," another comedy show by the same writer Larry David. Similarly, searching for Stanley Kubrick's "2001: A Space Odyssey" returns other Stanley Kubrick movies, as well as other critically acclaimed films from that era, particularly thematically similar science fiction movies. Searching for "Scooby-Doo" returns topically similar children's shows, specifically from

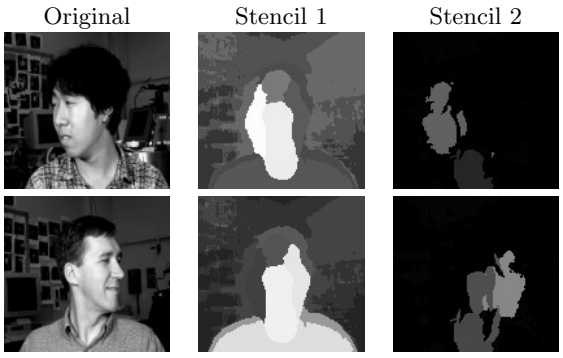


Figure 6: Examples of original images and the first two stencils. The decomposition is very similar to that of eigenfaces [22], albeit much more concise in its nature.

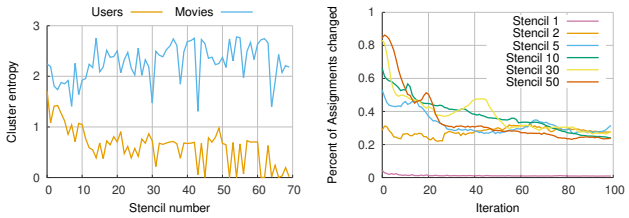


Figure 7: Left: Entropy in cluster assignments for users and movies. Right: Stability of the assignments in the sampler.

the mid to late 1900's. From this we get a sense that ACCAMS does not just find similarity in genre but also more subtle similarities.

5.6 Properties of ACCAMS

Aside from ACCAMS's success across matrix completion and approximation, it is valuable to understand how our method is working, particularly because of how different it is from previous models. First, because ACCAMS uses backfitting, we expect that the first stencil captures the largest features, the second captures secondary ones, etc. This idea is backed up by our theoretical results in Section 3.3, and we observe that this is working experimentally by the drop off in RMSE for our matrix approximation results in Figure 3. We can visually observe this in the image approximation of the CMU Faces. As can be seen in Figure 6, the first stencil captures general structures of the room and heads, and the second starts to fill in more fine grained details of the face.

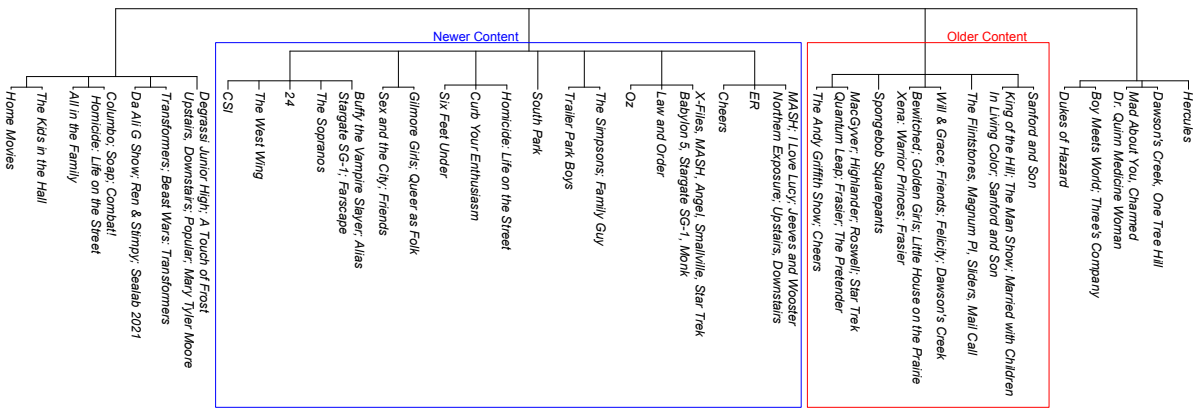


Figure 4: Hierarchy of TV Shows on Netflix based on the first three stencils generated by ACCAMS.

2001: A Space Odyssey	Sex and the City: Season 1	Seinfeld: Seasons 1 & 2	Mean Girls
Taxi Driver	Sex and the City: Season 2	Seinfeld: Season 3	Clueless
Chinatown	Sex and the City: Season 3	Seinfeld: Season 4	13 Going on 30
Citizen Kane	Sex and the City: Season 4	Curb Your Enthusiasm: Season 1	Best in Show
Dr. Strangelove	Sex and the City: Season 5	Curb Your Enthusiasm: Season 2	Particles of Truth
A Clockwork Orange	Sex and the City: Season 6.1	Curb Your Enthusiasm: Season 3	Charlie's Angels: Full Throttle
THX 1138: Special Edition	Sex and the City: Season 6.2	Arrested Development: Season 1	Amelie
Apocalypse Now Redux	Hercules: Season 3	Newsradio: Seasons 1 and 2	Me Myself I
The Graduate	Will & Grace: Season 1	The Kids in the Hall: Season 1	Bring it On
Blade Runner	Beverly Hills 90210: Pilot	The Simpsons: Treehouse of Horror	Chaos
The Deer Hunter	The O.C.: Season 1	Spin City: Michael J. Fox	Kissing Jessica Stein
Deliverance	Divine Madness	Curb Your Enthusiasm: Season 4	Nine Innings from Ground Zero

Star Wars: Episode V	The Silence of the Lambs	Scooby-Doo Where Are You?	Law & Order: Season 1
Star Wars: Episode IV	The Sixth Sense	The Flintstones: Season 2	Law & Order: Season 3
Star Wars: Episode VI	Alien: Collector's Edition	Classic Cartoon Favorites: Goofy	Law & Order: SVU (2)
Battlestar Galactica: Season 1	The Exorcist	Transformers: Season 1 (1984)	Law & Order: Criminal Intent (3)
Raiders of the Lost Ark	Schindler's List	Tom and Jerry: Whiskers Away!	Law & Order: Season 2
Star Wars: Clone Wars: Vol. 1	The Godfather	Boy Meets World: Season 1	MASH: Season 8
Gladiator: Extended Edition	Seven	The Flintstones: Season 3	ER: Season 1
Star Wars Trilogy: Bonus Material	Colors	Scooby-Doo's Greatest Mysteries	MASH: Season 7
LOTR: The Fellowship of the Ring	The Godfather, Part II	Care Bears: Kingdom of Caring	Rikki-Tikki-Tavi
LOTR: The Two Towers	GoodFellas: Special Edition	Aloha Scooby-Doo!	The X-Files: Season 6
Indiana Jones Trilogy: Bonus Material	Platoon	Scooby-Doo: Legend of the Vampire	The X-Files: Season 7
LOTR: The Return of the King	Full Metal Jacket	Rugrats: Decade in Diapers	ER: Season 3

Figure 5: For a given movie or TV show on Netflix, we can use the cluster assignments to find related content.

The Bayesian model, bACCAMS, backfits in the first iteration of the sampler but ultimately resamples each stencil many times thus loosening these properties. In Figure 7, we observe how the distribution of users and movies across clusters changes over iterations and number of stencils, based on our run of bACCAMS with $S = 70$ stencils and a maximum of $k = 10$ clusters per stencil. As we see in the plot of entropy, movies, across all 70 stencils, are well distributed across the 10 possible clusters. Users, however, are well distributed in the early stencils but then are only spread across a few clusters in later stencils. In addition, we notice that while the earlier clusters are stable, later stencils are much less stable with a high percentage of cluster assignments changing. Both of these properties follow from the fact that most users rate very few movies. For most users only a few clusters are necessary to capture their observed preferences. Movies, however, typically have more ratings and more latent information to infer. Thus through all 70 stencils we learn useful clusterings, and our prediction accuracy improves through $S = 125$ stencils.

6. DISCUSSION

Here we formulated a model of additive co-clustering. We presented both a k-means style algorithm, ACCAMS, as well as a generative Bayesian non-parametric model with a collapsed Gibbs sampler, bACCAMS; we obtained theoretical guarantees for matrix approximation through additive co-clustering; and we showed that our method is concise and accurate on a diverse range of datasets, including achieving the best published accuracy for matrix completion on Netflix.

Given the novelty and initial success of the method, we believe that domain-specific variants of ACCAMS, such as for community detection and topic modeling, can and will lead to new models and improved results. In addition, given the modularity of our framework, it is easy to incorporate side information, such as explicit genre and actor data, in modeling rating data that should lead to improved accuracy and interpretability.

Acknowledgements. We would like to thank Christos Faloutsos for his valuable feedback throughout the preparation of this paper. This research was supported by funds from Google, a Facebook Fellowship, a National Science Foundation Graduate Research Fellowship (Grant No. DGE-1252522), and the National Science Foundation under Grant No. CNS-1314632 and IIS-1408924. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties.

References

[1] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing. Mixed-membership stochastic blockmodels. *Journal of Machine Learning Research*, 9:1981–2014, 2008.

[2] A. Banerjee, I. Dhillon, J. Ghosh, S. Merugu, and D. S. Modha. A generalized maximum entropy approach to bregman co-clustering and matrix approximation. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 509–514. ACM, 2004.

[3] A. Beutel, K. Murray, C. Faloutsos, and A. J. Smola. CoBaFi: Collaborative Bayesian Filtering. In *World Wide Web Conference*, pages 97–108, 2014.

[4] A. Gittens and M. W. Mahoney. Revisiting the nyström method for improved large-scale machine learning. In *ICML (3)*, pages 567–575, 2013.

[5] D. Görür, F. Jäkel, and C. E. Rasmussen. A choice model with infinitely many latent features. In *Proceedings of the 23rd international conference on Machine learning*, pages 361–368. ACM, 2006.

[6] T. Griffiths and Z. Ghahramani. The indian buffet process: An introduction and review. *Journal of Machine Learning Research*, 12:1185âĹŠ1224, 2011.

[7] N. Halko, P. Martinsson, and J. A. Tropp. Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions, 2009. oai:arXiv.org:0909.4061.

[8] J. A. Hartigan. Direct clustering of a data matrix. *Journal of the american statistical association*, 67(337):123–129, 1972.

[9] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Knowledge discovery and data mining KDD*, pages 426–434, 2008.

[10] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[11] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos. *VOG: Summarizing and Understanding Large Graphs*, chapter 11, pages 91–99. SIAM, 2014.

[12] A. Kyrola, G. Blelloch, and C. Guestrin. GraphChi: Large-scale graph computation on just a pc. In *OSDI*, Hollywood, CA, 2012.

[13] J. Lee, S. Kim, G. Lebanon, and Y. Singer. Local low-rank matrix approximation. In *Proceedings of The 30th International Conference on Machine Learning*, pages 82–90, 2013.

[14] M. Leeuwen, J. Vreeken, and A. Siebes. Identifying the components. *Data Mining and Knowledge Discovery*, 19(2):176–193, 2009.

[15] M. Li, G. L. Miller, and R. Peng. Iterative row sampling. In *FOCS 2013*, pages 127–136, 2013.

[16] L. W. Mackey, M. I. Jordan, and A. Talwalkar. Divide-and-conquer matrix factorization. In *Advances in Neural Information Processing Systems*, pages 1134–1142, 2011.

[17] K. Palla, D. Knowles, and Z. Ghahramani. An infinite latent attribute model for network data. In *International Conference on Machine Learning*, 2012.

[18] E. E. Papalexakis, N. D. Sidiropoulos, and R. Bro. From k-means to higher-way co-clustering: multilinear decomposition with sparse latent factors. *Signal Processing, IEEE Transactions on*, 61(2):493–506, 2013.

[19] I. Porteous, E. Bart, and M. Welling. Multi-HDP: A non parametric bayesian model for tensor factorization. In D. Fox and C. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 1487–1490. AAAI Press, 2008.

[20] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[21] H. Shan and A. Banerjee. Bayesian co-clustering. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 530–539. IEEE, 2008.

[22] M. Turk and A. Pentland. Face recognition using eigenfaces. In *Proceedings CVPR*, pages 586–591, Hawaii, June 1991.

[23] J. Vreeken, M. Leeuwen, and A. Siebes. Krimp: mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011.

[24] P. Wang, K. B. Laskey, C. Domeniconi, and M. I. Jordan. Nonparametric bayesian co-clustering ensembles. In *SDM*, pages 331–342. SIAM, 2011.