Sparse Models for Sparse Data:
Methods, Limitations, Visualizations and Ensembles

By
David Gregory Purdy

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy
in
Statistics
and the Designated Emphasis in
Communication, Computation, and Statistics
in the Graduate Division
of the
University of California, Berkeley

Committee in charge:
Professor Bin Yu, Chair
Professor Haiyan Huang
Professor Laurent El Ghaoui

Fall 2012
Abstract
Sparse Models for Sparse Data
by
David Gregory Purdy
Doctor of Philosophy in Statistics
Designated Emphasis in Communication, Computation, and Statistics
University of California, Berkeley
Professor Bin Yu, Chair

Significant recent advances in many areas of data collection and processing have introduced many challenges for modeling such data. Data sets have exploded in the number of observations and dimensionality. The explosion in dimensionality has led to advances in the modeling of high dimensional data with regularized and sparse models. One of the more interesting and challenging varieties of high dimensional data are sparse data sets. Sparse data sets arise from many important areas involving human-computer interaction, such as text and language processing, and human-human interaction, such as social networks.

Our motivation in this thesis is to explore the use of sparse models for applications involving sparse data. In some cases, we have made improvements over previous methods that fundamentally involved dense models fitted on, and applied to, sparse data. In other cases, we have adapted sparse models developed for dense data sets. Along the way, we have encountered a recurring issue: due to both subsampling and regularization, we are faced with a problem that sparse models may not adequately capture the full dimensionality of such data and may be inadequate for prediction on test data.

The utility of sparse models have been demonstrated in contexts with very high dimensional dense data. In this dissertation, we shall examine two applications and modeling methods involving sparse linear models and sparse matrix decompositions. Our first application involves natural language processing and ranking, the second involves recommendation systems and matrix factorization.

In Chapter 2, we developed a novel and powerful visualization system. We named our system Bonsai as it enables a curated process of developing trees that partition the joint space of data and models. By exploring the product space of the space of training data, the space of modeling parameters, and the space of test data, we can explore how our models are developed based on the constraints imposed and the data they attempt to model or predict. More generally, we
believe we have introduced a very fruitful means of exploring a multiplicity of models and a multiplicity of data samples.

Chapter 3 is based on our work in the Netflix Prize competition. In contrast to others' use of dense models for this sparse data, we sought to introduce modeling methods with tunable sparsity. In this work, we found striking difficulties in modeling the data with sparse models, and identified concerns about the utility of sparse models for sparse data.

In conclusion, this thesis presents several methods, and limitations of such methods, for modeling sparse data with sparse models. These limitations are suggestive of new directions to pursue. In particular, we are optimistic that future research in modeling methods may find new ways to tune models for density, when applied to sparse data, just as much research on models for dense data has involved tuning models for sparsity.
Introduction

Significant recent advances in many areas of data collection and processing have introduced many challenges for modeling such data. Data sets have exploded in the number of observations and dimensionality. The explosion in dimensionality has led to advances in the modeling of high dimensional data with regularized and sparse models. One of the more interesting and challenging varieties of high dimensional data are sparse data sets. Sparse data sets arise from many important areas involving human-computer interaction, such as text and language processing, and human-human interaction, such as social networks. While there are certainly other contexts in which sparse data may arise, these types of contexts challenge us to learn about a great variety of human behavior with data matrices that are far from completely observed.

Our motivation in this thesis is to explore the use of sparse models for applications involving sparse data. In some cases, we have made improvements over previous methods that fundamentally involved dense models fitted on, and applied to, sparse data. In other cases, we have adapted sparse models developed for dense data sets. Along the way, we have encountered a recurring issue: due to both subsampling and regularization, we are faced with a problem that sparse models may not adequately capture the full dimensionality of such data and may be inadequate for prediction on test data.

The utility of sparse models have been demonstrated in contexts with very high dimensional dense data, for applications such as sparse linear models and sparse matrix decompositions. In this dissertation, we shall examine two applications, involving both of these types of sparse models, namely sparse linear models and sparse matrix decompositions. Our first application, in Chapter 1, involves ranking; our second, in Chapter 3, involves imputation of selected unobserved entries. The methods used in these chapters will involve variations on the linear models and matrix decompositions that others have addressed.

We will see that there are at least two types of sparse data and that our methods for fitting models and evaluating models are affected by data sparsity. We may briefly summarize these problems by contrasting with dense matrices.

For a dense matrix, it is common to assume that it is fully observed, typically with informative variation per variable. In contrast, sparse matrices are often defined as having many entries of 0. We extend this notion with sparsely populated matrices, i.e. matrices for which few entries have been observed. In short, the former type of sparsity arises from observed entries that are mostly 0, and the latter type arises from having many unobserved entries. In Chapter 1, our interest involves the former type of sparsity, while Chapter 3

\footnote{Although sparsely populated matrices are analogous to missing data, we avoid that terminology to avoid the assumption that such data is missing, as it was never collected, and may be collected in the future.}
Involves the latter, as our objective is to impute some of the unobserved entries.

In this dissertation, we will demonstrate two problems in the interaction of sparse matrices and sparse models: sampling and modeling. Regarding sampling, realize that in a sparse matrix, $X \in \mathbb{R}^{N \times P}$, many columns will have few non-zero or observed entries. If we sample rows from this matrix, producing a sample $X_S$, $S \subset \{1, \ldots, N\}$, it is often likely that some of our columns will have no information to impart: the column rank of the sub-sample will be less than the column rank of the full matrix, i.e. $\text{colrank}(X_S) \leq \text{colrank}(X)$. The effect of sparsity of data and model regularization can be contrasted with dense linear model for projecting $X_S$ onto some response, say, $Y_S$, (e.g. a linear regression model, such as $Y_S = X_S \beta_S$): the support, i.e. the number of non-zero entries, of our linear vector $\beta_S$ will be, at most, the same as the column rank of $X_S$. However, if we seek a sparse model, $\beta_S^*$, then its support could be far less than the $\text{colrank}(X)$. In the complement of $X_S$, namely any case where the dimensions of $X$ are not addressed by the model $\beta_S$, we must admit that the model is inadequate.

In Chapter 2, we attempted to understand the model inadequacy encountered in Chapter 1, in a project on sparse models for ranking data from natural language processing. We sought to improve upon our models by understanding their dependence on the data and our choices of modeling parameters. In the process, we developed an entirely new visualization system, allowing us to explore the joint space of data and models. We named our system Bonsai as it enables a curated process of developing trees that partition the joint space of data and models. By exploring the product space of the space of training data, the space of modeling parameters, and the space of test data, we can explore how our models are developed based on the constraints imposed and the data they attempt to model or predict. This allowed us to identify the limitations of our models in Chapter 1. More generally, we believe we have introduced a very fruitful means of exploring a multiplicity of models and a multiplicity of data samples.

Chapter 3 is based on our work in the Netflix Prize competition. The competition involved predicting users’ ratings for a selection of movies, based on a very sparsely populated training matrix of ratings the users had given to films. Other competitors explored dense models for this sparse data, while we sought to introduce sparse models with parameters that could allow the models to become as dense as others’ methods. In this case, we found striking difficulties in modeling the data with sparse models, which introduces new concerns about the utility of sparse models for sparse data.

In conclusion, this thesis presents several methods, and limitations of such methods, for modeling sparse data with sparse models. These limitations are suggestive of new directions to pursue. In particular, we are optimistic that future research in modeling methods may find new ways to tune models for density, when applied to sparse data, just as much research on models for dense data has involved tuning models for sparsity.

\footnote{Although Bonsai was inspired by the work of Chapter 1, the development and implementation of the Bonsai system was a collaboration with Daisy Zhe Wang, and was presented in 2008 at the Joint Statistics Meetings, in Denver, CO.}
Acknowledgements

Throughout graduate school and the completion of this dissertation, I have benefitted from the help and encouragement of a great many people. In a dissertation on sparse data and models, the concept of support will be seen again and again. It is very fitting that I acknowledge the support of so many in making this dissertation possible.

Working with Bin Yu, my advisor, has been the greatest reward of this experience. I came to graduate school in order to learn how to ask the right questions and how to answer them using existing methods, and to understand how to develop new methods when existing methods may not suffice. I am very grateful to Bin for this experience and for her support and guidance throughout. As a statistician, she is great, and as an advisor, she is phenomenal. A great statistician finds insights into data and can transform them meaningfully. A great advisor finds insights into people and can transform them meaningfully. Bin, thank you very, very much.

My dissertation and qualifying committee members, Haiyan Huang, Laurent El Ghaoui, John Rice, and Dan Klein, have all contributed to the depth and breadth of my understanding of statistical methods and applications. Their feedback on my research and the expertise they have shared are very much appreciated.

My wife, Jing (Annie) Wang, and my father, Jim Purdy, were very supportive before and during graduate school and throughout the completion of this dissertation. They have read numerous revisions, and have been consistently reminding me, weekly or even daily, to make progress in finishing this dissertation. I am exceptionally appreciative of their enduring and persistent support. Annie has been the most wonderful companion through this long journey and for all of the other adventures that have occurred along the way, from our wedding and honeymoon, to the birth of our son, Aiden. This is but one journey we have undertaken together, and I am glad to have so many more ahead. I am also very grateful for Annie’s parents and the help they provided when our son was born, and in numerous other ways, so that I could focus on my research and dissertation.

My brother, Steven, has also been a reliable source of encouragement and guidance over the many years. My aunts, uncles, and cousins, who have traveled far and wide to support me in graduate school and in life have been incredible in helping me stay on course, celebrating achievements and helping me navigate more challenging times.

Daisy Wang, my collaborator on the Bonsai project, played a great role in helping me transform data and models into visualizations. Working with her expanded my outlook on both my own research and the potential applications to come for a great many people, when databases meet statistics and machine learning. I am grateful to have had the opportunity to work with her and for the friendship that developed.
I twice took David Freedman’s statistical consulting course. I aimed to learn about statistical applications and methods, but I also learned what may be the hardest lesson for a graduate student at Berkeley: easily and openly admitting that one does not know something. I also appreciated his ability to enlighten students and transform their understanding of statistics by identifying and confronting the gaps in their knowledge. I will gladly admit that I do not know how to state my appreciation in just a few words.

I appreciate the assistance of Microsoft Research Asia, and Gao Jianfeng and Hisami Suzuki, in providing the data used in Chapter 1, the IME project. I am very grateful to the bold commitment of Netflix in offering the Netflix Prize competition and the data they released. Participating in the Netflix Prize was an incredible experience, especially the experience of working with my teammates and friends from The Ensemble, especially Lester Mackey, a fellow Berkeley graduate student.

Many friends and classmates have participated in my development and have played a role in encouraging me throughout my studies and my research. Friends such as Ivan Matic, Ron Peled, Kaushik Katari, Eleanor Hennessey, David Rosenberg, Nancy Wang, Frances Tong, Daisy Wang, Seth Roberts, Nate Coehlo, Vince Vu, Guilherme Rocha, Peng Zhao and many others made the experience more exciting and rewarding. We know that chance brought us together, but friendship and passion are what make our shared experiences so much more than just shared experiences.

Getting to the finish line also requires the work of many whose contributions are realized at the most stressful times in a graduate student’s life: many staff inside and outside the department. Ryan Lovett kept the SCF computers humming, despite the onslaught of my distributed computing jobs. We learned much together, and I am very grateful that he never banned me from the computers, instead working with me to scale to ever larger computing challenges. Angie Fong’s humor and insight are unforgettable and very much appreciated. In my short experience working with La Shana Porlaris, I have become much indebted to her for her kind and rapid assistance and very positive encouragement.

I return to my family: my family members’ attendance at my graduation ceremony is deeply appreciated. It was one of the last times my grandmother was able to travel for a family event. My family has shrunk and grown during the course of my graduate experience: I married a wonderful woman, elder relatives have passed away, and my son was born in November 2011.

There is one person whom I would most like to acknowledge, who never knew of my graduate experience, was never able to attend my wedding, the birth of my son, Aiden, nor see the completion of this dissertation. That is my mother, Linda Purdy. When I was young, I probably thought of statistics as being vital statistics. Over time, I realize that she taught me that such statistics are a projection of the texture of our lives: from deaths and births, we know of sadness and joy, of statistics reporting unemployment and education, we know of discouragement and optimism. However, she taught me much more than this. She encouraged me to explore, to go beyond my strengths and find those areas where I will fail, and accept that failure precedes success, and, given time, will again follow success. If we knew the path where life would take us, would we follow that path? What if we knew whom we would have as companions for that journey? While I wish that my mom could have seen this journey from end to end, I have been very fortunate to share it with so many wonderful people, and my mom has been a part of it all along the way.
Chapter 1

Ranking

1.1 Overview

Communication among people has distinguished our species for its permanence, propagation, and pervasiveness: we can convey information for longer periods of time, to more recipients, and from more sources, than ever before. Written communication has evolved over time and now much of our written communication occurs through computers; because of its scale, we now rely upon computers and statistical models for processing massive volumes of information. In this chapter, we are concerned with the first step: the interface between a single human and a computer, when the user enters textual information into the computer.

Many people see text entry as simple: their language has an alphabet of a few dozen characters, a keyboard can support many dozens of characters, so all of their text needs are addressed with a standard keyboard. However, for ideographic languages, such as Chinese and Japanese, there are far more characters than can be represented on a keyboard, necessitating software, rather than hardware, methods for facilitating text input. A common solution realizes that most languages can be phonetically transliterated via the Roman alphabet, e.g. Romaji for Japanese and Pinyin for Chinese. As many possible expressions can have the same pronunciation, the mapping from the phonological input to textual representation is not unique, and a computer program will offer a set of “guesses” at the intended text. In statistical modeling and machine learning, we develop predictive models, but this problem of sifting the possible representations involves neither classification nor regression, the two major modeling techniques from statistical machine learning; instead, this involves ranking, a task both separate from and related to regression and classification.

Our motivation for improving the ranking of candidate sentences is that users will benefit by being able to more efficiently type text in languages with complex scripts, as less effort will be expended to select among candidate transliterations or to revise a proposed transliteration to fix errors. Users’ efficiency and satisfaction with the input method are the primary motivations for transliteration software; improving the ranking of candidates can benefit both goals.

The comparison of ranking with more common machine learning methods, namely classification and regression, is informative. An extensive survey of machine learning methods is beyond the scope of this chapter, and is well addressed in books such as [10]. Instead, in
order to address the tools we will need for rank modeling, and the challenges faced in our work on sparse data, we will adopt a framework introduced by Hand, Mannila, and Smyth, in [9], who described the key components of a machine learning or data mining algorithm as: (1) an assumed model or pattern structure (i.e. a model family), (2) a score function (e.g. sum of squared errors), (3) an optimization and search method for finding the optimal model in the model family, and (4) a data management strategy. The last component, the data management strategy, is often given less significance from a methodological standpoint, though it will impact our work from several directions: data preparation, anomalies in the data, and in both the notation and mathematics of our algorithms and in the computational considerations involved in implementing our algorithms efficiently. In fact, this component, while listed last by Hand et al., will be the first that we will need to address. In addition to these four components of the framework, we would like to note another critical consideration: data – as we shall see, the selection of data plays a significant role in the nature of the optimal model fitted by a machine learning algorithm; moreover, we shall see cases where training and testing subsets arise from different distributions, thereby affecting our choice of loss functions.

We will address each of these components in developing our work on models for ranking. First, we should address the structural differences in the data and functions that arise in ranking, relative to typical regression and classification contexts, as this will have a number of unique implications. In ranking, the objective is to map from a set of observation vectors, e.g. \( \{X_i \in \mathbb{R}^P\}_{i=1,...,N} \), to either a permutation of the values \( \{1,\ldots,N\} \) or an ordering of scores associated with the observations, e.g. \( Y_1 \leq Y_2 \leq \ldots \leq Y_N \). In contrast, in typical regression and classification contexts, the goal is to find a function that maps a single observation, e.g. \( X \in \mathbb{R}^P \), to a single response, e.g. \( Y \in \mathbb{R} \) or \( Y \in \{0,1\} \).

In addition to addressing the structure of the data, we will find that the ranking context also affects our choices of evaluation, or loss, functions. In other contexts, we typically assume a family of models, \( \mathcal{F} \) and a loss function, \( L \), and typically seek to find a function, \( f^* \), which is optimal among all functions in its family, i.e. among all \( f \in \mathcal{F} \), in that it minimizes the expected loss. In particular, we may choose a function \( f^* \) as \( f^* = \arg \min_{f \in \mathcal{F}} E(L(f(X), Y)) \), where both \( X \) and \( Y \) can be described as arising from individual observations. In contrast, the ranking context involves a function that maps sets to sets, i.e. a set of observations to a set of responses, so the loss is calculated relative to the set, or a subset, of candidates. This additional complexity necessitates additional notation, as we need to think in terms of sets of observations. Notation is sufficiently important that we will address it in the next section, Section 3.6.

As addressed in the beginning of this section, the emphasis of this chapter is on the application of ranking to the problem of text input, namely the reranking of candidates in a Japanese text transliteration program, known as an Input Method Editor (IME). We will address the methods we have developed for improving the quality of the ranking of the transliteration candidates, and analyses of the results obtained. Analyses of the results has led to several insights on both the basis for improvements made in the ranking task and the limits associated with these methods.

This chapter has 10 sections. In Section 3.6, we address notation appropriate to ranking; Section 1.3 introduces the reranking application at the focus of our investigation; Section 1.4 addresses the over-arching goal of this research as well as the unique challenges for this
particular data set.

Section 1.5 addresses how we can evaluate our models and improvements. Section 3.8 introduces modeling methods, including structural models, loss functions and search methods for finding the model with an optimal loss.

Section 3.5 presents a summary of extensive exploratory data results, and we report our modeling results in Section 3.9. In Section 3.10, we discuss insights from exploring the errors and suggestions for further work. We conclude, in Section 3.11, with analysis of our applied work on this application, while leaving further methodological work for two subsequent chapters.
1.2 Preliminaries

Before delving into the ranking application and the methods used, several basic notational issues need to be addressed in order to discuss the data and modeling methods.

In traditional statistical and machine learning applications, we typically denote the data as a set of observations, \( X = \{X_i \in \mathbb{R}^p\}_{i=1,\ldots,N} \) and corresponding responses, \( Y = \{Y_i \in \mathbb{R}\} \); i.e., we often assume that each observation has \( p \) covariates (e.g. \( X_i \in \mathbb{R}^p \)) and the response is 1 dimensional (e.g. \( Y_i \in \mathbb{R} \) or \( \{0,1\} \)). In the ranking context, the basic unit of data is a set of candidates, rather than a single observation, with each candidate described in terms of its covariates, \textit{and in relation to the other candidates in the same set}. In other words, we have \( N \) sets, with \( J_i \) candidates per set, where \( i = 1, \ldots, N \) is used to index the sets of candidates and \( j = 1, \ldots, J_i \) indexes the candidates in set \( i \). For simplicity, we assume that constant dimensionality for all of our data, e.g. we assume \( X_{i,j} \in \mathbb{R}^p, \forall i, \forall j \), rather than address the issue of disjoint spaces for different candidate sets. While our notation allows for flexibility in the sizes of candidate sets, in our data, \( J_i \) is constant: all of our candidate sets have 20 candidates each, so we can use a constant value, \( J \) or 20, to represent the size of each candidate set. Conversely, while our notation assumes the feature space is constant across all candidate sets and candidates, we need not assume that all candidates (or sets) use all of the feature space. In fact, our data set is very sparse, so only a small fraction of the dimensions are used by any candidate. Finally, our full data set has \( N = 80,000 \) candidate sets.

We describe our notation as follows:

1. We will use \( i = 1, \ldots, N \) to index the candidate sets and \( j \) to index the candidates within a specific candidate set; as all sets in our data are the same size, \( j \in \{1, \ldots, 20\} \). To use a horse racing analogy, \( i \) can index the races, and \( j \) can index the horses within a specific race. In the application that we will examine, a string of text input from a user will result in a number of output candidates.

2. The candidates in candidate set \( i \), once ranked according to a selected score, such as an empirical loss or cost, are identified as \( X_{i(1)}, \ldots, X_{i(J)} \), where \( X_{i(1)} > X_{i(2)} \) means that, according to their scores, Candidate \( X_{i(1)} \) is preferred to, or ranks more highly than, Candidate \( X_{i(2)} \), and so on. In particular, we are provided, per candidate, with a count for the number of errors, which we will define in the next section. This is provided only in a training context; for the test data, we will use \( \widehat{X}_{i(j)} \) or, for simplicity, \( X_{i(j)} \) to represent the candidate that is predicted to be ranked in position \( j \), for set \( i \). This notation is used for simplicity in ordering the data, and is not used to affect the fitting of the model, i.e. the model will neither know nor utilize the character error count, but our data can be re-ordered for our ease of working with the data in increasing order of the number of errors.

3. We are provided with a cost, or empirical loss, for each of the candidates in a set, denoted as \( \text{Err}_{i,j} \), which refers to the number of typographical errors occurring in the candidate (this value, also known as the \textit{character error rate}, is the number of insertions or deletions necessary to change from a given candidate to the correct result; this will be described further in the next section). This is provided for training purposes, and
will be used in evaluation, but it is not available as a predictor: it is a cost that we must minimize. We cannot predict the number of errors per candidate, but our task will involve a surrogate score for each candidate, such that the ordering by our surrogate score will suffice for being able to select the candidate with the fewest errors.

4. An **oracle candidate** is a candidate with a minimum cost among candidates in its candidate set. Formally, \( j^* \) indexes an oracle if and only if \( \text{Err}_{ij^*} = \min\{\text{Err}_{ij}\} \); note that this definition allows for multiple candidates that are tied for the minimum cost - each can be an oracle candidate. For simplicity, we will sometimes refer to the oracle candidate as an "oracle" or as "oracles" if used in plural.

5. For model-based scoring of individual candidates in a set, we may use \( S_{ij} \) to refer to the score assigned to \( X_{ij} \), and \( F(X_{ij}) \) to refer to the function assigning the score, such as \( S_{ij} = F(X_{ij}) \).

6. We will utilize a surrogate loss function, \( L \), in place of the character error rate \( Err \), and the loss associated with each candidate \( j \) in set \( i \) is denoted \( L_{ij} \).

7. We will use \( k = 1, \ldots, p \) to index the covariates of an observation vector, \( X_{i,j} \). These covariate indices refer to the same set of features, across all candidates and candidate sets.

8. In the problem addressed, the data will be very sparse and each covariate is an integer: an entry is the number of times a feature (e.g. a word) appears in a candidate sentence. For each candidate, the function \( h_k(X_{i,j}) \) is the count of the number of times feature \( k \) appears in candidate \( X_{i,j} \). Most of the time this value is 0.

9. In reranking, which is our context, each set of candidates is the output of a generative system that produces a score that we may denote as \( L_0(X_{i,j}) \).

### 1.2.1 Frameworks for Ranking

A number of approaches for ranking have been developed. The reader is referred to [14] for a fuller treatment of these frameworks. The broader class of problems can be considered as ranking and selection, with particular approaches attempting to optimize one or more of the following approaches:

- Choose the best 1 candidate of \( n \) candidates. For instance, to pick the winner of a horse race. This is often extended trivially to ranking the entire set, by picking one best candidate, then the next best, and so on.

- Choose the best \( k \)-element subset of \( n \) candidates. Applied contexts could include portfolio selection, college admissions, papers selected for a conference, and so on.

- Choose the best 1 candidate of \( k \) of \( n \) candidates. This may seem to be a rare task, but can occur when successive filtering operations are performed, as may be done in search engines and information retrieval. The IME problem is one such example: we could increase \( k \) up to some theoretical maximum \( n \), and the data set we have used
involves a subset of just 20 candidates, among many more that were generated by the generative system.

• Choose the most probable permutation of \( n \) candidates. This has some useful applications, though it is often the case in applied contexts that selection matters more than a full permutation: in many cases, errors in the ranking of the lowest candidates do not matter, as presenting these candidates would be a waste of resources. For instance, in IME, it would be a waste of time for a user to investigate permutations of thousands of potential transliterations.

Often, the techniques are expressed structurally as either pairwise comparisons or as functions applied to permutations. The structural models include both probabilistic models and algorithmic models. A basic framework for such models is pairwise comparison models, known as Bradley-Terry models, where the probability that item \( i \) is preferred to item \( j \), for \( i, j \in \{1, \ldots, N\} \), is given by \( p_{ij} = \frac{v_i}{v_i + v_j} \); the values \( v_i \) may be calculated via a variety of models, such as logistic regression. A more general class of models, for calculating probabilities for permutations of \( k \) items, are known as Plackett-Luce models. In this framework, the probability that item \( i \) is ranked first is given by: \( p(W_1 = i) = \frac{v_i}{v_1 + \cdots + v_k} \). Note that this model may increase the computational complexity, relative to pairwise comparisons, as updating a model could require recalculating each of the values \( v_j \).

In our work, we are essentially using a Bradley-Terry model, though there are many different structural models for pairwise comparison, along with different loss functions, and different optimization algorithms. Some of the most common model families for Bradley-Terry models include: ordinal logistic regression (described in 1.2.1) and ordinal probit regression, support vector machines, neural networks; in addition, general Plackett-Luce methods may invoke many other methods for pairwise or multiclass classification.

**Ordinal logistic regression**

Ordinal logistic regression is a special adaptation of basic logistic regression. We may use a simple binary classification formulation, as in Equation (1.1), where \( x_{i,j} \) may be the difference in a pair of feature vectors, \( v_i \) and \( v_j \), and the outcome is 1 if candidate \( i \) is to be ranked higher than candidate \( j \), i.e. \( \text{score}(i) > \text{score}(j) \), and 0 for the opposite ranking.

\[
\Pr(Y = 1|X = x) = \frac{\exp(\alpha + \beta^T x)}{1 + \exp(\alpha + \beta^T x)}
\]  

(1.1)

We may also describe the relationship, using the logit link, as a linear expression in terms of the predictor variables, as in Equation (1.2).

\[
\logit(p_i) = \log E(Y|X_i) = \log \Pr(Y_i = 1|X_i) = \ln \frac{P_i}{1 - P_i} = \alpha + \beta^T x_i
\]

(1.2)

For precise ranking positions or tiered (stratified) outcomes, we may employ a multiclass formulation, where we let \( Y_i = k \) if \( x_i \) is ranked in position \( k \), \( k \in \{1, \ldots, K\} \), as in Equation (1.3), where the model is specified relative to an arbitrary position, such as the last one,
\( K \), in order to account for the \( K - 1 \) degrees of freedom for the \( K \) probability estimates (i.e. they sum to 1). As above, we can invert the logit function to obtain the exponential formulation as in Equation (1.4). As the sum of the probabilities is 1, the numerator of the final comparison is 1, as shown in Equation (1.5).

\[
\begin{align*}
\log \frac{\Pr(Y = k | X = x)}{\Pr(Y = K | X = x)} &= \alpha_k + \beta_k^T x \\
\Pr(Y = k | X = x) &= \frac{\exp(\alpha_k + \beta_k^T x)}{1 + \sum_{j=1}^{K-1} \alpha_j + \beta_j^T x} \quad \text{for } k \neq K \\
\Pr(Y = K | X = x) &= \frac{1}{1 + \sum_{j=1}^{K-1} \alpha_j + \beta_j^T x}
\end{align*}
\]

In traditional usage, the loss function is the log likelihood, and several methods exist for efficiently searching for an optimal parameter vector, \((\alpha, \beta)\), involving either matrix inversion or coordinate descent. A concise description of several efficient methods for fitting logistic regression models is presented in [10], which includes traditional methods involving the Newton-Raphson algorithm and more recent coordinate descent approaches.

**Other Plackett-Luce variations**

Neural Network formulations have also been developed for ranking, such as RankNet developed by Burges et al. [2]. Support Vector Machines have been explored for paired comparisons, as explored in [11],[12], and [19]. Other Plackett-Luce variations include LogLoss and Maximum Entropy models, explored in NLP applications in [3] and [5].
1.3 Introduction to the IME Application and Data

In order to sell software for global use, Microsoft depends on an Input Method Editor (IME) to provide a standard method for inputting text in different languages. The primary job of the IME is text transliteration, which is conversion of input phonetic strings into the corresponding text using the characters of the user’s target language, which is called the ideographic representation. Because this process converts a phonetic representation to a text representation, it is much like speech recognition, and the IME, like many speech recognition systems, uses a version of a Hidden Markov Model (HMM) to generate and score a number of candidate sentences.

Our interest is to extend an IME system, which involves a generative system and a scorer, by developing a reranker; the interaction between each step in the IME process is depicted in Figure 1.1. The generative system generates a set of candidates, based on matches of the phonetic strings with a phonetic “dictionary” of words: sub-strings are matched to words that have the same pronunciation, and different choices of matches produce different candidate sentences. The scoring system uses a Markov Model to score each candidate sentence, based on the observed frequencies of words, and transitions between words, in a large text corpus. At this point, one could propose to the user that the best candidate is the one with the highest score from the Markov Model; however, we aim to improve upon this by using various candidate features to rerank the set of candidates and potentially find a better candidate among the set of candidates produced by the generative system. The inputs to the reranker are the features of the generated candidates and the output of the scorer; the output is a new ordering of the candidates.

Finally, in order to generate large amounts of training text, which could be costly to manually input, a useful trick is that a given Japanese sentence can be automatically transformed into its phonetic representation using information from a dictionary. Microsoft researchers used this insight to reverse-transliterate a large amount of text from the Nikkei Shim bun newspaper, a leading business-related newspaper in Japan. In this way, they knew the correct output and could give a matching phonetic sequence. The data set for our research was provided by Jianfeng Gao and Hisami Suzuki, of Microsoft Research, and consists of sparse feature vectors for 80,000 input sentences, with 20 candidates per sentence, for a total of 1.6 million rows of data.

In the next section, we will give an example of usage of the IME and the creation of training data for our problem.

1.3.1 Example

In this section, we will demonstrate the usage of the IME, from the perspective of creating training data. Recall that our training data are generated in a synthetic manner: a large amount of newspaper text was assumed as the target; to generate synthetic user input, the target text was “pumped backwards” through a system that converted the text into a phonetic representation, and then to the Latin alphabet representation that a user would have entered in order to obtain the target text. That representation is synthetic: it was utilized as if it was from a user, and the IME was tasked to generate Japanese text candidates. Our training set begins with the candidates generated based on that synthetic input, and assessed against
the true text from the newspaper corpus. In this section, we will take one sentence and “pump backwards” to generate the Latin input text, and then demonstrate the candidates generated by the IME.

For the input text, suppose that a user intends to enter the text in Figure 1.2, which, translated from Japanese, means “DuPont fell on profit taking.” (i.e. the stock price of DuPont dropped, presumably after a run-up in its price over a recent time period). A minor note for this example is that all three of the different character sets in Japanese are in use\(^1\). This is a reasonable headline for a news article about DuPont’s stock.

First, we demonstrate how we can “pump backward” from this text and generate a phonetic representation - a synthetic version of what a user would be required to enter. We have transliterated the reference text, in Figure 1.2, into the basic syllabary (syllable-based character set), Hiragana, in Figure 1.3, which converts each of the Katakana and Kanji characters to their corresponding forms in Hiragana. As the Katakana and Hiragana syllabaries have a 1:1 relationship, this conversion can be done without error. The Kanji-to-Hiragana transformation requires referencing a phonetic dictionary, to find the correct pronunciation of a word. Some ambiguity and errors may arise at this point, as most Kanji characters could have multiple pronunciations, and a sequence of characters has the potential for being split into one or more words. For the IME data set, if multiple dictionary matches were found, with different lengths, then the longest one was used\(^2\).

The next issue to address is to map from the Hiragana representation to the keyboard characters that a user would press. In particular, we are converting from the Hiragana representation that has been identified to a corresponding Romaji (Roman character, i.e. Latin alphabet) transliteration. This can be done without error, as Hiragana is a phonetic syllabary with a standard transliteration into the Latin alphabet. An example of this Hiragana and Latin alphabet transliteration is shown in Figure 1.3.

\(^1\)The first 4 characters are from Katakana, which is typically used to represent words from foreign languages; the 6th, 7th, 9th, and 12th characters are from Kanji, which is derived from Chinese characters; all remaining characters are from Hiragana. These are the three most common sets of characters in the raw data provided; there are three other sets that also appear: numbers, Roman letters (e.g. “CEO”), and non-verbal characters, such as punctuation.

\(^2\)It is possible for errors to arise in this matching, but, from our inspection of related data and in our experience with IME systems, such errors appear to be uncommon.
Figure 1.2: Example sentence, in Japanese. Kana (character sets) used: Kanji (Chinese characters, complex characters), Katakana (simple characters, with straighter strokes used for foreign words), and Hiragana (simple characters with more curved strokes). Translation: “DuPont fell on profit taking.”

でゅぼんはりくいりでさげた。
dyupon wa riguiuri de sageta.

Figure 1.3: Phonetic representation of the example sentence, in Japanese Hiragana, with Romaji (Latin alphabet transliteration) below.

Next, supplied with synthetic Romaji text strings, we may use the IME system to produce candidate transliterations. At this point, a generative system produces multiple candidates. An example of 20 such candidates, using the Figure 1.3 input, is presented in Figure 1.4. In this example most of the variation between candidates occurs in the 5th position. In fact, the first result is the correct transliteration. The other examples vary in the selection of a syntactic particle, which is an interesting orthographic problem: the character used is pronounced “wa” but is transliterated as “ha”, and most of the incorrect transliterations use a character pronounced as “ha”. In this case, the training corpus frequency of the correct character is sufficiently high that it is still chosen correctly as the top candidate transliteration.

Figure 1.4: Example of 20 proposed transliterations (varying “ha” versus “wa”)

Finally, in this example, were a user selecting from these candidates, they would select the first candidate. In our data, all candidates, including the first, are assessed for the number of character differences with the target text. In this case, the first candidate has
zero errors, while others would require at least 2 edits: 1 to delete an erroneous character, and 1 to insert the correct character. Some would require more than 2 edits. If the correct candidate was not proposed, then the best candidate in the set would have at least 2 errors, and this happens in many of the candidate sets in our data. Moreover, regardless of whether or not the correct candidate is in the set, we identify the candidate with the fewest errors as the \textit{oracle candidate}, i.e. the candidate an omniscient oracle would pick, if it could choose among the output of the generative system.
1.4 Overview of Reranking Problem

1.4.1 Data Set for Reranking

Before stating our reranking problem, we must first define the data that we will use. In this sub-section, we will describe the data. The modeling objective is described in Section 1.4.2, and additional challenges are addressed in Section 1.4.3.

Each transliteration “task” involves a set of candidates produced by the generative system; the aggregate of these sets will be referred to as candidate sets. We enumerate the candidate sets as \( \{ S_i \in \mathbb{R}^{J_i \times P} \} \), \( i = 1, \ldots, N \), with \( J_i \) candidates per set. Thus, we can denote the union of the sets of candidates as \( \{ X_{ij} \in \mathbb{R}^P \} \), \( i \in \{1, \ldots, N\}, j \in \{1, \ldots, J_i\} \). In our data, there are \( N = 80,000 \) candidate sets, and \( J_i \) is a constant, which we can denote simply as \( J \): there are 20 candidates in every candidate set.

In order to build a model, we need both predictors and responses. The predictor inputs for each candidate are a combination of a baseline score, \( L_0(X_{ij}) \), and a feature vector, \( X_{ij} \). The baseline score is produced by the Markov Model that follows the generative system. The feature vector, \( X_{ij} \), is represented as a vector of counts corresponding to the features that appear in the candidate sentence. These features are words and two-word sequences, known as bigrams; due to the dimensionality of the vocabulary, the vector is mostly 0s, thus is very sparse. (On the computer, we store only the non-zero entries.) The indices for these features are determined a priori by referencing a very large body of text.

In our case, the response data present a few additional wrinkles. Unfortunately, we were not given the correct transliterations, but a vector of error measurements, \( \{ Err_{ij} \} \), for each candidate in each set. The error value, \( Err_{ij} \), is the number of characters that differ between the correct transliteration and a given candidate, \( X_{ij} \). It is important to note that the generative system, described in Section 1.3, which precedes our reranker, is not guaranteed to produce a candidate set that includes the correct transliteration. Instead, we introduce the notion of an oracle candidate as the candidate with minimal error, where \( Err^*_i = \min_{j \in \{1, \ldots, J\}} Err_{ij} \), and we are not guaranteed that \( Err^*_i = 0 \). Moreover, there may be multiple oracle candidates, i.e. \( \#\{ X_{ik} | Err_{ik} \leq Err_{ij}, \forall j \in \{1, \ldots, J\} \} \geq 1 \). Occasionally, all of the candidates are equally bad, from the character error perspective.

1.4.2 Functions for Ranking and Reranking

Prior to defining our reranking task, we will define scoring and ranking function, \( f_{\text{score}} \) and \( f_{\text{rank}} \). A scoring function maps a vector to real values, i.e. \( f_{\text{score}}(X) \rightarrow Y \), where \( X \in \mathbb{R}^P, Y \in \mathbb{R} \). A ranking function maps from the score to an ordering, or ranking, e.g. \( f_{\text{rank}}(Y_j) \rightarrow r(j) \), where \( j = 1, \ldots, J, Y_j \in \mathbb{R} \), and \( 1 \leq \ldots \leq r(j)-1 \leq r(j) \leq r(j)+1 \leq \ldots \leq J \), where \( r(j) \) corresponds to the rank assigned to \( Y_j \). The standard mapping function in this case is to map to order statistics, where \( Y_{(1)} \leq Y_{(2)} \leq \ldots \leq Y_{(J)} \) and \( r(j) = \arg \min k \) such that \( Y_{(k)} = Y_j \). An important terminological issue is that in ranking literature, given a standard reliance upon order statistics for ordering or ranking the scores, it is quite common for the terms “ranking” and “ranking function” to be applied to either the scoring function or the combination of both the scoring and ranking functions. When we use the term, we refer to the combination, as we also assume the above ordering.
Our reranking task is possibly more precisely defined as “re-scoring”: we aim to find an optimal scoring function, $f^*$, in a family of functions, $\mathcal{F}$, for the identification of the oracle candidate within each set, using both the score previously assigned, i.e. $L_0(X_{i,j})$ and the predictor features $X_{i,j}$; more precisely, let:

\[
X_{i*} = \arg \min_{j \in \{1,...,J\}} f(X_{ij}, L_0(X_{i,j})), \text{ where } \tag{1.6}
\]

\[
f^* = \arg \min_{f \in \mathcal{F}} \sum_{i=1}^{N} Err(X_{i*})
\]

Our research goal, then, is to specify $\mathcal{F}$ and a method for finding an optimal member of $\mathcal{F}$ that minimizes the errors, as stated above. Because of the size of the data set, we will face additional computational constraints.

### 1.4.3 Additional challenges

As addressed in Section 1.1, our work on an algorithmic model for reranking requires a structural model (i.e. a family of models), a loss function, an optimization or search method, and methods for processing the data. The last two issues present several challenges that we can now clarify with examples, but will return to later, to address empirically.

Our interest in this particular application involves careful consideration of the interaction between a large, yet sparse, data set and the goal of finding a computationally feasible, sparse model. We will address model sparsity later, but we need to understand how sparsity and growing dimensionality occur in our data. Sparsity in our data is due to the fact that our data arise from sentences (or candidate sentences), and every sentence uses only a very small fraction of the words available in the vocabulary. Growing dimensionality is an issue related to both sampling and to the properties of language growth, as new words are added to the language.

In natural language processing (NLP) contexts, such as ours, one commonly finds that as more data are acquired the dimensionality increases. This growth in dimensionality, or novel features, is obvious when we realize that new words are added to the language in an ongoing basis. The most notable example in recent years of an explosion of popularity of novel word sequences are those derived from the name “Barack Obama”. Because of both the preceding rarity of both his given name and his surname, and the frequency with which his name appears in the news, a large number of new bigrams have appeared in contemporary news that did not appear a few years ago, nor could they, as he attained several new positions very quickly. For instance, a list of a few of the most common bigrams involving his name could include those in Table 1.1, below. As this example illustrates, the addition of just one or two new very frequent words can have a tremendous impact on the number of new bigrams that occur. Moreover, new words are continuously added in every language and many specialized fields. In fields involving ever-changing fashion or technology, such words gain greater prominence and diffusion, with notable recent examples deriving from websites, such as Twitter and Facebook, or from popular consumer technology. This propensity to add new words to the language, especially among computer users engaged
in extensive communication has important implications for our problem, as we will revisit later.

<table>
<thead>
<tr>
<th>Barack Obama</th>
<th>Senator Obama</th>
<th>President Obama</th>
<th>Michelle Obama</th>
<th>for Obama</th>
<th>Obama said</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Obama</td>
<td>Obama campaign</td>
<td>Obama advisor</td>
<td>Obama spokesman</td>
<td>or Obama</td>
<td>Obama or</td>
</tr>
<tr>
<td>Nominated Obama</td>
<td>said Obama</td>
<td>Obama for</td>
<td>Obama asked</td>
<td>President Barack</td>
<td>Senator Barack</td>
</tr>
</tbody>
</table>

Table 1.1: Example of increasing dimensionality in vocabulary, using bigrams involving Barack Obama.
1.5 Evaluation

Just as there are many potential loss functions in regression and classification, there are numerous methods for evaluating ranking functions. We will first address evaluations on test data, and, in later sections, introduce surrogate loss functions for use in training models. We will list the evaluation criteria that may be desirable, and identify those that are both feasible and desirable.

For the IME context, in addition to calculating mis-orderings (e.g. assessing rank correlation), several additional evaluation options arise: (1) the proportion of times the oracle selection has been picked, (2) the total character error rate, (3) the word error rate, and (4) the total number of errors. Character errors, the second option, are related to a score known as the edit distance, or Levenshtein distance, which is the number of insertions and deletions necessary in order to transform one string to another. As our system depends on a generative system, we may also compare performance on any metric to that achieved by an oracle.

In contrast to Gao et al. [8], we are unable to report the character error rate, as we do not have access to the true reference transliterations, so we will instead reference the total character errors or the average number of character errors per sentence. As these evaluations are based on large corpora, there should be a very close relationship between the character error rate and the total number of character errors, namely the second and fourth additional criteria should be strongly correlated. In addition to reporting the mean errors per selected candidate, we will also report on Oracle precision: the proportion of the test results in which an oracle was correctly identified as the best, i.e. our first additional scoring criterion.
1.6 Methodology

As discussed in 1.1, Hand et al. [9] identified that a machine learning implementation requires several components, including: a structural model, a loss function, and an optimization or search algorithm. We address the basic structural models in Sections 1.6.1 and 1.6.2, loss functions in 1.6.3, and an iterative method known as boosting in 1.6.4. We address efficient data structures and algorithm design in Section 1.6.5. In order to address complexity of the model and avoid overfitting, we review the Lasso regularization method, by Tibshirani [17], in 1.6.6 and a search algorithm known as Boosted Lasso, by Zhao and Yu [20], in 1.6.7. Other approaches developed for ranking models are discussed in Section 1.2.1.

1.6.1 Linear Scoring

As stated in Section 1.4, we wish to find a scoring function that minimizes errors when choosing the best scoring candidate, as in Equation (1.6).

In many scoring systems, the underlying score is derived as a linear combination of features. This has been the case for Natural Language Processing (NLP) problems involving ranking of transliteration candidates [8], machine translation [16], parsing [5], part of speech tagging [4], and others.

For a linear function, for a given coefficient vector of length, $\alpha \in \mathbb{R}^{p+1}$, our scoring function is:

$$F(X_{i,j}; \alpha) = \alpha_0 L_0(X_{i,j}) + \sum_{k=1}^{p} \alpha_k h_k(X_{i,j})$$

(1.7)

1.6.2 Paired Comparisons and Margin-based approach

We extend our model by embedding the linear score in a paired comparison framework. Our work is based on methods introduced by Collins in [5], which builds on the paired comparison approach identified in Section 1.6.2. We base our comparisons on the difference between the scores assigned to an oracle candidate and each of the other candidates from the same set. This difference is the margin between the two (Equation (1.8)), given a particular coefficient vector, $\alpha$.

$$M(i,j; \alpha) = F(x_{i,1}; \alpha) - F(x_{i,j}; \alpha)$$

(1.8)

The margin allows us to formulate the problem as a pairwise comparison between the oracle and every other candidate.

1.6.3 Loss function

Our next component is a loss function and a method for finding the best member of the linear family, $\mathcal{F}$. The primary evaluation metric of the number of character errors associated with each choice is computationally infeasible as an optimization criterion, as it is discrete, non-differentiable, and could require us to undertake a grid search across all $P$ dimensions and many of the $N \times J$ candidates (recall that $N$ is the number of candidate sets, and $J$ is
the number of candidates per set, which is 20 for all of the sets in our data). Instead, for efficiency purposes, we need to employ a surrogate loss function that is both computationally tractable and associates well with our original loss function.

In many traditional ranking problems, logistic regression is often used, but this is not amenable to very high dimensional problems, due to the computational complexity involved in fitting or applying a model to a candidate set. The computational complexity of fitting and applying models is a challenge for Markov Random Fields [5] and other methods that require access to most of the data in any one iteration in order to update likelihood estimates. Instead, we turn to a surrogate loss function which is amenable to iterative methods. We will address an iterative method in the next chapter, called Boosting. For now, we can consider exponentiating the margin as:

\[
ExpLoss(\beta) = \sum_{i=1}^{N} \sum_{j=2}^{20} e^{-M_{i,j}(\beta)}
\]

This is a standard loss function for boosting, and is a useful surrogate loss function for ranking: there are no misranking errors if the margins are positive, and the inequality \(\exp x \geq 1_{x \leq 0}\) implies that \(\sum_i \sum_j \exp^{-M_{i,j}(\beta)} \geq \sum_i \sum_j 1_{M_{i,j} < 0}\), so that \(ExpLoss\) becomes a useful upper-bound for our desired empirical misranking loss.

1.6.4 Boosting

Boosting [7] is sometimes described in the computer science literature as an ensemble of "weak" learners, or as a method of iteratively reweighting data in order to refit models based on the errors of the previous iteration. Friedman, Hastie and Tibshirani [10] recharacterized the approach as a forward stagewise additive model: at the \(m\)-th iteration, choose:

\[
Update(\beta, k, \delta) = \{\beta_0, \beta_1, \ldots, \beta_k + \delta, \ldots, \beta_p\}
\]

\[
(k^*, \delta^*)^{(m)} = \arg\min_{k, \delta} Loss(Update(\beta, k, \delta))
\]

\[
\beta^{(m)} = Update(\beta^{(m-1)}, k^*, \delta^*)
\]

\[
F_m(x; \beta^{(m)}) = F_{m-1}(x; \beta^{(m-1)}) + \delta^* h_{k^*}(x)
\]

\[
M_{i,j}(Update(\beta, k, \delta)) = M_{i,j}(\beta) + \delta [h_{k}(x_{i,1}) - h_{k}(x_{i,j})]
\]

\[
ExpLoss(Update(\beta, k, \delta)) = \sum_{i=1}^{N_i} \sum_{j=2}^{N_j} \exp(-M_{i,j}(\beta) + \delta [h_{k}(x_{i,1}) - h_{k}(x_{i,j})])
\]

Collins [5] applied boosting to reranking of parsing candidates, motivated by Freund et al. [7].

1.6.5 Efficient data structures and algorithm

Due to the scale of the data set, we utilize a number of additional data structures, namely, \(A^+, A^-, A^0, W^+, W^-, B^+,\) and \(B^-\), as presented in Table 1.2, in the notation of Collins [5].
Most of the additions are index sets that can be created for more efficient computation of the boosting updates.

\[
A_k^+ = \{ (i, j) : [h_k(x_{i,1}) - h_k(x_{i,j})] = 1 \} \\
A_k^- = \{ (i, j) : [h_k(x_{i,1}) - h_k(x_{i,j})] = -1 \} \\
A_k^0 = \{ (i, j) : [h_k(x_{i,1}) - h_k(x_{i,j})] = 0 \}
\]

\[
W_k^+ = \sum_{(i, j) \in A_k^+} e^{-M_{i,j}(a)} \\
W_k^- = \sum_{(i, j) \in A_k^-} e^{-M_{i,j}(a)}
\]

\[
B_{i,j}^+ = \{ k : (h_k(x_{i,1}) - h_k(x_{i,j})) = 1 \} \\
B_{i,j}^- = \{ k : (h_k(x_{i,1}) - h_k(x_{i,j})) = -1 \}
\]

Table 1.2: Notation of index sets

For boosting, we followed Collins’ “BoostLoss” algorithm, in Algorithm 1.
Algorithm 1 ExpLoss Algorithm (“BoostLoss” algorithm from Collins [5])

Initialize

- $\hat{\alpha}^0 = \{\alpha_0, 0, \ldots, 0\}$
- Set $M_{i,j} = \alpha_0[L_0(x_i, 1) - L_0(x_i, j)]$
- Calculate $W_k^+, W_k^-, BestWt(k)$ and $BestLoss(k)$ as:
  - Let all $W_k^+, W_k^-, BestWt^*(k)$ and $BestLoss^*(k) = 0$
  - For $(i, j) \in A_k^+$, $W_k^+ = W_k^+ + e^{-M_{i,j}}$, similarly for $W_k^-$
  - $BestWt(k) = \frac{1}{2} \log \frac{W_k^+}{W_k^-}$
  - $BestLoss(k) = 2 \sqrt{W_k^+ W_k^-} - W_k^- - W_k^+$

Repeat

- For $t = 1$ to $T$
  - Let $k^* = \arg \min_k BestLoss^*(k)$
    $\delta^* = BestWt^*(k^*)$
  - For $(i, j) \in A_k^+$:
    * $\Delta = e^{-M_{i,j} - \delta^*} - e^{-M_{i,j}}$
    * $M_{i,j} = M_{i,j} + \delta^*$
    * for $k \in B_{i,j}^+$, $W_k^+ = W_k^+ + \Delta$
    * for $k \in B_{i,j}^-$, $W_k^- = W_k^- + \Delta$
  - For $(i, j) \in A_k^-$:
    * $\Delta = e^{-M_{i,j} + \delta^*} - e^{-M_{i,j}}$
    * $M_{i,j} = M_{i,j} - \delta^*$
    * for $k \in B_{i,j}^+$, $W_k^+ = W_k^+ + \Delta$
    * for $k \in B_{i,j}^-$, $W_k^- = W_k^- + \Delta$
  - For all features $k$ whose $W_k^+$ or $W_k^-$ changed during this iteration, recalculate $BestWt^*(k)$ and $BestLoss^*(k)$.
  - Let $\hat{\alpha}^t = \text{Upd}(\hat{\alpha}^{t-1}, k^*, \delta^*)$

Output $\hat{\alpha}^T$

1.6.6 Regularization: Lasso and Boosted Lasso

Several concerns arise in estimating model parameters in a very high dimensional context. In maximum margin learning, the first concern is overfitting - as the margin increases, our training error decreases, but we will eventually see test error increasing. One way to address this is to add an additional objective function: while we seek an optimal solution in $\mathcal{F}$, we may add an additional constraint to the number of “active” features in the model. As the data for the IME problem has 500,000 features, it is unwise to find a model that has 500,000.
non-zero weights.

One method that has become quite popular for regularization is the Lasso [17]. The Lasso penalizes for the L1 norm of the coefficient vector, $\beta$, as: $\text{LassoLoss}(x; \beta, \lambda) = \text{ExpLoss}(\beta) + \lambda \sum_{k=1}^{P} |\beta_k|$. As has been explored by Donoho [6] and others, the Lasso is a surrogate for the L0 loss, which is non-convex. Furthermore, the Lasso can be implemented in conjunction with Boosting in the method known as Boosted Lasso, as developed by Zhao and Yu [20].

It is notable that boosting and forward stagewise fitting yield parsimony only by limiting the number of iterations. This parsimony can approximate the Lasso solution, but a backward step, as Zhao and Yu [20] suggested, can implement a closer approximation to the Lasso method, by incorporating the Lasso penalty at every step.

In addition, we investigated other variations on boosting, in order to find sparse solutions; these methods include: early stopping, bagging [1] and subset overlap [derived independently, but related to Stability Selection, as described by Meinschahen and Bühlmann [15]].

1.6.7 BLasso algorithm

The Boosted Lasso (BLasso) is outlined below, in Algorithm 2, with the forward step adapted from the ExpLoss Algorithm (1): instead of choosing an optimal weight, the forward step size was fixed. As with the ExpLoss Algorithm, we make use of the index sets, $A$, $B$, and $W$, previously described in Table 1.2; their use in efficiently updating the loss values remains the same for both the forward and backward steps.

Algorithm 2 BLasso Algorithm

1. Initialize:

   $\beta^{(0)} = \arg \min_{\beta; \beta_k = 0, k \geq 1} \text{ExpLoss}(\beta_0)$

   i.e. $\beta^0 = (\beta^*_0, 0, \ldots, 0)$

2. Take a forward step:

   $$(k^*, \delta^*) = \min_{1 \leq k \leq P, \delta = \pm \epsilon} \text{ExpLoss}(\text{Upd}(\beta, k, \delta))$$

   $$\beta^{(t)} = \text{Upd}(\beta^{(t-1)}, k^*, \delta^*)$$

3. Calculate

   $$\lambda = \frac{\text{ExpLoss}(\beta^{(t-1)}) - \text{ExpLoss}(\beta^{(t)})}{\epsilon}$$

4. For the backward step: Calculate $\text{LassoLoss}(\beta) = \text{ExpLoss}(\beta) + \lambda \cdot ||\beta||_1$. If the difference for any feature $k \in \{1, \ldots, P\}$ is greater than a specified tolerance value, let $\beta_k = \beta_k - \epsilon \cdot \text{sign}(\beta_k)$ for the feature $k^*$ leading to the greatest reduction in the lasso loss.
1.7 Data Analysis

1.7.1 Data Summary

At a preliminary level, the IME data set is reasonably large: there are \( N = 80000 \) candidate sets, with \( J = 20 \) candidates per set, for a total of 1.6 million rows. There are also almost \( P = 500000 \) dimensions, or “features”, arising from 68,000 words and 407,000 two-word sequences, known as bigrams.

The matrix is very sparse: the density of the data matrix\(^3\) is \( 3.5 \times 10^{-5} \). There are 27 million entries, though only 25 million are “discriminative”, in that \( h_k(x_{i,j}) \) is not constant across all candidates, \( j = 1, \ldots, 20 \), in the \( i \)-th set, for feature \( k \).

Most candidates are short, while a fraction of candidate sets have candidates that are exceptionally long. Per-candidate, the median sentence length is 9 words, and the median number of total features is \( 15^4 \). The mean number of features per candidate is 57 features. Figure 1.5 shows a histogram of sentence lengths across all candidates.

As mentioned previously, the possibility exists for multiple oracle candidates per set. In fact, the mean number of “oracles”, i.e. oracle candidates, per candidate set is 2.6, and the median is 2. A histogram of the number of oracles per set is in Figure 1.6.

\[ \text{Figure 1.5: Histogram of sentence lengths} \]

Finally, in Section 1.4.1, we noted a concern that the generative system does not necessarily produce the correct transliteration among the candidates. As a result, perfect performance in candidate selection is unobtainable - we can only do as well as an oracle, and we can examine how well an oracle would by assessing the number of errors per oracle. Figure 1.7 shows a histogram of the errors per oracle candidate.

\(^3\)Matrix density, for a sparse matrix, refers to the proportion of entries that are non-zero.

\(^4\)Recall that the feature space includes both words and bigrams, so a 9 word sentence should have 17. The observed number of bigrams is slightly lower than what could be observed due to some bigrams being sufficiently rare that they did not occur in the reference corpus used for indexing all features.
1.7.2 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is an important part of any applied statistics project. However, examination of a data set involving 500,000 dimensions challenges any modern data visualization capability. Methods such as parallel coordinate plots decrease in interpretability after 10-20 dimensions; even establishing interpretability for such high dimensionality is feasible only with auxiliary techniques such as shading and coloring, rearranging of correlated variables, and other methods that have been summarized in a variation called “Textile Plots” [13].

At present, there does not appear to be a solution for visualizing thousands of dimensions simultaneously. Instead, we must pragmatically realize that our investigations must focus less on the data than on interactions between the data and iteratively-fitted models. In this context, we delve into assumptions about the data and the structure of the model. In this sense, we may look at using different components of Hand et al.’s [9] machine learning framework in order to examine the data; we address such visualization concepts further in Chapter 2, as such a problem is beyond the scope of this chapter. Chapter 2 presents a method for a guided tour of the data using insights from the model structure, the loss function, and the search or optimization algorithm, in order to understand how the model fits or errs with different subsets of the data.
1.7.3 Noise

Exploratory analysis of the data set revealed a number of anomalies. Prior experience with natural language processing and handling text derived from webpages instilled in the author some concern that the data should be explored carefully to assess whether it truly reflects natural language data or if there could be problems arising during preparation of the data set. Unfortunately, due to copyright restrictions, the original text was unavailable in its entirety. However, we have observed a few anomalies that cause concern about the preparation of the data and the generalizability of insights for this data set.

Below, we describe several types of noise or anomalies in more detail, such as pre-processing noise, erroneous oracle candidates, multiple oracle candidates (“oracles”), and orthographic and linguistic anomalies. In total, we classified as much as 30% of the data as noisy, using the descriptions of noise in the sub-sections below. In order to explore the effects of such noise and anomalies, we developed models on the original data and on screened subsets; our results are addressed in Section 3.9. Due to important aspects of the noise and some properties of our model building algorithms, it appears that the performance of the models were not much affected by the noise screening, though the dimensionality of the data and the models were reduced and computational efficiency was increased. In short, the models appeared to be robust in the presence of such noise, but removing the noisy candidate sets allowed for much faster computation.

Pre-processing noise

We define as pre-processing noise that noise that arises either in the preparation of the raw text data or as preparation for the reranker. Such candidate sets exhibit elementary and unusual statistical properties that make them candidates for review before reranking is attempted. Two such concerns are:

- Repetition: There are some candidate sentences with exceptional levels of repetition (when this extreme repetition occurs, it has been observed in all of the candidates in a candidate set). The extreme is a candidate that has the same word repeated 38 times out of 47 total words. This does not generally occur in meaningful human language, but can frequently arise due to processing errors when dealing with web text. Although we cannot confirm this with the original text, in other projects we have seen such behavior when a program improperly handled HTML tags, which could cause the appearance of “non-words” such as {html, jpg, br, p, css, div, href, a, b} to appear in the vocabulary.

- Sentence length: some candidate sentences are up to 74 words long. While not inconceivable, such length is unusual. In the author’s experience, this can occur due to incorrect identification of sentence boundaries, causing sentences to run together. Furthermore, due to the sequential nature of the input stream, an error at one character may be propagated for many characters.
“Bad Oracles”

A significant concern is that 18% of the candidate sets have oracle candidates with five or more errors, and over 5% of the training set has oracle candidates with 10 or more character errors. In practice, the user would encounter the errors as these arise from the generative system, rather than after the entire sentence is produced. If a user were to encounter 10 errors in a sentence at least 5% of the time, their frustration level could be quite high.

From a practical perspective, to compare an oracle with 10 character errors to another candidate with 11 errors may yield relatively little value, as we are starting with suggestions that are quite far from the correct transliteration. In recognition of both the real-world usage and the effect on the overall loss caused by such noisy data, we may wish to downweight such samples in training. In our work, we modeled data with and without candidate sets with such “bad oracles”, to explore how the models changed.

Oracle multiplicity

The majority of candidate sets had 2 or more oracle candidates. This is a stimulating problem, in that we cannot assure that one is practically better than another, except in that they have an equal number of character errors. This is a flaw in the paired comparisons methodology. Pragmatically, we followed the example of prior researchers [8], and simply chose one candidate as an oracle, breaking ties by using the baseline score (the “LogProb” score). While this is practically simple, there remains the possibility that, of 2 equally good oracle candidates, a preponderance of other candidates in the set will be more similar to one of the oracle candidates than to the other. In that case, we may have an indication about the nature of the errors produced by the generative system. We investigated this to a limited extent, but came to the same recurring issue: without access to the original Japanese text, it is occasionally impossible to separate issues of improving ranking versus fixing bugs in preprocessing methods.

Atypical language and characters

In addition to noting various pre-processing or filtering anomalies in the primary data set, we also examined samples of text released by Microsoft, and found that there were at least 3 sub-types of text in the corpus: news text, headlines, and photograph captions. In standard computer usage, most users usually write in a style most similar to the body of news articles than the other two forms, which do not tend to be sentences. Another anomaly is that variation among candidate sentences is sometimes due to different selections of non-linguistic characters, such as variations among several types of quotation marks. In the sample corpus, we observed instances where the variation among top candidates was not found in the words used, but in various combinations of beginning and ending typographic symbols used for quotations. Wikipedia [18] lists 12 such symbols, so it is possible that quite many variations could be produced before the generative system produces alternative word choices. We are unable to address these variations in language sub-types and orthographic permutations, but it should be addressed in future work, both as an aim to improve pre-processing and as extension to developing specialized models for ensemble methods of ranking.
Noise summary

In conclusion, we found that as much as 20% of the data set may be corrupted by easily identifiable noise; such candidates are characterized by one or more of the following properties: excessive redundancy, errors, length, and errors relative to length (i.e. short candidates with a large number of errors). Furthermore, we suspect that more of the data may be anomalous in that it is a mixture of language types (e.g. articles and photograph captions) or involves excessive orthographic-based variation. Because existing papers on this data set do not exclude such noise, we report results on the full data set, but will also note that there are significant potential benefits to model accuracy and efficiency if we exclude such noise. In practice, we cannot push the noise issue beyond the limit of what is identifiable without access to the original text.

1.7.4 Length Stratification

Two hypotheses motivated further exploratory data analysis (EDA) and modeling work related to partitioning the data by sentence length:

- A transliteration error by the generative system might affect whether subsequent words are transliterated correctly, at least up to punctuation, if not all the way to the end of the sentence.

- Very short sentences may involve fewer rare words or complex constructions, which could have sufficient frequency in general use to allow for complete or nearly complete matching against prior observations in the corpora. The most extreme example would be single word sentences: there may be 2 or more potential transliterations, but there would be no data within the sentence to motivate any choice but the most common form. As a result, for very short sentences, there may either be very accurate transliterations or little room for improvement in reranking, because it is unlikely that any information could be extracted to supplant the more popular of two potential transliterations.

An example of transliteration errors that may propagate is presented in Figure 1.8. The three lines presented are: the original text, the phonetic representation, with space around the troublesome phoneme, and a candidate transliteration. The first four characters would be entered as: “Mata ei chii”; the independent character is “chii”, and should merge with the subsequent phoneme, “i”. However, the candidate transliteration merged the “chii” with the preceding 2 syllables, “ei”, to produce “eichii”, which happens to be a Japanese pronunciation for the English character “li”. In this case, there are three characters affected in the local area, causing an error rate of at least 6 characters for the sentence (3 deletions and 3 insertions would be necessary for replacement).

Figure 1.8: “H” example
Figure 1.9 illustrates the relationship between the number of character errors in oracle candidates and the sentence length of those candidates. Until candidate sentences reach about 30 words, the mean number of errors remains fairly constant (in fact, the proportion of oracle candidates with errors declines slightly until about 30 words), and then a large number of errors becomes very common.

![Figure 1.9: Mean errors per oracle candidate versus sentence length](image)

### 1.7.5 Dimensionality: \( P \) grows with \( N \)

In order to explore the growth of the dimensionality, i.e. the number of features observed, with the number of candidate sets used for training, we assessed the dimensionality for an average of 1000 bootstrap samples, without replacement, of increasing numbers of candidate sets. The results, presented in Figure 1.10, indicate two issues: (1) we should expect that the number of total number of unique features or dimensions observed will increase with new candidate sets, and (2) any subsample will very likely be a subspace of the dimensionality of the full data set available. From these two issues, we realize that by splitting the data into a training set and test set, each subset will very likely have features that are not present in the complementary set. If those features are useful for accurate prediction, say, for the test set, we face a challenge in that those features may not appear in any observations in the training set.

![Figure 1.10: Number of Active dimensions versus sample size, via bootstrap samples.](image)
1.7.6 Conclusions

There are numerous issues that arose in the preparation of the data, as evidenced from the exploratory analysis. First and foremost is the presence of various kinds of non-linguistic noise. Sentences that are exceptionally long and repetitive simply do not occur in human language, but can occur in automatically prepared corpora. Without access to the original Japanese text, we need to proceed carefully in terms of exclusion of likely noise and in generalizing the results of modeling to novel application domains, as the question arises of whether or not the modeling is improving ranking in general or fixing preprocessing errors.

In addition, we are faced with challenges such as a multiplicity of oracle choices, so-called “bad oracles” - candidate sets where even the oracle is very unacceptable to a human user, and the growth of the dimensionality, $P$, with the number of observations, $N$. The first two challenges are domain-specific and require domain expertise in order to give guidance; the latter issue, of $P$ growing with $N$, is an area of active research throughout much of the statistical machine learning community, and we hope to offer guidance from a practical application.
1.8 Results

1.8.1 Overview

In Table 1.3 we have presented the performance of both Boosting and Boosted Lasso, relative to the baseline, i.e. the “LogProb” score that is the output of the baseline ranking method for IME. Boosting offered a sizable reduction in the error rate across multiple measures. Boosted Lasso further reduces the error rate, but the improvements are relatively small across several metrics.

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean errors per selected candidate (Errors relative to Oracle)</th>
<th>Oracle Precision</th>
<th>Iterations</th>
<th>Error reduction relative to baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogProb (baseline)</td>
<td>3.23 (0.87)</td>
<td>0.50</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Boosting</td>
<td>2.93 (0.57)</td>
<td>0.64</td>
<td>300</td>
<td>9.9%</td>
</tr>
<tr>
<td>Boosted Lasso</td>
<td>2.92 (0.57)</td>
<td>0.65</td>
<td>5000</td>
<td>10.2%</td>
</tr>
</tbody>
</table>

Table 1.3: Results of different methods

1.8.2 Calibration and “Score Gap”

Upon extensive investigation of various changes to the models and the development of a new method for examining and comparing iteratively developed models, which is explored in Chapter 2, we fundamentally sought to identify those features that predicted or “calibrated” the performance of our reranking model. In seeking to calibrate the ranker, we sought some function of multiple variables that describe a candidate set and mapped this to a probability that an oracle would be selected.

Our work identified that the most important predictor for successful discrimination of oracle candidates from non-oracle candidates relates to an approximation of the margin, which we term the “Score Gap”. Recall that the margin for candidate $j$ in set $i$ is defined as $M_{i,j} = F(x_{i,1}; \alpha) - F(x_{i,j}; \alpha)$, where $x_{i,1}$ indexes the oracle candidate. In a test set, the Score Gap, or just Gap, is defined as: $G_{i,j} = F(x_{i,1}^*; \alpha) - F(x_{i,j}; \alpha)$, where $x_{i}^* = \text{arg max}_{j \in 1, \ldots, 20} F(x_{i,j}; \alpha)$. This particular candidate, if there is just one, is the one that is predicted to be an oracle candidate.

Among all variables investigated, the score gap allows us to accurately predict the probability that our methods will correctly identify an oracle among members of the candidate set, and we found a monotonic association between the score gap and the oracle selection probability. This type of accuracy prediction is known as calibration, and Figure 1.11 demonstrates the monotonic nature of the score gap and the oracle precision (i.e. proportion of times the method selected the oracle) of one of our models. This gives us empirical confidence in using the score gap to stratify the data into tiers that are reranked well down to tiers where the reranking is poor.

Figure 1.12 illustrates the cumulative proportion of errors that are attributed to the sets arranged in order of increasing score gap. In general, the problems that remain are in the
lower score gap quantiles. In further work, we stratified the data by the score gap. The lowest stratum was always the worst in terms of the proportion of errors.

### 1.8.3 Methodological Analysis

In the next chapter, we will focus on methodological analyses of such a family of models and the interaction between the resulting sparse models and sparse data. The primary issues addressed are:

- Candidates in the lowest score gap strata exhibit especially rare features, which impacts both the number of modeling iterations and the discriminative capacity of such models.

- The sets, $W^+$ and $W^-$, previously described in Table 1.2, have varying relative impacts in ranking features at each update. This is due to the primary discriminating variables that remain important with each update. This suggests further work in intra-set analyses, such as examination of the overlap of features that occur in an oracle candidate versus non-oracle candidates.

### 1.9 Discussion

This investigation has revealed several notable insights regarding data preparation, the interaction of sparse data and sparse models, and techniques for creating ensemble models. We address each of these in the following subsections.
1.9.1 Data processing: stratification, filtering, transformation, and weighting

As noted in Section 3.5, the data set presented a number of anomalies. Cleaning the data certainly improved the error rate, but did not have a practical impact on the models themselves. Stratification by sentence length had an impact on both the models and the error rates. Transforming the data by way of several dimensionality reduction methods did not lead to improvements, primarily due to data sparsity. Finally, model construction was impacted by the noisy data: the time for construction was increased and the resulting models were denser. Although the models were denser relative to those constructed with clean data, the additional features were those that are quite rare, so the practical impact was minimal.

One of the data processing steps that we did not address is the identification and weighting or elimination of sets that are potentially inappropriate for training and testing, such as those with “bad oracles” or those sets that differ only in the punctuation presented in the candidates. To that end, we believe an additional step linking pre-processing and training could be introduced - some form of plausibility classifier or weighter could be introduced. For instance, if a candidate set is especially bad, the weight assigned to it would be lower. If the candidate set varies on issues that are very minor (e.g. punctuation), then it would be down-weighted: it is implausible that this would be presented by a user, and other processing steps would likely be involved to ensure matching punctuation. Similarly, one could use corpora-based methods to determine which candidate sets are most likely to be encountered in the future, which could be useful for domain adaptation.
1.9.2 Interaction of Sparse Data and Sparse Models

As we have seen, iterative methods for building sparse models over sparse data can be very fast and very accurate. However, with exceptionally sparse data, e.g. candidate sets with features that do not appear elsewhere in the training set, it sometimes becomes impossible to automatically determine which variables to add to a model. For instance, at a given step two variables may have equal support and equal impact on $\Delta ExpLoss$. Moreover, for many of the candidate sets, the differences between candidates are among variables that are very rarely encountered in other candidate sets. As a result, the difference between candidates, and especially between the oracle and non-oracle candidates, is not sizable when projected onto a simple weight vector that has support only the most common dimensions.

One unusual conclusion is that typical Lasso-style regularization methods may diminish the performance of some models in this context, by excluding variables that do not occur sufficiently frequently. As a result, we need to think more carefully about the goal of seeking sparse models: is robustness a greater goal than discrimination? We need to tailor our regularization methodologies to address the level of support for the variables under consideration. This suggests new considerations about hypothesis testing relative to the number of samples presented for each variable. These considerations take on additional importance in a streaming context, as the dimensionality, $P$, continues to grow with the number of observations, $N$.

A more important result is that some subsets of the data are very hard to model accurately, due to the sparsity of the data in those subsets. As a result, any iteratively built model, stopped after a small number of iterations, may never include such variables. Moreover, as the dimensionality grows with the number of observations, it is expected that we will not have prior exposure to the appropriate discriminating variables for some candidate sets. As a result, we need to address both the model search method (e.g. add many dimensions at once) and the intra-set evaluation mechanisms for those cases where the “score gap” is unable to discriminate between candidates in a test set. Even with such approaches, we need to be aware of a hard limit on the ability to model these subsets, and then determine how close we are able to come to this performance limit. Obviously, obtaining models that result in zero errors is impossible, and we should attempt to establish a better estimate of realizable performance.

1.9.3 Ensemble methods

Until recently, ensemble models frequently referred to aggregations of multiple models based on the same structural model family, albeit fitted on different training data subsets or with different training parameters. Our work suggests that this is only part of an improved solution: we must also look for different methods over different subsets of the data. For instance, data with very large score gaps with a basic boosted model are likely to be correct, but this confidence (or calibration) decreases with the score gap. Future ensemble methods to address this may incorporate models fitted over sentences of different lengths, for instance. Further, we may seek to use very different methods for those cases where the score gap is very small, by opting instead to use intra-set methods for identifying important variables. We may also need to return to analyzing the candidates as text sequences rather than as a
“bag of words”. For such an approach, incorporation of sequence similarity measures may be more appropriate.

In combining the two major insights, the noise analyses and the score gap, we can effectively construct an ensemble approach for learning and prediction. When the sample data set involves extreme noise or features that are very, very rare, it is effectively “unlearnable”. For such data, the default model, perhaps the baseline score given to the reranker, is the model used for prediction under these circumstances. When the data set is learnable, we use the model(s) resulting from the algorithms described in this chapter.

Finally, regarding model aggregation methods, we believe that model averaging (bagging) appears to have little effect. Although variable subset selection (e.g. via a method known as “stability selection”, as developed in [15]) appears to have some value, this value is limited in that our need for greatest improvement is in finding discriminating features in the sparser dimensions.

1.9.4 Conclusion

Overall, we have significantly reduced the error rate and are able to approach what appears to be the best realizable rate for models based on linear functions in a paired comparisons approach. In doing so, we have identified several assumptions that need to be revisited regarding both the data and the appropriateness of the families of models that we have explored. As we seek better results, we will ultimately approach a goal of specifying a unique model for each candidate set. Our initial steps in this direction, involving noisy and sparse data, indicate that we can identify the limits of model applicability and data learnability. The goal of a unique model for each candidate set will significantly extend our capabilities for ensemble models that are well-calibrated for a given observation, as we seek to avoid the limitations of using a single universal model for all data.
1.10 Conclusion

In conclusion, we offer a new set of methods for improving ranking results for very high dimensional sparse data, methods for understanding the limits of such methods, and an identification of several issues that arise in at least one data set, which can affect the interpretability and generalizability of both results and models, as there is a large amount of idiosyncratic noise.

We have demonstrated several methods that can be used for reranking of text transliteration candidates, and have demonstrated improvements relative to prior work. Beyond this, we have demonstrated that the sparsity of the data appears to be strongly related to our ability to improve the models. In the next chapter, we explore this in greater depth. We believe that we can demonstrate an upper bound on the performance of all such models within the family of generalized linear models, and thus all such models will be affected by data sparsity. Beyond this, methods used for regularizing models has an adverse impact when the data are both sparse and growing in dimensionality, as is the case for the IME data set.
Bibliography


Chapter 2

Bonsai: Interactive Supervision for Machine Learning

2.1 Abstract

We introduce Bonsai, a visual system developed for statistical machine learning researchers to explore and interact with the model building process and to compare between different models over the same data set. The system is especially valuable for classification problems arising from large and high dimensional data sets, where manual inspection or construction of classification models can be prohibitively time-consuming. In addition, the system encourages a machine learning “guided tour” through the data, improving the user’s understanding of the data and participation in the modeling process. In contrast to much previous work, the emphasis is on considering the joint space of the data and multiple machine learning models, rather than providing either an interface for manual classification or for post-construction analysis of a single model.

2.2 Introduction

Machine learning or statistical learning seeks algorithmic approaches to modeling data and making predictions on new data. However, the growth in scale of applicable data sets and learning tasks has outstripped many tools for carefully supervising this modeling process, to the point that most real-world implementations involve an imbalance in the effort spent preparing data, generating structural models, evaluating model performance, and tuning parameters of a given modeling approach, relative to effort spent in understanding and interacting with the model itself. In such contexts the modeling algorithm is perceived as a “black box”.

In this chapter we introduce Bonsai, a system we have developed with the aim of addressing this problem of supervision by giving the human multiple visual lines of inquiry into the model development process and the interaction of the model with the data. By these means, the user can have a far deeper understanding of the data, specific modeling techniques and their strengths and weaknesses, and then open the door to alternative methods for modeling the data. In effect, we aim to make transparent the inner workings of the “black box”. In
In general, Bonsai allows for a deeper understanding of how to compare different models from a large space of potential models.

The main contributions we make are the following. First, we generalize the model development and model comparison process in applications of machine learning, and show that the core model navigation paradigm of Bonsai is that models are developed through a branching process. Second, we present the design of Bonsai, which enables the visualization of partitions of the joint space of the “modeling space” and data, and the interactions between the model and the data distributions. Third, we illustrate an implementation of Bonsai over a large data set, and show how Bonsai can help in comparing and improving the model generated from “black box” machine learning algorithms.

In motivating the development and usage of Bonsai, we will review techniques for exploration of data and the space of models. In Section 2.3, we will briefly describe machine learning techniques for generating models and existing methods for manual development of comparable models; in Section 2.4 we address usage cases for interacting with models of data and analysis of the data, from the perspective of partitioning the product space of data and models. Section 2.5 introduces Bonsai’s interface and design considerations. In Section 2.6, we discuss the implementation and an example application, and we conclude with a section on future work, in Section 2.7.

2.2.1 Data Visualization

Much work in computer visualization of large, high dimensional data sets introduced methods for speeding up the process of examining lower dimensional projections of data. Some of these methods include scatterplot matrices, conditional plots, and mosaic plots. In addition, systems such as PRIM-9 [8] introduced computer-guided “tours” through the data. Algorithmic methods were subsequently developed to assist in touring the data, by characterizing which dimensions might be especially interesting. These include methods such as Projection Pursuit [10], which involves examining the principal component axes, or linear projections representing the greatest variance within the data, in an algorithmically determined sequence. Other methods, such as Scagnostics [29], introduced measures for suggesting which potential scatterplots may be most worth investigating.

Several commercial applications, such as SAS and JMP, and open source projects, such as R [21] and GGobi [25], allow for very intimate exploration and manipulation of data. For a survey of significant advances in visualization of large and complex data sets, we refer the reader to [27].

2.2.2 Model Visualization

Traditional methods for examining the suitability of a given model for a given data set include: examination of errors (e.g. misclassified test data), identification of influential or leverage points - observations that have a substantial impact on the parameters of the model, and identification of patterns or properties of the distributions of errors made by a model.

\[1\] Note that we are using “branching process” in a methodological or heuristic sense, and not in the probabilistic sense.
Unlike simple linear regression models, many types of machine learning models are very rarely explored visually, as they are often complex functions of numerous explanatory variables. It is more common to plot the performance of a model against measures of the “learning” it has been afforded, such as training set size or the number of iterative calculations performed. One interesting approach is to explore projections of classification boundaries, as in [28].

2.2.3 Product Space Perspective

We have previously alluded to the perspective of Hand, Mannila, and Smyth [15] in that a machine learning model requires the following four elements for construction: data, a structural family of models, a loss function, and an optimization algorithm. In general, we may enumerate or index our selections, such as indices of samples of data, indices for families of models, and so on.

In essence, we can define a product space of data and model-building parameters (family, loss function, optimization algorithm), and it is this product space that we seek to understand with Bonsai. In addition to understanding the training or test data, we may introduce derived data, such as the output of the loss function(s) (e.g. residuals, margins, signs of the errors, etc.), and equally adeptly explore this data.
2.3 Binary Classification Models

We'll review basic notation for machine learning models, in order to identify various questions that motivate the development of Bonsai.

We denote predictor variables, \( x = \{ x_1, \ldots, x_p \} \), response variables, \( y \), and our training set as \( \{ x_i, y_i \}_{i=1}^N \). For a given loss function \( L \), and a family of models, \( F \), we want to find a function \( F \) mapping from \( x \to y \), which minimizes a loss or “cost” function, \( L(y, F(x)) \) over the joint distribution of \( (x, y) \):

\[
F^* = \arg \min_{F \in F} \mathbb{E}_{y, x} L(y, F(x)) \quad (2.1)
\]

The choice of loss function, \( L \), family of models, \( F \), and algorithm for finding the optimal member of the family are the key differentiators of various machine learning algorithms. We will introduce a specific technique, and the reader is referred to [16] for detailed examination of many modeling approaches.

2.3.1 Decision Trees

Tree-based models partition the feature space into successively smaller rectangles. A simple model is fitted over each rectangle, such as a constant value (e.g. the mean response for each observation in the rectangle). In essence, the model can be expressed as: \( f(X) = \sum_{m=1}^M c_m I(X \in R_m) \), where \( R_m \) indexes the rectangles the data has been partitioned into.

This process is principally a result of recursive partitions of the space of data, so each iteration produces a model that is built upon a previous model. As a result, the modeling process adapts to local regions, and the criteria by which a new data partition (and new model) is chosen are tunable and can be readily examined. This approach forms the basis for several systems for interactive decision tree models, such as [19], [1], and [26].

2.3.2 Generalized Model Development

In general, the optimization criteria for finding the best model in a given family of models can be parameterized: a loss function (possibly including complexity-based penalization) is specified, some model parameters may arise from a stochastic model (e.g. estimating optimal parameters for a likelihood function), stopping criteria (e.g. tolerances, numbers of iterations, etc.), per-iteration parameters (e.g. step size parameters for coordinate descent methods), and so on. So, for a given training set and model family, \( F \), the determination of the appropriate classifier involves a search over the space of potential models, subject to optimization criteria.

This idea is very general. Recall that in Equation (2.1), the parameters that affect \( F^* \) were: the family of models, the chosen loss function, and the algorithm by which the optimization was conducted. In selecting the family of models, the user is making a choice of \( \mathcal{F} \), of all such families. While some algorithms are iterative in nature, others are not. In addition to variations in the model-building criteria, different training data sets result in different models. So, one could partition the data into \( K \) subsets, such as \( X_1 \subset X_2 \subset \ldots \).
\[ \ldots \subset X_K \text{ or } \{X_i=1,...,K : X_i \cap X_j = \emptyset\} \text{ – i.e. trained on supersets or disjoint sets of data, respectively.} \]

The essential problem is the partitioning of the space of data and models: \( X \times Y \times \mathcal{F} \), and just as trees offer one model-based approach to partitioning the space of data, we demonstrate that branching processes can be used to examine partitions of the joint space of data and models.

### 2.3.3 Generalized Model Comparison

Generalizing even further, an iterative process for developing models is equivalent to comparing all possible models that are a simple refinement of the given model, and choosing among them to determine the best one.\(^2\) For instance, in coordinate descent methods, in addition to examining variables used for an update or for prior or successive updates, examining the alternative variables that were nearly as optimal, is a key approach to understanding how the model was refined and what alternatives might be considered.

This comparison of models can be generalized for even non-iterative cases: if we are given models \( F_1, \ldots, F_k \), then we can compose a sequence of decisions: “Did Model \( F_1 \) accurately classify \( x \)?”, “Did Model \( F_K \) accurately classify \( x \)?” The models may be listed in any order, though prediction accuracy is a good one. Then, from a usage standpoint, data that end up in the far left was “easy” to classify, while data that ends up toward the right is frequently misclassified by all of the models. This is just one of many ways in which this generalized comparison framework can be utilized to learn about how different models fit the data, in order to understand the data and modeling processes much better.

### 2.4 Usage Cases for Bonsai

**Data Statistics Visualization**  
Basic marginal and conditional distributions of data are a fundamental part of exploratory data analysis, and are the basic exploratory visualization in the data panel. Since classification requires understanding differences in selected conditional distributions, this is key to understanding a classification model. The machine learning algorithm or branching process becomes a guided tour through the various stages of conditioning.

**Model Diagnostics**  
Basic examination of a model involves identifying where errors occur and how various methods for calibration (i.e. estimation of a prediction’s accuracy) relate to the conditional distributions. Because of potential differences in the model-building process, e.g. different selections of training data or different model-fitting parameters, different models may result. These different models may, in turn, lead to differences in predictions, such as differences in the predicted class of an observation. As a result, we may compare the differences in the errors, resulting in new variables, such as tests of classification accuracy, sizes of classification margins, and so on.

\(^2\)It is possible that the model cannot or should not be refined further, such as if it already partitions all of the data, or begins to overfit the data. These are important considerations, and within the kind of usage we envision for Bonsai.
Model and Data Interaction When variables are highly correlated, the model-fitting algorithm may greedily choose just one of the variables. In some cases, this may not be reasonable, but in other cases the user may recognize that the correlation is just an artifact of the sample. Correlation analyses are usually performed before modeling is done, but it is sometimes the case that correlations within subpopulations have a more significant impact on the development of the model. This effect is more readily observed for methods that partition the data or when sparse data are involved.

In addition, it is possible to branch on whether variables were affected (e.g. their predicted class was changed) by an update. If the number of affected observations is particularly small for a given update, relative to those before and afterward, this can indicate very influential points, which should be investigated further - in training, if they are noise or outliers, this could have a significant impact on the quality of the model; conversely, in testing, it would indicate that the update may have been useful for the training, but is irrelevant for testing.

Model Fitting and Fitness of Model to Data A critical interface need for machine learning applications is an easy ability to examine which data were correctly or incorrectly classified. In addition, one can use the interface to examine the training process, to identify which features had the most impact on the successive optimization criteria (e.g. the most impact on the loss function in boosting). For testing, this is very useful in understanding how the loss function varies over the training data and whether it is really producing the best update for the given test data (i.e. the fitness of the model to the test data). In addition, the fitness of the entire family of the model can be examined in the course of this work, to determine if changes should be made to reflect specific interactions or dependencies in the data.

Comparing and Selecting Models One of the most important considerations in “model selection”\(^3\) is not only how well a model fits a given data set, but how well the model performs on held-out data, which may arise from a slightly different distribution than the data used for training. This goal of model robustness can be pursued through several strategies, including re-sampling methods, model sparsity, and fewer splits, in the case of tree-based methods.

By examining how new variables are added to a model and by employing a conservative strategy for building a model, the user can consider if a model may perform better if it is sparser. Alternatively, for very high dimensional, sparse data, it may be the case that an overly sparse model has very poor performance, and relaxing model sparsity considerations could offer a great boost in performance.

In addition to comparing among models, e.g. across model families, the user may find that one model is more interpretable than another, while having an equivalent performance. Interpretability is simply the province of humans (and is often the province of specialists for the data domain - even statisticians or machine learning experts may not have much authority on interpretability for a given model).

\(^3\)Note that we refer to model selection, per the Hand et al. [15] paradigm, rather than variable selection: finding an optimal model may include constraints on the nature of the variables included, but we mustn’t overlook the impact of selecting the appropriate model family or the optimization algorithm used.
Active Learning  One area where humans and computers cooperate more intensely in machine learning is the area of active learning: previously untagged data (i.e. observations without a previously assigned response) are identified as potentially influential in affecting the model, but for which the model has only a weak capability to classify. For instance, the predicted value from boosting may be close to 0, instead of having a magnitude close to +1 or -1. If such a variable could be influential in a model, then it would be among those that are most desirable for a human to tag if tagging resources are limited. By examining the model development process, the user can investigate which subsets of a large data set are least definitively classified by the model, and actively tag new training sets.

2.5 Visualization Design

The goal of the Bonsai visualization is many-folded:

1. To open the “black-box” by visualizing the machine-learning model building process;

2. To allow users to drill-down to see the statistics of the data at a particular point of the process; and

3. To explore, compare, and give feedback on the fitness of different models to the data.

In order to achieve these, the design of the visualization consists of three main panels – Model Panel, Data Panel and Navigation Panel. In the following sections, we will first describe our general visualization framework. Then we will describe each of the three panels in the visualization. Lastly, we will describe the interactions between the panels.

2.5.1 General Visualization Framework

Figure 2.1 illustrates the three major components of the Bonsai visualization. On the left is the Model Panel, which is used to navigate the branching process encountered in building a model. On the lower right is the Data Panel, which is used to explore the distribution of data, via histograms, scatterplots, parallel coordinates plots, etc. On the upper right is the Navigation Panel, which is used to control the visualization displayed in the Data Panel from a meta-level. The Navigation Panel allows for higher level “zooming” over data, by selecting dimensions, statistics and graphs to explore from the data space. Users can select specific dimensions, graphs and statistics to examine; in addition, top-K suggestions can be derived from the machine learning system, or the interface can incorporate analyses derived from Scagnostics [29] (see Section 2.5.4 for further details).

2.5.2 Model Navigation Panel

The Model Navigation panel is the starting point for the user's exploration of the model building process, and investigation into the interaction with the data. The Model Panel visualizes and supports the navigation of the model building process. In this project, we focused on applications whose model building process can be visually explored via a process of branching, where each branching induces a partition of the data space or the product
of the data and model spaces. For example, the Model Panel in Figure 2.1 generalizes the building of a CART tree or the boosting process as a branching process in a tree. For both the CART tree and the boosting algorithm, the branching tree represents the partitioning of data – each node in the tree maps to a unique subset of data and models – based on the branching condition of all of the ancestors of the node in the tree.

### 2.5.3 Data Panel

The Data Panel allows for exploration of the data using projections of the data and examination through various plots, such as histograms, scatterplots, parallel coordinates plots, etc. For example, we examined the income prediction problem described in Section 2.6.3, where the task is to predict a person’s income level, based on 15 predictor attributes, such as education, marital relationship, gender, and age. We can explore the data via a density plot of the two salary classes with respect to age – mapping a 15 dimension space to 2 dimensions. This information can be displayed as a scatterplot of frequencies or as a histogram.

A large variety of visualizations can be shown in the Data Panel. The visualization varies not only in terms of the data or low dimensional projections selected – which variables (e.g. age and salary in the previous example) in the high dimensional data are used, and what statistics (e.g. density) are computed; but also in the selection of the graph type displayed (e.g. histogram, scatterplot).

In addition, the Bonsai visualization can include multiple Data Panels, to enable the user to easily compare histograms of different variables. For example, the user might want to compare the frequency plots for age, relationship and education versus salary class, to understand conditional distributions and the selection of variables for models.

### 2.5.4 Data Navigation Panel

The Navigation Panel is added to solve the problem of selecting among many variables for a closer inspection of the data. In the Navigation Panel we present different choices of visualizations that we can display in the Data Panel: different dimensions, statistics or graphs used in the visualization. Among those, the users can select the visualization of their interest. For example, for the Census Data, we can provide users with the control to choose
between all 15 dimensions to see the data histogram projected to the dimension. In addition, the users can choose among different graphs, such as scatterplots or histograms, and may examine the response via class-based counts or density plots.

Of course, the above method is imperfect, because we are exposing the full possibilities of a huge space without guidance to the users. We are exploring automated methods for guiding users in the exploration of the huge space of visualizations, to find the interesting ones, which best reveal features of the data. Several techniques for doing so include: Scagnostics [29]; selection based on multiple future splits (as in [1], for decision trees) and top-K alternates (applicable in many different instances, such as in boosting, decision trees, and other forward and backward regression approaches); variables already included in the model; variables sorted by Gini, entropy, other splitting criteria; and sorting of variables by the frequency of occurrence (e.g. in sparse data) or discriminative impact (using, for instance, a margin, as in support vector machines (SVMs) and other margin-based families of models).

2.5.5 Interaction

As shown in Figure 2.2, Bonsai is an implementation of “Model Explorer”, a bridge between tools for data management (e.g. PostgreSQL) or modeling tools (e.g. R and MATLAB) and applications such as model or data visualization, debugging of machine learning methods, query answering, among many other uses. In addition, the Bonsai visualization depends upon the Prefuse [18] system for data visualization.

Within the Bonsai visualization system, the most important feature is its interaction between the three panels.

- Model Panel and Data Panel Interaction: when a node in the Model Panel is selected, the visualizations in the Data Panels are changed to default visualizations, and the data set corresponding to the node becomes the active data set.

- Model Panel and Navigation Panel Interaction: when a node in the Model Panel is selected, the choices in the Navigation Panel are changed back to the default settings.
• Navigation Panel and Data Panel Interaction: when a choice of the visualization (e.g. dimension, statistics, graph) is changed, the display in the Data Panel is changed accordingly.

2.6 Implementation

We have implemented the Bonsai visualization interface on top of the Prefuse interactive information visualization toolkit, in Java. Currently, the implementation does not support the interactive query and feedback interface to the data management or the model building tools. Instead, necessary input files are prepared, as pre-fetched query-answer tables. Also, several basic statistical functions were implemented in Java, e.g. to compute frequency counts and density. In future work, we will build the interface between Bonsai and data management and model building tools, and expose an API for higher level applications.

2.6.1 System Architecture

The system architecture of Bonsai visualization is shown in Figure 2.3. The Model Panel is built on top of Prefuse’s TreeView visualization, and the Data Panel is build on top of the ScatterPlot visualization. Model Panel and Navigation Panel can trigger DynamicQuery, which is linked to histograms displayed in the Data Panel. Also, linking and brushing (L&B) functionality exists between the three panels.

![Figure 2.3: Diagram of system architecture](image)

2.6.2 Screenshots of Implementation

In Figure 2.4, a CART decision tree is shown in in the Model Panel, as used for the example in 2.6.3. A specific node in the tree maps to a unique subset (partition) of data. The data in the nodes in the Model Panel are linked to corresponding Data Panels on the right side of the display, for easy comparison between different dimensions. The toolbar above each Data Panel is the Navigation Panel, for selection of variables for the the density plot. Within each histogram, points can be clicked and the interface reports the variable and
its frequency, the response (classification) variable frequency, and the corresponding response (e.g., “salary category”). The example on the right involves variables added iteratively via Boosted Lasso [31] applied to a sparse data set with 500,000 dimensions, linked to multiple calibration variables. Section 2.7 address applications of Bonsai to iterative and ensemble methods.

Figure 2.4: Bonsai interface screenshots

### 2.6.3 Example: Predicting Income Level

Using census data from the UCI/KDD data repository, [2], the machine learning task was: predict whether or not a person reported earning more than $50,000 in the previous year, using variables such as age, educational level, marital status, gender, occupation, and others as predictors. Some of our findings were:

1. Using Bonsai analysis of CART models, we saw very quickly that there was a nonlinear dependence on age: early splits were based on whether or not a person was within a
certain range associated with peak earning years (age 40-50). After this window, it is possible they have retired, and before this window they may be comparatively early in their career. Although this narrative makes sense, it also suggests new features that could be augmented to the data set, to indicate their career stage, such as: {retired, early career, peak earnings years, pre-career/student}. While age-specific policies could necessitate the inclusion of age as a predictor variable, having a career-stage variable helps in understanding other perspectives.

2. We also used logistic regression for the purposes of classification, with a linear model that included all of the explanatory variables. The initial misclassification rate was 14.7%, which was superior to the default CART modeling algorithm in R, which had a misclassification rate of 15.5%. However, the insight that the conditional distribution of \( Pr(Salary|Age) \) is not monotonic suggested that we introduce a new variable for the generalized linear model: \( YearsFromPeak = |Age - Peak| \), where \( Peak = \arg\max_X(Pr(Salary|Age = X)) \). With this new variable, the logistic regression model misclassification rate fell to 14.6%. (Similar results were obtained using a probit model: 14.6% error rate initially, and 14.5% after the new variable.) These correspond to a 1% decrease in errors. By comparing the distributions of errors of CART and logistic regression, we were encouraged to use the output of the logistic regression model in training a new CART model. By incorporating a generalized linear model with a logit link, the CART misclassification rate fell from 15.5% to 15.3%, and when we used the additional “YearsFromPeak” variable, the rate fell to 15.0%, or a 3.5% decrease in misclassifications.

While such results are anecdotal, the general approach of partitioning the product space of data and models has allowed us to very rapidly (i.e. usually in just a few minutes) find shortcomings of specific models and either make adjustments to the model fitting process or adjustments to the data set, such as variable transformations, and re-run the machine learning algorithms.

2.7 Future Work

Our plans for future work are along 4 avenues: extension of the data navigation concept to tools such as Scagnostics, extension of the learning methods to ensembles of methods, extension to unsupervised methods, i.e. those for which a response variable doesn’t exist, and, finally, methods of evaluating the utility of the Bonsai system.

2.7.1 Data Navigation and Scagnostics

According to [29], John and Paul Tukey proposed a means for navigating high dimensional spaces and many choices of pairwise scatterplots, by scoring scatterplots and sorting by these statistics. This approach was implemented and explored in [29], and we believe this could be a very useful aid in further exploring high dimensional space. In addition, we would like to follow up on suggestions from Hadley Wickham to do something similar with histograms and other visualizations.
2.7.2 Ensembles

We believe that in addition to exploring a single modeling process or sets of models trained on different subsets of the data, there is a rich opportunity to explore ensembles of models. These can often be represented as inputs to a boosting algorithm or tree, but we would like to lay out a richer exploration path, if possible.

2.7.3 Unsupervised Learning

One very natural method for obtaining a branching process is to look at agglomerative clustering methods: such an approach builds a natural tree, starting from “leaves”, by iteratively combining points, or sets of points, that are closest among all pairs of distances. Although this is a very simple way to derive a binary tree, any branching process could be used in Bonsai, so other clustering algorithms, especially hierarchical clustering, could be examined.

Another very simple examination would be to examine data projected onto the principal components of the sample covariance matrix. If we suppose that the data has been centered, i.e. the per-column means have been subtracted out, then a branching process based on the sign of the projection onto the first principal component, then the second, etc., would be one easy way of exploring the data, with a very simple branching heuristic.

2.7.4 Layout and Number of Branching Splits

The current visualization shows branches without much reference to the predicted class. For many machine learning algorithms it is feasible to tie the node to a value between +1 and -1, or between 0 and 1 for probabilities, and by representing the nodes as tied to specific values, we believe more insight can be gained into which branches may have had a larger impact than others. It is also possible that a branch may cross over many others, thereby correcting, for a subset of data, errors made at previous branching points. This could be extremely illuminating.

In addition to tying down branches, another natural change would be to have variable numbers of branches from a given node. Such n-ary splits could be useful in examining many other properties, such as predictions into quantiles of values or for comparing multiple models or branching decisions at the same time.

2.7.5 Evaluation

Finally, we would like to further assess this system in a variety of ways. Extending [26] and [1], we believe these interfaces should lead to classifiers that are superior to purely algorithmic ones, since the “cyborg” investigation of the data should lead to a pairing of skills. In the case of ease of use or time to understand a given data set, this is far more challenging, since the target audience would likely have an extremely high level of competence in data analysis and exploration; to test them and this system would require a very challenging test
set. So, we expect evaluation to primarily focus on performance results, sparsity, robustness and useful subjective metrics such as user responses on the interpretability of the models.

### 2.8 Acknowledgments

We are very grateful to Jeff Heer for making prefuse available and for his assistance and advice on its use. We would also like to thank Hadley Wickham, Maneesh Agrawala, and several anonymous reviewers, for numerous valuable suggestions and insights.
Bibliography


Chapter 3

Sparse Matrices: Sparse Decompositions and Ensembles

3.1 Overview

In this chapter, we address one of the most notable sparse data modeling problems of recent years: the Netflix Prize Contest\(^1\). Our work in this competition required utilizing and extending all of the material explored thus far in this thesis: methods for analyzing sparse data, methods for controlling the sparsity of models, extensions of linear models to matrix models, and methods for examining differences between multiple models in order to develop ensemble models. In this chapter, we limit our scope to a method for sparse matrix decomposition and a simple ensemble, and will more fully address ensemble techniques and model diagnostics in future publications.

The algorithms we have developed allow us to create a spectrum of modeling approaches that includes matrix centering at one extreme and matrix factorization at the other extreme, with agglomerative clustering as a bridge between the two. We exploit and maintain sparsity to a degree we have not seen in other works, both in modeling the Netflix Prize data and in other work on matrix factorization. Mathematically, these methods encompass approaches used in other matrix factorization methods, though we have also discovered computational and accuracy limitations and tradeoffs of such generality. In practice, we believe our methods produce results that are more interpretable than other methods and are more robust. Much of our algorithm can be distributed to multiple computers, allowing for simple parallelization speedups. We also address unforeseen shortcomings of our methods, in relation to accuracy and computational costs that arise with such sparse data and sparse models, and how these may be improved in future work.

We describe recommender systems and notable commercial and research applications, as

\(^1\)In most announcements and the contest’s terms, the competition was referred to as the “Netflix Prize” or “Netflix Prize Contest”. The standard name is typically just the “Netflix Prize”, which also commonly appeared with terms like “contest”, “competition”, and “challenge”, which we use interchangeably. We generally prefer to think of a “contest” when addressing issues from the Netflix perspective (e.g. the awarding of prizes, legal issues, etc.), “competition” when writing about participants (e.g. when addressing strategies, teams, etc.), and “challenge” when addressing the modeling problem itself (akin to other “grand challenges” from mathematics and the sciences), though such terms are often used interchangeably.
well as possible motivations for Netflix to offer the Netflix Prize Contest in particular, in Section 3.2. Although the existing use of commercial and research applications in the field of approaches known as recommender or recommendation systems (also known as collaborative filtering or automatic personalization), is already notable, there remain many more potential applications of recommendation systems.

In Section 3.3, we describe the Netflix Prize contest in detail. Section 3.4 addresses several key issues that arise in handling a data set of this size. As the computational complexity of algorithms is dependent on data storage and manipulation tasks, we cannot simply apply an arbitrary algorithm without considering how we will access the data. We also give consideration in Section 3.4 to several data structures and algorithms that facilitate our modeling work.

In Section 3.5, we present basic analyses of the different data sets involved in this contest. In preparation for describing related work and our own algorithms, we have compiled, in Section 3.6, some of our most frequently used notations and acronyms.

Related work and some of the fundamental algorithms involved in our approach are described in Section 3.7, while our novel methodology is described in Section 3.8. Results from applying these algorithms are presented in Section 3.9, and discussed in Section 3.10. Finally, we conclude in Section 3.11.

### 3.2 Motivation: Recommendation Systems and the Netflix Prize

#### 3.2.1 Recommendation Systems

In this chapter, we address one particular data problem, called the Netflix Prize, but this competition is one instance in a large class of problems known as recommender systems or recommendation systems. In general, merchants and many others would like to tailor their interactions with users to account for users’ interests and the propensity of a “favorable” outcome, such as the user buying a product or having a favorable opinion of a movie. As organizations have ever-increasing customer bases, they can collect a great deal of data about these customers, and if the company also has a large inventory of products to offer these customers, they have a challenge in matching the customers to products. To that end, recommendation systems fill a niche. It is important to note that when a firm has a large, diverse inventory, items may be sorted in terms of their popularity, where the most popular item is listed first, and the ranking of the inventory extends to a “long tail” of less popular items, which are offered to satisfy more diverse tastes. Thus, we have the crux of recommendation systems problems: how can we match obscure items with unusual individual tastes, while also correctly matching popular items to users with mainstream tastes? By using similarities with items for which a user has given feedback we can better personalize recommendations for a user. In order to gauge similarity, though, we have to relate items by way of responses given by many users. This dependence on multiple users’ responses is a reason “collaborative filtering” is another term often associated with recommendation systems. In this chapter, we use terms such as “automatic personalization”, “recommendation systems”, and “collaborative filtering” as synonyms and practically interchangeable; in other
contexts, these will have many similarities and some differences.

The current and potential contexts for using personalized recommendations are much larger than many may realize. In part, this is understandable: companies should make the recommendation process as effortless and unobtrusive as possible. Conversely, one reason that personalization may not be well promoted is due to the vast amounts of data that are acquired via interaction with users, which reveal fairly personal insights that may alarm users. In general, personalized recommendations are a benefit to users: exploration of large databases or inventories can be assisted by "learning" from the behavior of similar users as well as the user's own responses to candidate items (e.g., movies) similar to those retrieved from database queries.

Some large-scale examples of personalization include:

- **Amazon**: across its many stores and product categories, Amazon utilizes consumer feedback, viewing history, purchasing behavior, and other information in order to suggest products and promotions.

- **Apple**: One of Apple's most notable and profitable sources of revenue is the sale of digital music. By examining users' purchasing patterns, playlists, and listening behavior, Apple can tailor music recommendations, which can be applied to both suggesting music to acquire and the sequence of music to play, in one's own music library.

- **Streaming media websites** such as YouTube, Pandora and Last.fm, like Apple, also create recommendations for users based on past consumption and interactions.

- **Google**: By examining one's search behavior, visits to websites (via either Google toolbar or sites that use Google Analytics or AdWords), email (if one uses Gmail), and other data streams, Google can tailor search results, advertising, and other recommended content, such as news.

- **Many other large merchants and marketing firms**, from Walmart to coupon and promotion firms, tailor their sales and marketing in the same way as we have illustrated above.

- **Numerous startups** are developing applications in the fields of recommending restaurants, potential romantic partners, scientific articles, iPhone applications, and even beers, based on models trained on large amounts of user data.

---

2 Although users are often identifiable to the service, e.g. Netflix, their behavior may indicate additional information that they have not explicitly revealed. For instance, political preferences may be inferred for some Netflix users. Beyond this, concerns may arise in any context where behavior can be used to infer a user's gender, age, income, illnesses or health conditions, national origin or native language, sexual orientation, and marital status, among other information that they may not desire to explicitly reveal. Such behavioral-based insights are often indirectly revealed to web search engines, but many of these private details may influence taste-based activities such as movie rentals. Indeed, political and commercial organizations have long-used magazine subscriptions, organizational memberships, and demographic information in order to identify and categorize potential voters and consumers, in order to tailor direct mail campaigns to their desired audiences.
Finally, newer areas of applications include applications that are far more serious, such as the nascent field of personalized medicine, where recommendations on therapy are based on a variety of patient diagnostics and medical history information. Diagnostics can include blood assays, genomic tests, and a variety of radiological scans. Patient history can include successful or unsuccessful outcomes of different therapies, allergies, familial information, and so on. However, despite the social and economic impact, ubiquity, and technology of this field, medicine is surprisingly lagging in applications of personalization.

As these examples indicate, there is a wide range of opportunities already, and many more ahead, in fields where we can apply and develop statistical methods for predicting user responses to various suggestions. However, in addition to the privacy impact already mentioned, there are often serious commercial inhibitions against releasing data that could be used advantageously by a competitor, especially if the competitor is also able to learn from algorithmic advances in analyzing the data. Beyond this, many such data sets are either quite small (many startups would be blessed if they had hundreds of thousands of users) or so exceptionally large and complex that only researchers within the organizations could work with the data (e.g. Kaiser Permanente’s patient population of about 10 million people, including massive radiological data sets, and very complex patient history records).

Statisticians may also be particularly interested in the basic properties of such data sets: as the number of users, \( N \), grows, inventory typically grows in breadth as well. In matrix terms, the dimensionality of the item-space, \( P \), which we may represent as columns, grows with the number of users, \( N \), represented by rows. Entries in the matrix may represent users’ feedback or consumption for certain items. As users rarely give feedback, or even have exposure, to a large number of items, such data tend to be very sparse. What’s more, user populations and item inventories often change over time, so there are many ways in which such data tend to be challenging: they tend to be non-stationary, they tend to grow in dimensionality, and they are often very sparse.

### 3.2.2 The Netflix Prize

Netflix is a web-based company whose customers enroll in monthly subscription plans that allow them to borrow DVDs by mail, and, increasingly, access streaming media online. As of 2007, Netflix had more than 11.7 million subscribers, 85,000 DVD titles\(^3\), and more than 1.9 million DVDs shipped per day [5]. It is a business imperative that the firm use a strong system to match users to movies they may enjoy. Although users can query by a variety of movie descriptors such as titles, actors, genres, awards, and other meta-data, an arbitrary user is unlikely to identify all films of potential interest solely through searching the catalog. Therefore, Netflix assists users in discovering films of interest through analyses of users’ rental and ratings history in order to build models to predict how a user would rate a given movie. Netflix’s proprietary system for predicting user ratings is called Cinematch, and Cinematch suggestions are responsible for more than 60% of all rentals. In addition, Netflix’s users are actively training Cinematch: as of a few years ago, users had entered

\(^3\)We will often use the word “film”, which we know is a misnomer, as the DVDs include TV shows and many other non-cinematic content.
more than 2 billion ratings, and are adding about 2 million more per day [5]. By increasing
the prediction accuracy of Cinematcch, it is believed that Netflix could better match movies
to users, thereby increasing user satisfaction and retention, and utilizing a greater fraction
of its extensive inventory, each of which contributes to its profit: users continue to pay
subscription fees month after month, satisfied users may encourage friends to join, and
utilization of more of the inventory means that less well-known films do not gather dust
while waiting for someone to stumble upon them - instead, these movies are promoted to
users who may be inclined to enjoy them.

User satisfaction and consumption of the “long tail” of inventory are two of the primary
motivations in the field of recommendation systems. Although the commercial impact of
recommendation systems for merchants is obvious and a great deal of research existed prior
to the Netflix Prize contest, Netflix stated that they felt they had only achieved an elementary
level of performance, relative to what might be feasible if a greater number of researchers
were to take a look at their problem.

In October 2006, after years of incremental improvements with Cinematcch, the Netflix
Prize contest was announced. Details of the contest are described in the next section. In
order to attract researchers and motivate them, Netflix offered $1 million to the winning
team. The financial stake attracted substantial interest, making the competition one of the
most prestigious challenges of its type. The widespread interest, along with the structure
of the contest and the openness of researchers, spurred many great advances in the field
of recommendation systems. In turn, these developments led to advances in algorithmic
modeling, especially in clustering, ensemble methods, and analyses of sparse data.

3.3 Problem Definition

The Netflix Prize data set consists of a training set and a qualifying set. The training
set is a list of 100,480,507 ratings as 4-tuples (movie ID, user ID, date of rating, rating).
Ratings are integers in the range of 1-5. For the qualifying set, the rating was omitted,
but the remaining 3-tuple is provided. Both the training set and the qualifying set were
sampled from the same set of users and movies. There were 480189 users and 17770 movies.
A separate index matched movie IDs to the title of each DVD and its release date. After the
contest, the observed ratings for the qualifying set were released; this “judging” file included
the quiz or test assignment information for each entry in the qualifying set.

A subset of the training set, with 1,408,395 ratings, is identified as the “probe” data set,
while the qualifying set, with 2,817,131 rows, consists of two subsets, identified as “quiz” and
“test”. These sets were constructed by taking up to 9 of the most recent ratings for each
user, and assigning them to one of these three sets, each with a probability of \( \frac{1}{3} \). The probe
set was included with the training set, so that contest participants could examine data that
arose from the same distribution as the qualifying set. In order to protect the privacy of the
individuals, the ratings data were lightly perturbed: some of the tuples were changed, i.e.
the user, the movie, the date, or the rating was modified [11]. Additional spurious ratings
were added and some ratings were removed. In addition, user errors could add a small degree
of noise to the data.

During the contest, teams made submissions to predict the ratings for the qualifying
set; however, only results on the quiz set were reported, while the test set was the subset used for true evaluation of submissions, in order to identify the winning entry. Although the qualifying set consisted of two disjoint subsets, quiz and test, contest participants were not informed of the assignment for each tuple (however, this was revealed in the judging file released after the contest). As a result, participants could repeatedly submit their full qualifying results, with no capability to “cheat” by fitting to the test subset by using multiple scoring results. Although it is beyond the scope of this particular project, there was a small amount of information leakage that occurred when results were reported for the quiz subset, which allowed competitors to “overfit” on the quiz subset. Although it was uncertain whether multiple quiz evaluations would improve results on the test subset, it was later clear that aggregating so many results did help a small amount on the test subset.

The evaluation metric was root mean squared error (RMSE), i.e. for a submission of predictions, \( \hat{Q} \), with predictions for each of the entries in the qualifying set, \( Q \), indexed by users \( i \in 1, \ldots, N \), and movies \( j \in 1, \ldots, P \), this calculation is expressed in Equation (3.1) (also recall that not all \((i, j)\) pairs are in the qualifying set, \( Q \)).

\[
RMSE(\hat{Q}) = \sqrt{\frac{1}{|Q|} \sum_{(i,j) \in Q} (Q_{i,j} - \hat{Q}_{i,j})^2}
\] (3.1)

The baseline on the test set was established by Cinematc, with a performance of 0.9525 [11]. The grand prize would be awarded to the winning team that surpassed a 10% improvement in the RMSE, i.e. a test RMSE of 0.8572 or less [11]. There were additional clauses in the contest announcement that addressed potential annual awards, called Progress Prizes, and the manner of determining the winner at the end of the contest. The contest would not end immediately when a team passed the 10% threshold; instead, a 30 day “final call” period would begin. The winner would be the team that submitted the best result by the end of the contest, and the earliest such team to do so, if there were ties.

The contest was won by an ensemble team, known as BellKor’s Pragmatic Chaos. The second place team was known as The Ensemble, of which the author of this thesis was a member. The two teams tied on the test set with an RMSE of 0.8567 [12], triggering the tie-breaking condition based on time of submission, where twenty minutes separated the two teams’ submissions. Members of the two top teams describe the final hour of the 3 year contest in [10, 14].

### 3.4 Data Preparation and Handling

In this section, we will address a number of key issues that arose in the processing of the data. Most participants in the competition were challenged by the size of the data as well as the difficulty of adapting basic data structures and algorithms for use with sparse data. While computational considerations impose limits, they also force us to shear away superfluous calculations. Thus, when facing an unfamiliar data set size, we must change from some of our familiar practices in modeling data and develop new methods that are more amenable to the computational resources. Statisticians who are unfamiliar with the issues of computing on this scale or with such data may find our exposition presents several useful techniques for adapting algorithms and data structures.
In Section 3.4.1, we address basic data management and indexing. Our computational work was performed on a variety of computing systems, which are described in Section 3.4.2. Although we used a large amount of resources at times, especially for exploratory research, reproducing the results of this chapter could be done with more modest computing capabilities. Related to this, we describe our data sub-sampling technique in Section 3.4.3. Most of the work of this chapter is reported on a “slice” of the data that is 10% of the size of the full data set, and should be accessible on any modern laptop. Additional topics relating to the dates of ratings, computational efficiency, and indicator matrices are addressed in Sections 3.4.4, 3.4.5, and 3.4.6, respectively.

Because we will often refer to sparse matrices and dense matrices, as well as data sparsity and density, we should now define each of these terms as we intend to use them, especially as “density” already has a meaning for random variables:

- A **sparse matrix** is any matrix of size $N \times P$ where most of the entries are either 0 or unobserved. The identity matrix of size $N \times N$, for instance, is a very sparse matrix. We distinguish between observed and unobserved sparsity as follows: observed sparsity occurs when we know the entries that predominate, e.g. when a 0 occurs, the entry is known to be 0; unobserved sparsity refers to matrices where the majority of entries have not been observed. For instance, most users have rated only a fraction of the movies in Netflix’s inventory, making this a matrix with many unobserved entries. In contrast, matrices derived from language data, where entries reflect the counts of words (indexed by columns) in each sentence (indexed by rows), have many 0s due to the fact that sentences generally have only a small subset of words in the total vocabulary. [Note: We prefer not to use the words “missing entries” to describe unobserved data, as such data are not truly missing, as they never existed, and some assumptions about missingness, such as “missing at random” are fallacious.]

- A **dense matrix** is any matrix with entries that are mostly observed and are non-zero; in practice, all entries are assumed to be observed or may be imputed.

- The **density** of a sparse matrix refers to the ratio of the number of observed, non-zero entries, to the total size of the matrix, $N \times P$. When describing a vector, we refer to its **support**, rather than its density.

Where appropriate, we will clarify whether sparsity arises from a predominance of 0s or of unobserved values. In practice, it is common to assume that unspecified entries are 0, though we address this more carefully with companion indicator matrices, which are described in Section 3.4.6.

### 3.4.1 Storage and Reordering

Data preparation was one of the most challenging hurdles for participants in the competition, especially when the contest began, as the data set size was quite large$^4$ and most participants had never worked with such a large, sparse data set. Uncompressed, the files

---

$^4$One well-known statistician told the author that he and colleagues struggled and gave up because there was “too much data”.
exceeded 2GB in size. In the description above, we omitted descriptions of the files, which gave the 4-tuples as a set of 17770 files, one for each movie, with 3-tuples within each file. Aggregating the files is only a few minutes of work, if done in an efficient manner. Many participants discovered that the size would hobble most database systems, such as MySQL. Instead, it is important to represent the data in a far more compact manner.

For many useful modeling methods, we may ignore the dates of the ratings. In practice, dates are unimportant for many algorithms, though rating order could be useful and it is reasonable to order data that is presented to iterative algorithms, namely the class of algorithms known as on-line learning algorithms. This class of algorithms includes the somewhat misnamed set of algorithms known as “stochastic gradient descent” algorithms; a key algorithm that uses gradient descent optimization appears in Section 3.7.4. In this chapter, we primarily address the 3-tuples of (user ID, movie ID, rating), which are analogous to a standard method for representing entries in a sparse matrix: (column index, row index, entry). Practically, we need not and cannot represent the ratings matrix with all entries: the set of 480,189 users and 17,770 movies would yield a matrix with 8.5 billion entries, while the density of our matrix is only 1.2%. In order to give an understanding of the size of this problem, we should emphasize that 100 million ratings, stored in Matlab or R as 3 columns, will often be represented as 3 8-byte entries per 3-tuple, or 24 bytes per entry. As a result, 2.4 gigabytes is an entry point for storing the ratings matrix in standard statistical packages\(^5\). There are many additional tricks that are feasible, such as storing just the entries per row as a set of lists, but these either inhibit basic matrix analysis or require special matrix calculation code. Fortunately, in our experience, Matlab and R were able to somewhat reduce RAM utilization for the sparse matrix, relative to the 3-tuple representation, by way of more compact data structures.

In order to be concise, we will omit a few tricks that were necessary in order to prepare the data in limited RAM. Our most fundamental data preparation step was to reorder movie and user IDs. In the original data set, movie IDs were contiguous, i.e. there were no gaps from indices 1 to 17770. However, user IDs ranged from 1 to 2649429, while there were only 480189 users. In our experience, gaps serve only to distract the researcher and to complicate implementation of algorithms. We decided to reorder the movies and users by their support, so that the first column was the movie with the largest number of viewers, the second column had the second most viewers, and so on. The users were similarly re-indexed: the first row recorded the ratings of a user who had rated an incredible 17653 titles.

An alternate ordering of the data would be to order by the $L_2$ norm, which is closely aligned with the goal of improving the overall RMSE and is also closely related to the support of each movie. We opted not to pursue this ordering, though the order in which movies were analyzed was determined by the total squared error, which is related to an $L_2$ norm of the residuals. We felt that support, rather than $L_2$ norm, would serve multiple purposes, including interpretability, speed, storage, and algorithmic simplicity. However, these are merely hypotheses, and it could well be the case that a different ordering would confer some benefits.

\(^5\)The user need not supply this amount of RAM: by using “memory mapped” disk storage, the user can reserve the system’s RAM for rapidly-accessed short-term storage during calculations, and use the disk for general long-term storage. We utilized such methods when working on machines with less than 8GB of RAM. Matlab and R both enable disk-based memory mapping.
3.4.2 Computing Infrastructure

In addition to several programming environments, we used a variety of computers for our work. Our primary programming selections were: Perl, for reordering the original text files, Matlab for most of our novel algorithm development and data analysis, and R, for work beyond the scope of this chapter, which involved using a variety of “community” packages as we developed ensembles for use with our team, The Ensemble, during the contest. In addition, most of the Matlab code was later reimplemented in R, in order to make use of cloud computing services.\(^6\)

Our computing infrastructure included:

- The author’s personal laptop with 4 GB of RAM and 2 cores, which was our primary platform for code development and for testing with “slices” of the data (slices generally involved 10% of the users, and are described in more detail in Section 3.4.3). In addition, most visualization was performed on this computer, as network delays often make interactive data analysis tediously slow.

- *Arwen*, a Statistics department server with 64 GB of RAM and 32 cores (8 processors with 4 cores each). This computer was our primary platform for either long-duration computing, large RAM computing (though we rarely needed more than about 5 GB for an individual process, as we preferred to work with multiple slices simultaneously), experiments with different modeling parameters, and for exploration of results across multiple slices. Another machine, *Beren*, with 8 processors and 16GB of RAM, was also utilized on occasion.

- Amazon’s *Elastic Compute Cloud*, also known as *EC2*. EC2 is a commercial “cloud” computing service offered by Amazon. Customers may rent computing facilities by the hour, with “virtual” machines ranging from a 1 core server with about 1.7GB of RAM, to a very large server with 8 cores and about 68GB of RAM. During the contest, the largest available servers had “only” 4 cores and 15GB of RAM. Late in the contest, the author requested and received permission to use up to 100 instances at a time, while the typical maximum at that time was for just 8 instances. Most of the results produced for this chapter were rerun on the larger server type (8 cores, 68GB of RAM).

In the later stage of the contest, the author utilized between 55 and 60 cores simultaneously, though up to 400-500 cores were available. Generally, research was conducted in earlier stages on smaller or academic machines, while competition work was performed on commercial (i.e. EC2) machines\(^7\). Although the extensive computing needed during the final period of the contest involved methods different from those explored in this chapter, being able to efficiently utilize these computational resources depended on skills and insights

---

\(^6\)The choices of programming languages were based on our expertise at the outset of the competition and the state of the art in each environment at the time. In Section 3.10, we address some of the different choices we would now make. For instance, we would no longer recommend Matlab or Perl, and would recommend Python instead.

\(^7\)The distinction between research and contest work involves a number of considerations regarding whose code was run, legal issues of software licenses, intellectual property, and contest rules that are beyond the scope of this chapter. For instance, Matlab could not be used for the contest, due to its academic license restrictions, so models that the author developed for the contest were based on programs written in R.
developed throughout research conducted in the course of this thesis. For instance, such skill-reuse includes knowledge of: modular programming, code profiling and optimization, data sampling (both “slicing” and bootstrap sampling), searching parameters on distributed computers, model aggregation and comparison, managing process “queues”, and efficiently aggregating many hundreds of gigabytes of data files.

3.4.3 Sampling

With a data set of this size, regardless of one’s available computing resources, it is often better to try out algorithms on a subset and then examine the performance on the full set of data: the savings in terms of memory and computational complexity (which translates into time) yield productivity dividends. In addition, one can examine whether results are stable across different samples and use a variety of means to aggregate models fitted on different samples. In this chapter, we focus on subsampling in order to minimize computation time, and will leave robustness and model aggregation to future work.

We identified three basic sampling choices: sample among users, sample among movies, and sample among entries. The third method, sampling among entries, can be rejected as it decreases the density of the matrix, while potentially doing little to reduce the dimensionality (i.e., we could have fewer ratings per user and per movie, but most users and movies may still have at least a few ratings). This could only negatively impact our ability to analyze and model users or movies.

Sampling a subset of films is also unappealing, as we do not know a priori if some movies have a strong association with each other, such as sequels or movies by the same director. On the other hand, with almost 500,000 users, we are examining a population on a par with the adult population of a moderately large city, such as Seattle (pop: 600,000). We believe that an excellent method for subsampling the user base is to sample from the reordered matrix, by identifying rows with the same indices modulo a desired constant (i.e. they have the same remainder when divided by the constant). In our sampling, we chose to use row indices, \( i \), such that \( i \equiv 1 \pmod{10} \), i.e. rows 1, 11, \ldots, 480181, yielding a sample with 48019 rows. This produces a training matrix that can be processed and explored rapidly, even on a laptop computer.

Because of the reordering of users by their support, we can easily bound the differences in density between the 10 subsets. For example, compare the first “mod 10” slice to the second: although row 1 has more ratings than row 2, row 11 has more than row 12, and so on, if we remove row 1 from consideration and compare row 2 to row 11, row 12 to row 21, and so on – each has more entries than its counterpart. As a result, the matrix densities of the slices vary at most in relation to the total number of ratings by the heaviest user, amortized over almost 50,000 other users in the sample. As indicated in Table 3.1, the 10th slice, i.e. all entries with row index \( i \equiv 0 \pmod{10} \), has 16546 fewer entries than the largest slice, or less than 0.2% fewer. In Section 3.5, we report comparisons between the sample and the full training set in terms of the mean and variance of the ratings.
Table 3.1: Number of entries for 10 “mod 10” slices of the training set: we re-index the users by the number of movies they have rated, and then assign users 1, 11, 21, . . . to slice 1, and so on, up to slice 10, which is made of users 10, 20, 30, . . . (i.e. those indices \( i \equiv 0 \pmod{10} \)), as described in Section 3.4.3.

<table>
<thead>
<tr>
<th>Slice (index mod 10)</th>
<th># of entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10040441</td>
</tr>
<tr>
<td>1</td>
<td>10056987</td>
</tr>
<tr>
<td>2</td>
<td>10056086</td>
</tr>
<tr>
<td>3</td>
<td>10053805</td>
</tr>
<tr>
<td>4</td>
<td>10051736</td>
</tr>
<tr>
<td>5</td>
<td>10049939</td>
</tr>
<tr>
<td>6</td>
<td>10042654</td>
</tr>
<tr>
<td>7</td>
<td>10043362</td>
</tr>
<tr>
<td>8</td>
<td>10042654</td>
</tr>
<tr>
<td>9</td>
<td>10041233</td>
</tr>
</tbody>
</table>

3.4.4 Date

Although date was not utilized in our work, it can be stored in the same fashion as the ratings: (user ID, movie ID, date). This would also allow for easy row-wise or column-wise ranking of the sequence in which movies were seen or ranking of the viewers by the order in which they rated the movie.

3.4.5 Computational Efficiency and Memory Efficiency

As Matlab and R store and access matrix data in a column-major manner, it is usually more efficient to access columns rather than rows. As a result, we have found that for some matrices whose data we may access by row or by column, we can gain significant improvements in computational speed when we also store its transpose, as we can access columns in the transpose more quickly than rows in the original matrix.

In addition, some stages of our methods produce and utilize block diagonal matrices, which can be processed in parallel and use a smaller memory footprint. We do not address these implementation details in this chapter, but the reader is advised to consider this when implementing such algorithms.

3.4.6 Indicator Representations

As stated earlier, we would like to identify the visible entries of sparse matrices, independent of their values. To that end, we use a companion “indicator” matrix associated with a given sparse matrix, to indicate which entries are observed and which are unobserved. For instance, if \( X \) is the ratings matrix, of size \( N \times P \) (where \( N \) is the number of users, and \( P \) is the number of movies), \( V \) is a binary matrix of the same size, indicating only whether or not a corresponding entry was observed (i.e. \( V_{ij} = 1 \) if and only if \( X_{ij} \in \{1, \ldots, 5\} \), and 0
otherwise, for \( i = 1, \ldots, N \) and \( j = 1, \ldots, P \). This might be called the matrix of “visible” ratings (or “viewed movies”).

\footnote{We can also create logical matrices associated with particular ratings values, producing 5 matrices that have indicators of whether or not the corresponding entry in \( X \) is equal to the given rating value. Such representations are very useful for examining pairwise comparisons of movies, though analysis of the joint distributions of ratings is beyond the scope of this work.}
3.5 Data Analysis

3.5.1 Overview

Table 3.2 describes the basic sets of data, the training, probe, and qualifying sets, from the perspective of the participants during the contest. The values in parenthesis are for the given subset with the probe data removed. In Section 3.5.2, we compare the data from the probe, quiz, and test sets.

<table>
<thead>
<tr>
<th></th>
<th>Training (w/o Probe)</th>
<th>Probe</th>
<th>10% Sample (w/o Probe)</th>
<th>10% “Probe”</th>
<th>Qualifying</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Ratings</td>
<td>100480507 (99072112)</td>
<td>1408395</td>
<td>10056987 (9916647)</td>
<td>140340</td>
<td>2817131</td>
</tr>
<tr>
<td># of Users</td>
<td>480189 (480189)</td>
<td>462558</td>
<td>48019 (48019)</td>
<td>46223</td>
<td>478615</td>
</tr>
<tr>
<td># of Movies</td>
<td>17770 (17770)</td>
<td>16938</td>
<td>17768 (17768)</td>
<td>10935</td>
<td>17470</td>
</tr>
<tr>
<td>Matrix Density x 100</td>
<td>1.18 (1.16)</td>
<td>0.018</td>
<td>1.1787 (1.1623)</td>
<td>0.028</td>
<td>0.0337</td>
</tr>
<tr>
<td>Mean Rating</td>
<td>3.6043 (3.6033)</td>
<td>3.6736</td>
<td>3.5995 (3.5985)</td>
<td>3.6752</td>
<td>NA</td>
</tr>
<tr>
<td>Variance of Ratings</td>
<td>1.1777 (1.1763)</td>
<td>1.2711</td>
<td>1.1754 (1.174)</td>
<td>1.2676</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 3.2: Summary statistics for the full training set, probe set, and qualifying set, and two sub-samples of the training set. (Note: NA is given for those values unknown to participants during the contest; a post-contest data summary is presented in Table 3.3.) Matrix Density refers to the proportion of entries that are non-empty, i.e. observed.

3.5.2 Post-contest: Comparing Probe, Quiz, Test

As expected, the three subsets, probe, quiz, and test exhibited very similar statistical properties, as shown in Table 3.3.

3.5.3 Sparsity and Support: Per Movie and Per User

Plots in Figure 3.1 display the distribution of the ratings support for movies and users, respectively, for the training and qualifying data sets.

3.5.4 Observations

There are a few peculiarities that we found in modeling the data, which sometimes complicated our work. Most notable is what we call the “Michael Moore” effect. Michael
Table 3.3: Summary statistics across probe, quiz, and test subsets. The Matrix Density is the proportion of entries that are observed.

Moore is an Academy Award-winning American documentary filmmaker whose influence, liberal political interests, and fame have sometimes inflamed American conservatives. This has produced interesting results, especially bimodality in the ratings data for his movies.\footnote{This intense response has led to a “Michael Moore” conjecture: some viewers rate movies without ever seeing them, and their ratings tend to be very negative.}

Another unusual film is “Napoleon Dynamite”, which seemed to have very little association (e.g. correlation and Jaccard index, as discussed in Section 3.7) to any other film, despite many viewers. While this is not a unique incident among films, “Napoleon Dynamite” was the most frequently watched movie that was practically orthogonal to all other films.
3.6 Notation

Table 3.5 lists some of the most common symbols that will be used in the remainder of this chapter.

We will also use a few acronyms. In particular, we will use several values associated with RMSE, or “root mean squared error” as previously mentioned. For a particular ratings matrix, \( X \), with entries \( X_{ij} \), for \{\( (i, j) : i \in 1, \ldots, N, j \in 1, \ldots, P, (i, j) \in \mathcal{R} \} \), we may have a prediction matrix, \( \hat{X} \), with corresponding predictions \( \hat{X}_{ij} \) for each of the (user, movie) pairs in \( \mathcal{R} \). For our error calculations, we will refer to the entries as a list of values, indexed by \( (i, j) \); computationally, we utilize the 3-tuple sparse representation as (user, movie, rating), and simply match the (user, movie) indices across the two matrices.

In this chapter, we will frequently refer to RMSE, MSE (mean squared error), and TSE (total squared error); for reference, the corresponding calculations are in Table 3.4 below. From the 3-tuple sparse matrix representation, if \( X_{\text{Ratings}} \) and \( \hat{X}_{\text{Ratings}} \) are matching ratings and predicted ratings, then \( TSE(\hat{X}) \) is the \( L_2 \) norm of \( X_{\text{Ratings}} - \hat{X}_{\text{Ratings}} \). We prefer these acronyms to related terms, such as standard deviation, variance and \( L_2 \) norm, in order to underscore our objective of reducing RMSE. Also note, that when referring to the qualifying set, instead of the training set, we replace \( \mathcal{R}, X, \) and \( \hat{X} \) with \( \mathcal{Q}, Q \) and \( \hat{Q} \), respectively.
\[
RMSE(\hat{X}) = \sqrt{\frac{1}{|\mathcal{R}|} \sum_{(i,j) \in \mathcal{R}} (X_{i,j} - \hat{X}_{i,j})^2} \\
MSE(\hat{X}) = \frac{1}{|\mathcal{R}|} \sum_{(i,j) \in \mathcal{R}} (X_{i,j} - \hat{X}_{i,j})^2 \\
TSE(\hat{X}) = \sum_{(i,j) \in \mathcal{R}} (X_{i,j} - \hat{X}_{i,j})^2
\]

Table 3.4: Error calculations: root mean squared error (RMSE), mean squared error (MSE), and total squared error (TSE). When referring to the qualifying set, replace \( \mathcal{R} \), \( X \), and \( \hat{X} \) with \( Q \), \( \hat{Q} \) and \( \hat{\hat{Q}} \); these symbols respectively refer to: the set of (user, movie) pairs, the observed (correct) matrix of ratings, and the matrix of predicted ratings. Users are indexed by \( i \), and movies by \( j \).

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N )</td>
<td>The number of users in the ratings set.</td>
</tr>
<tr>
<td>( P )</td>
<td>The number of movies in the ratings set.</td>
</tr>
<tr>
<td>( \mathcal{R} )</td>
<td>The set of ratings pairs for the training data: ( (i,j) \in \mathcal{R} ) if and only if user ( i ) has rated movie ( j ). Note: Not all pairs ( (i,j) ) for ( i = 1,\ldots,N ) and ( j = 1,\ldots,P ) are in ( \mathcal{R} ).</td>
</tr>
<tr>
<td>( \mathcal{Q} )</td>
<td>The set of Qualifying Set ratings pairs. ( \mathcal{Q} ) and ( \mathcal{R} ) have the same users and movies, though no overlap.</td>
</tr>
<tr>
<td>( X \in {1,\ldots,5}^{N\times P} )</td>
<td>The matrix of training ratings, with unobserved entries to impute. Rows correspond to users, columns to movies. Entries are integers between 1 and 5. For ( (i,j) \notin \mathcal{R} ), ( X_{i,j} ) is undefined (as it is not observed). Mathematically, we may denote these entries as 0; in software, these are typically denoted as 0 or NaN (“not a number”).</td>
</tr>
<tr>
<td>( V \in {0,1}^{N\times P} )</td>
<td>An indicator matrix of “visible” (or “viewed”) entries in ( X ); ( V_{i,j} = 1 ) iff ( (i,j) \in \mathcal{R} ), 0 otherwise, for users, ( i = 1,\ldots,N ), and movies, ( j = 1,\ldots,P ).</td>
</tr>
<tr>
<td>( \hat{X}^{(0)} \in [1,5]^{N\times P} )</td>
<td>A centered approximation of ( X ). Entries may be continuous in the range ([1,5] ), though entries that are not in ( \mathcal{R} ), i.e. ( {(i,j) \notin \mathcal{R}} ), are not defined.</td>
</tr>
<tr>
<td>( K \in \mathbb{N}^+ )</td>
<td>The number of “factors” that we will use for factorization. Note: This need not be specified in advance, as it is determined as a result of our algorithms.</td>
</tr>
<tr>
<td>( L \in \mathbb{R}^{N \times K} ) and ( R \in \mathbb{R}^{K \times P} )</td>
<td>“Left” and “Right” factorization matrices that we will use for factoring ( X ), i.e. we attempt to fit ( X = X^{(0)} + LR ). ( L ) denotes a users-by-factors matrix, and ( R ) denotes a factors-by-movies matrix.</td>
</tr>
<tr>
<td>( K^{(t)}, L^{(t)}, R^{(t)} )</td>
<td>For ( t ) successive “refactorizations”, we estimate new values for ( K ), ( L ), and ( R ).</td>
</tr>
<tr>
<td>( \hat{X} \in [1,5]^{N\times P} )</td>
<td>Predicted values of ( X ), e.g. ( \hat{X} = X^{(0)} ) or ( \hat{X} = X^{(0)} + L^{(1)}R^{(1)} ). The value of ( \hat{X} ) will be defined by context. Entries that are not in ( \mathcal{R} ), i.e. ( {(i,j) \notin \mathcal{R}} ), are not defined.</td>
</tr>
<tr>
<td>( \hat{E} \in \mathbb{R}^{N\times P} )</td>
<td>An “Error” matrix: ( \hat{E} = X - \hat{X} ), also defined by context; entries that are not in ( \mathcal{R} ), are not defined.</td>
</tr>
<tr>
<td>( \hat{Q} \in [1,5]^{N\times P} )</td>
<td>A sparse matrix of predicted ratings for elements of the qualifying set, ( \mathcal{Q} ).</td>
</tr>
<tr>
<td>( d(A,B) )</td>
<td>Similarity distance (defined per context by a subscript for ( d )) between two vectors, ( A ) and ( B ).</td>
</tr>
<tr>
<td>( \neg w )</td>
<td>Negation of a logical (binary) vector, ( w ); i.e. ( 0s ) are replaced by ( 1s ), and ( 1s ) are replaced by ( 0s ), and vice versa. Also applicable, element-wise, to all entries in an indicator matrix, such as ( V ).</td>
</tr>
</tbody>
</table>

Table 3.5: Notation reference table
3.7 Related Work

Many different modeling algorithms for collaborative filtering were developed prior to and during the Netflix Prize contest. Some algorithms were simple, such as improvements on centering, while others include: many variations on nearest neighbors; matrix decomposition methods such as SVD and Matrix Factorization; Neural Networks algorithms, such as Restricted Boltzmann Machines; time series models applied to user and movie ratings behavior; and various types of decision tree algorithms such as random forests and gradient boosted decision trees.

In the research for this chapter, we focus on integrating two methods: similarity calculations and matrix factorization methods. In particular, we seek to produce sparse matrices, $L$ and $R$, to estimate $X - X^{(0)}$, where $X$ is the original ratings matrix and $X^{(0)}$ is a centered estimate (e.g. a combination of user and movie average ratings). Unlike other methods, which produce much denser decompositions, we believed that sparser decompositions would be more interpretable and potentially more robust as a result of filtering steps and shrinkage penalties that we will present later.

Our method, which we will describe in Section 3.8 will follow this basic approach:

1. Center the training matrix, i.e. estimate $X$ with $\hat{X}^{(0)}$, using a simple linear combination of user and movie average ratings.

2. Calculate similarities between movies, using several similarity functions.

3. Factor the residual, $X - \hat{X}^{(0)}$, via the following process:

   (a) Anchoring: Select a movie based on its impact on total squared error (TSE), and make this movie the representative film for a factor. Use the similarity distances to identify other films to be assigned to the same factor. Note that this process is very similar to agglomerative clustering, and we will uniquely relate clusters to factors, allowing us to use “factor” as a synonym for “cluster”, in that each movie will be assigned to one and only one factor.

   (b) Initialize the $K \times P$ matrix $R$, the factors-by-movies matrix, and the $N \times K$ matrix $L$, the users-by-factors matrix.

   (c) Iteratively update the factors, reducing the RMSE on the training set, until reaching a stopping point. Identify the resulting matrices as $L^{(0)}$ and $R^{(0)}$.

4. Estimate $X$ as $\hat{X}^{(0)} + L^{(0)}R^{(0)}$; these factorizations may be used to estimate the held-out test data.

5. Repeat Steps 3 and 4 by calculating the residual, $E^{(1)} = X - \hat{X}^{(0)} - L^{(0)}R^{(0)}$, and use this as the input for Step 3, in place of $X - X^{(0)}$. In this process, we build an additive ensemble of factorizations, each based on residuals of prior factorizations, i.e. $X \approx \hat{X}^{(0)} + L^{(0)}R^{(0)} + L^{(1)}R^{(1)} + L^{(2)}R^{(2)} + \ldots + L^{(t)}R^{(t)} + E^{(t+1)}$. We refer to this process as “Layering”, and denote each layer as $\hat{X}^{(i)} = L^{(i)}R^{(i)}$, thus the summation is written as $X = \hat{X}^{(0)} + \hat{X}^{(1)} + \hat{X}^{(2)} + \ldots + \hat{X}^{(t)} + Error$. Note that we do not require the same dimensionality for each layer of factorizations.
In this section, we will cover the basics of each of these steps, as they are commonly practiced, and we will merge them in a new algorithm in Section 3.8. In contrast to related work, we integrate the similarity calculations into the factorization step, Step 3a, and we produce layers of successive factorizations, in Step 5. We address centering and related basic data and prediction normalization in Section 3.7.1. Similarity calculations are addressed in Section 3.7.2, and matrix factorization methods are addressed in Section 3.7.4.

3.7.1 Basics

Two basic components of nearly all systems used in the competition involved bounding and centering. Many algorithms are not naturally constrained to the range $[1, 5]$, so a simple way to make this constraint is to bound, or, as it was more frequently called, “clip”, the predictions and map out-of-range entries to specific values within the range. Centering, so that the mean value of $X - \hat{X}$ is approximately 0, is a vital modeling step that appears quite simple, but prepares all data for subsequent modeling. Considerations involved in clipping and centering are addressed in this section.

Clipping

Both the rules and logic dictated that predictions must be within the range of 1-5. However, many standard modeling methods do not place bounds on the predicted values, so most teams applied some variation of box constraints, known as trimming or “clipping” of predictions to fit within this range. For instance, the author’s team, The Ensemble, clipped values that were below 1 to about 1.05, and those above 5 to about 4.95. This was guided by empirical results and by examining the mean in the probe set for movies whose predicted values were out of range. Clipping, or Winsorization, was primarily employed in final post-processing of the predictions, just before submission, rather than as a pre-processing step. For ensemble methods, many competitors preferred to use clipping for the output of each constituent method.

Centering

In this section we introduce several common methods for centering the ratings matrix. Centering the data matrix was a commonly reported pre-processing step among competitors, prior to matrix factorization, just as centering and scaling are a common practice before calculating a decomposition of dense data via principal component analysis (PCA). It is important to note that “de-centering” is also important, in order to ensure that the mean of the predicted values for the qualifying set reflects the expected mean for the quiz set (or a combination of the quiz and probe mean values). As demonstrated in Table 3.2, the overall mean for entries in the training data was about 3.60, while the probe set had a mean of about 3.67. Due to a variety of possible factors, such as improvements in Cinematc, selection bias of users, a larger movie library that satisfies more users, user retention, and so on, the mean has increased over time.

In this section, we will refer to the estimate of the center as $\hat{X}^{(0)}$, as we intend to estimate $X$ as a sum of multiple modeling results, as: $\hat{X} = \hat{X}^{(0)} + \hat{X}^{(1)} + \hat{X}^{(2)} + \cdots + \text{Error}$, with
\( \hat{X}^{(t)} \) defined as the result of successive matrix factorizations.

The simplest method of recentering is to use the overall mean, \( \bar{X}_{\text{Global}} = \frac{1}{|\mathcal{R}|} \sum_{(i,j) \in \mathcal{R}} X_{i,j} \). On the training set alone, this would yield an RMSE of 1.18, and would yield an RMSE of 1.27 on the probe and qualifying sets (assuming we were to correctly estimate the mean for qualifying set).

The next level of improvement is to calculate the means for each row and column, i.e. to find the user-means and the movie-means. In this case, we have two vectors of “means”, \( \bar{X}_{\text{Movies}} \) and \( \bar{X}_{\text{Users}} \). For a given movie, \( j \), we define \( \bar{X}_{\text{Movies}}(j) = \frac{1}{|\{(i,j) \in \mathcal{R}\}|} \sum_{(i,j) \in \mathcal{R}} X_{i,j} \), and, for a given user, \( i \), \( \bar{X}_{\text{Users}}(i) = \frac{1}{|\{(i,j) \in \mathcal{R}\}|} \sum_{(i,j) \in \mathcal{R}} X_{i,j} \). We may then estimate a matrix \( \bar{X}_{\text{Interpolated}} \) as in Equation (3.2).

\[
\bar{X}_{\text{Interpolated}}(i,j) = 0.5 \bar{X}_{\text{Users}}(i) + 0.5 \bar{X}_{\text{Movies}}(j) \tag{3.2}
\]

An obvious improvement is to replace the coefficients of 0.5 with an optimal weight for combining the two means, which can be found via linear regression, as in Equation (3.3), and make predictions as in Equation (3.4).

\[
X_{i,j} = \beta_0 + \beta_U \bar{X}_{\text{Users}}(i) + \beta_M \bar{X}_{\text{Movies}}(j) + \epsilon \tag{3.3}
\]

\[
\bar{X}_{\text{Balanced}}(i,j) = \hat{\beta}_0 + \hat{\beta}_U \bar{X}_{\text{Users}}(i) + \hat{\beta}_M \bar{X}_{\text{Movies}}(j) \tag{3.4}
\]

**Variations on Centering**

Others have explored many variations on centering, including weighting the contribution of the user and movie mean values by functions of their corresponding, and global, standard deviations, level of support, and other variables.

In [6], Funk proposed several variations for centering \( X_{i,j} \), based on differing methods for weighting the averages of users and movies based on their support. A number of other papers also addressed variations in centering, typically with much less attention and commentary on the rationale for variations. Variations in centering methods typically accounted for only a small proportion in overall performance differences. For other major variations, we refer the reader to [2] and [18] for notable variations published during the competition.

### 3.7.2 Neighbor-based Methods and Similarity Scores

**Overview**

The “basic” methods, e.g. centering models fitted via linear combinations of movie or item and user average ratings, are useful to an extent, but traditionally recommendation systems have used “item-item” similarity scores \(^{10}\) to build recommendations, as in Sarwar et al. [15]. In addition, similarity distances are the basis for clustering approaches such as multidimensional scaling and K-nearest neighbor models. A number of other contest competitors used methods like those of Sarwar et al. to build prediction models that would

---

\(^{10}\) One may often encounter “similarity measures”, where “measure” is used with a general meaning of score or distance, rather than a measure-theoretic interpretation familiar to a statistical audience. As a result, we prefer to use “score” or “distance” rather than “measure”.

be ensembled with others, such as matrix factorization models. In contrast, our approach builds similarity scores into the matrix factorization method. In this section we will describe several similarity scoring methods that we incorporate and the use of such methods in related work.

In the competition, two uses for similarity scores arose: as a key component for nearest neighbor methods, which were ensembled, or “blended”, as predictions along with the results of other methods, and as a post-processing method to refine predictions from matrix factorization methods. We are also interested in using both nearest neighbors and matrix factorization; however, we extend the use of similarity functions as a method of filtering or pre-screening pairs or sets of vectors in order to identify fundamental “factors” that may be used for sparse factorizations. This approach melds agglomerative clustering with matrix factorization, and in Section 3.8, we describe how we use these similarity measures in order to define clusters, which will identify factors (the basics of matrix factorization will be presented in Section 3.7.4).

Although we have yet to define matrix factorization, the impetus for our use of similarity scores is to link factorization to agglomerative or centroid-based clustering, so that factors represent films that are similar according to various criteria or filters. In our approach to factorization, we expect this will make factors more coherent and potentially more robust than other methods. As we shall discuss in Section 3.9.1, our thresholds for similarity are tunable, even to the extremes of both a single cluster containing all films to a complete separation of all films into individual clusters. Before addressing our use of similarity scores, we will review the similarity scores and implementations that we and others have used for the Netflix Prize.

Applications to the Netflix Data Set

An example of neighbor-based predictions is the formulation in Equation 3.5, presented in [16], where a rating $x_{i,j}$ is expressed as a weighted average for movies considered related to a given movie. As noted in [16], one advantage of neighbor-based methods is that the estimation process tends to be completed in a single pass, and tends to have few or no parameters to tune, unlike many training methods.\footnote{The authors of [16] suggest that there is no training to be done, but we regard estimation (e.g. of correlation values) and training as the same. Differences arise between estimation, fitting, or training methods due to the number parameters to initialize or estimate, the time consumed to estimate optimal values, and the need to update models as more data arise (e.g. with more rows or columns in the data matrix). In any case, similarity measures have many attractive properties, including their utility in models, their statistical interpretability, and their relative ease of implementation.}

\begin{equation}
\hat{x}_{ij} = \frac{\sum_{k: (i,k) \in R} s_{jk} f_{jk}(x_{ik})}{\sum_{k: (i,k) \in R} s_{jk}}
\end{equation}

In this case, $i$ indexes the user, $s_{jk}$ is a similarity distance between the $j$-th and $k$-th movie, and $f_{jk}$ is the function that predicts the rating of $j$-th movie based on rating for the $k$-th. The set of the neighborhood of movies is denoted $\mathcal{R}_{jk} = \{i : (i,j), (i,k) \in \mathcal{R}\}$. A common choice of similarity function is Pearson correlation.
For a number of reasons, similarity metrics were applied to just movie-movie pairs, and user-user pairs were ignored. The primary consideration was there were more than 27 times as many users as movies, causing a massive increase in computational costs. This was tentatively, but unproductively, explored by contestants. The impact on performance was negligible, and generally did not justify the computational costs.

One important issue to address is that many similarity measures are typically calculated over fully observed data, and adapting such measures to sparse data without full pairwise support is prone to producing misleading results, especially if unobserved values are replaced with a value of 0, a “default” practice when working with sparse data.\(^{12}\)

In this chapter, we have restricted similarity calculations to pairs of vectors with sufficient pairwise support that the similarity may be meaningful. When two vectors are sufficiently orthogonal in terms of their support (e.g. as calculated via the Jaccard index, which we describe in Section 3.7.2), we usually cannot expect very much information from one variable when estimating the other.

The following sub-sections address tests for similarities of sets, notably the Jaccard index and a score we refer to as “RatioToMin”, which we believe is novel in application to the Netflix data set.\(^{13}\) We refer the reader to Table 3.6 for some of the fundamental binary co-occurrence calculations, for two arbitrary sets, \(A\) and \(B\), using standard set theory notation, and we re-write these calculations using linear algebra on indicator vectors in Table 3.7. The Jaccard index relates the ratio of \(\frac{|A \cap B|}{|A \cup B|}\), while the test that we refer to as “RatioToMin” depends solely on \(\frac{|B|}{|A \cup B|}\), where we assume that \(B\) is the smaller set, i.e. \(N_B \leq N_A\). Details of the implementation of these distances are described in their corresponding passages, though the reader may skip these details if implementation is not a concern, and proceed to Section 3.7.2 to continue reading about other screening methods and an example applied to a small subset of the data.

As a concrete example, we may consider \(A\) and \(B\) to be the sets of users who have watched two movies, say “Annie Hall” and “Bullets Over Broadway”, both directed by Woody Allen. Note that \(A^C\) refers to the complement of the set \(A\); in our case, it would be the Netflix users who have never rated “Annie Hall”. Similarly, \(B^C\) refers to those users who have never rated “Bullets Over Broadway”. (The author of this thesis, were he in the Netflix population, would be in the set \(A^C \cap B^C\), as he has not seen either movie.) In addition, \(N_A\) and \(N_B\) refer to the number of elements of the corresponding sets, which may be interpreted as the number of viewers of \(A\) and \(B\), respectively; \(N\) refers to the size of the total population, which we may analogize to the number of users in our data set.

In order to process the data efficiently, we should consider vectorization of these calculations, which is often far more efficient than examining the sets as sets.\(^{14}\) As we have also

---

\(^{12}\)This has been a common mistake when working with re-centered data: as the expected value is 0, a common mistake is to implicitly assume the safety of using 0 as a placeholder value for missing (i.e. unobserved) entries. We have found this assumption to be specious. For instance, one reason the rating is unobserved is that the user did not rate the movie, because they anticipated that they may not enjoy the movie as much as those movies they have rated. By extension: if forced to rate all movies, most movies may be rated as less enjoyable than those movies that were initially rated.

\(^{13}\)We have not seen this in other works on collaborative filtering, though its simplicity makes it unlikely that it is novel.

\(^{14}\)This need not be the case in theory, but, in the author’s experience, it is currently the case for higher
produced an indicator matrix, $V$, we may express these calculations as a set of dot products of corresponding vectors, $V_A$ and $V_B$, which we will denote as $V_A$ and $V_B$, for simplicity. The corresponding calculations are presented in Table 3.7, where $V_A \cdot V_B$ refers to the dot product of the two vectors; recall also that $\neg$ refers to the negation operation, applied element-wise (e.g. $\neg(0,0,1,0) = (1,1,0,1)$). Note, too, that using columns of $V$ is merely an example: we may extend this to comparing the similarity of users, such as comparisons between the sets of movies viewed by two arbitrary users, “Alice” and “Barbara”.

<table>
<thead>
<tr>
<th>$B$</th>
<th>$A$</th>
<th>$A^C$</th>
<th>Row Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_{AB} =</td>
<td>A \cap B</td>
<td>$</td>
<td>$N_{ACB} =</td>
</tr>
<tr>
<td>$B^C$</td>
<td>$N_{ABC} =</td>
<td>A \cap B^C</td>
<td>$</td>
</tr>
<tr>
<td>Column Sum</td>
<td>$N_A =</td>
<td>A</td>
<td>$</td>
</tr>
</tbody>
</table>

Table 3.6: Binary co-occurrence calculations, in set-theoretic notation, used for such calculations as the Jaccard index and others described in Section 3.7.2.

<table>
<thead>
<tr>
<th>$V_B$</th>
<th>$V_A \cdot V_B$</th>
<th>$\neg V_A \cdot V_B$</th>
<th>Row Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_{AB} = V_A \cdot V_B$</td>
<td>$N_{\neg A,B} = (\neg V_A) \cdot V_B$</td>
<td>$N_B = V_B \cdot V_B$</td>
<td></td>
</tr>
<tr>
<td>$\neg V_B$</td>
<td>$N_{A,\neg B} = V_A \cdot (\neg V_B)$</td>
<td>$N_{\neg A,\neg B} = (\neg V_A) \cdot (\neg V_B)$</td>
<td>$N_{\neg B} = N - N_B$</td>
</tr>
<tr>
<td>Column Sum</td>
<td>$N_A = V_A \cdot V_A$</td>
<td>$N_{AC} = N - N_A$</td>
<td>$N$</td>
</tr>
</tbody>
</table>

Table 3.7: Co-occurrence calculations, using linear algebra on indicator vectors, $V_A$ and $V_B$.

Finally, before we introduce specific similarity distances, we should note that our interest in distances is not so much ratings predictions, as with nearest neighbors methods, as it is in allowing us to screen pairs of vectors, such as movies, for exclusion from further consideration. (Recall that we will use selected films as “anchors”, and only wish to consider similarities with films that may be in the same cluster as a given anchor film.) For instance, our first similarity measure, the Jaccard index, is used to screen out a large subset of movie-movie pairs.

We shall later address modeling benefits that arise from these methods, but we should also note the major computational benefit in not needing to store all movie-movie similarity scores. A fully dense square matrix of 17770 dimensions per side consumes about 2.5 gigabytes in Matlab and R - a consideration that was not trivial when Netflix began the contest. In contrast, some of our filtered similarity matrices consume between 50 and 500 megabytes. By “filtering”, we mean that all entries below a given threshold are replaced with 0, yielding a sparse similarity matrix.

level languages such as Matlab and R. In addition, linear algebra libraries, such as LAPACK and BLAS have been extraordinarily well optimized to maximize throughput for the processor and primary memory components of a computer, via packages such as ATLAS, Goto BLAS, and special libraries from Intel and AMD.
Jaccard Similarity Index of Viewing Data

During our investigation of pairwise similarities, the Jaccard index proved to be one of the most useful methods for comparing pairs of vectors, as part of the process of identifying movies that are in the same cluster as a given “anchor” film. The Jaccard index for two arbitrary sets, \( A \) and \( B \) is the ratio of the size of their intersection to the size of their union, i.e. \( d_{\text{Jaccard}}(A, B) = \frac{|A \cap B|}{A \cup B} \). In the case of the ratings data, we can apply this similarity score to the indicator vectors for two movies, from the matrix \( V \).

The next section describes very specific implementation details in order to rapidly perform the Jaccard index calculations, and the reader may safely skip this section if implementation details are not of great concern. We will first introduce calculations for vectors, and extend this to a sequence of calculations for a matrix.

Implementation of Jaccard Similarity Index

Vector Formulation of the Jaccard Index Calculation

When implementing the Jaccard index in high level languages, it is preferable to use vectors of indicators, rather than sets, as set comparisons tend to be much more expensive than manipulation of bit vectors. Assume that \( V_A \) and \( V_B \) are column vectors from \( V \), with support (i.e. the number of non-zero entries, or cardinality) \( N_A \) and \( N_B \), and joint support \( N_{AB} \). Then, we may calculate the Jaccard index as in Equation (3.6), which makes use of the inclusion-exclusion principle in the denominator. Equation (3.7) illustrates the use of dot products in order to calculate the Jaccard index with the logical vectors.

\[
d_{\text{Jaccard}}(V_A, V_B) = \frac{N_{AB}}{N_A + N_B - N_{AB}}
\]

\[
d_{\text{Jaccard}}(V_A, V_B) = \frac{V_A \cdot V_B}{(V_A \cdot V_B) + (V_B \cdot V_B) - V_A \cdot V_B}
\]

Matrix Formulation of the Jaccard Index Calculation

In order to exploit infrastructure developed for sparse matrix multiplication, we have refined the calculations using the \( N \times P \) indicator matrix, \( V \), (previously identified in Table 3.5), where \( V_{i,j} = 1 \) if \( (i, j) \in R \), and 0 otherwise, for users \( i = 1, \ldots, N \) and movies \( j = 1, \ldots, P \).

First, we calculate pairwise intersections, then exploit the inclusion-exclusion principle (using a temporary “super union”: \( |A| + |B| \)). In Equations (3.8 - 3.10), all of \( T_1, \ldots, T_5 \) are \( P \times P \) matrices, where entries that are not specified should be assumed to be 0 (e.g. the off-diagonal entries in \( T_2 \)). As \( T_2 \) is non-zero only on the diagonal, we may refer to \( T_2(i,i) \) as \( T_2(i) \). [In short, \( T_1 \) counts the number of times both movies \( i \) and \( j \) have been watched, where the diagonal entries count the number of times individual movies have been watched.] For \( T_5 \), the ratios between entries are calculated element-wise (denoted by the notation ./, borrowed from Matlab).
\[ T_1 = V' \ast V \text{ (Pairwise intersections)} \]  
\[ T_2(j) = \sum_{1 \leq i \leq N} V_{ij} = \text{Diag}(T_1) \]  
\[ T_3 = (T_2)_{j=1}^{P} + (T_2^T)_{j=1}^{P} \]  
\[ T_3(i, j) = T_2(i) + T_2(j) \text{ (Pairwise super-unions)} \]  
\[ T_4 = T_3 - T_1 \text{ (Pairwise unions)} \]  
\[ T_5 = \frac{T_1}{T_4} = T_1 / T_4 \]  

Equation (3.10) uses the inclusion-exclusion principle to deduct \( T_1 \), the number of intersections, Equation (3.8), from \( T_3 \), the “super-unions” (i.e. with redundancies).

The following key may be useful in order to draw an analogy with our earlier calculations, where \( N_{AB} \) is the count of the number of items in both sets A and B, and \( \sim \) indicates that the two values are related to each other.

\[ T_1 \sim N_{AB} \]  
\[ T_2 \sim N_{AA} \]  
\[ T_3 \sim N_A + N_B \]  
\[ T_4 \sim N_A + N_B - N_{AB} \]  
\[ T_5 \sim \frac{N_{AB}}{N_A + N_B - N_{AB}} \]

**Pearson Correlation of Ratings Data**

Another filter we incorporated was Pearson’s correlation coefficient. Many competitors used correlations in their post-processing, especially the BellKor team, e.g. in [1] and [3]. In this section, we will focus on implementation, as correlation will be familiar to the reader.

Because our data set is sparse, we do not have pairwise complete observations for practically all pairs of movies, so we must modify our correlation calculations so that only observed pairs are used in estimating movie-movie correlation. In addition, as with the Jaccard index calculation, we will modify the standard calculation to optimize the calculation via vector operations. Many contestants mentioned on the Netflix Forum that they did not attempt to calculate correlations due to the computational complexity; by reformulating the calculations and by making use of optimized linear algebra libraries (e.g. those produced by chipmakers Intel and AMD), we were able to calculate all pairwise correlations in under an hour, using the full training data set, on a machine with 8 cores.

Suppose we compare two movies’ columns in the sparse ratings matrix as real-valued vectors \( a \) and \( b \) (using lower case to distinguish from sets \( A \) and \( B \), as used above), with cardinality (i.e. the number of observed entries) \( N_a \) and \( N_b \). With these vectors, we may calculate the sample correlation as in Equation (3.12)\(^{15}\). Unfortunately, this approach has

\(^{15}\text{Note that we can assume that pairwise missing entries are 0, as this will not affect the sum.}\)
a number of problematic issues, not least of which is the possibility that the population of people who watch both movies may have different mean ratings for each movie, relative to those who watch just one film. In order to examine the correlation on the joint subset of users, we reformulate \(a\) and \(b\) as in Equations (3.13) and (3.14), and calculate the correlation over their intersection, as in Equation (3.15). In order to interpret (3.13) and (3.14), we simply keep both vectors the same if both have entries in the same position. However, if one of the vectors is empty in a given position, then we replace the corresponding entry in the other vector with a 0.

\[
\hat{\rho}_{a,b} = \frac{(a \cdot b) - (\bar{a})(\bar{b})}{[(a \cdot a) - (\bar{a})(\bar{a})][(b \cdot b) - (\bar{b})(\bar{b})]}
\]

\[
= \frac{(a \cdot b)}{N_{ab}} - \left(\frac{\sum a}{N_a}\right)\left(\frac{\sum b}{N_b}\right)
\]

\[
\sqrt{\left[\frac{(a \cdot a)}{N_a} - \left(\frac{\sum a}{N_a}\right)^2\right]}\sqrt{\left[\frac{(b \cdot b)}{N_b} - \left(\frac{\sum b}{N_b}\right)^2\right]}
\]

(3.11)

(3.12)

\[
a^*_i = \begin{cases} a_i & \text{if } b_i \neq \emptyset \\ 0 & \text{otherwise} \end{cases}
\]

(3.13)

\[
b^*_i = \begin{cases} b_i & \text{if } a_i \neq \emptyset \\ 0 & \text{otherwise} \end{cases}
\]

(3.14)

\[
\hat{\rho}_{a^*,b^*} = \frac{(a^* \cdot b^*)}{N_{ab}} - \left(\frac{\sum a^*}{N_{ab}}\right)\left(\frac{\sum b^*}{N_{ab}}\right)
\]

\[
\sqrt{\left[\frac{(a^* \cdot a^*)}{N_{ab}} - \left(\frac{\sum a^*}{N_{ab}}\right)^2\right]}\sqrt{\left[\frac{(b^* \cdot b^*)}{N_{ab}} - \left(\frac{\sum b^*}{N_{ab}}\right)^2\right]}
\]

(3.15)

Although we are describing very simple calculations, we believe it is very important to reduce each method to its simplest form, as these will be necessary for efficiently calculating correlations in a RAM constrained environment over just those pairs of observations that two vectors have in common. Note that, in this case, \(a \cdot b\) will assume that the entries are 0 for the unobserved values in either vector, which allows us to calculate the dot product only over entries that are observed in both vectors.\(^{16}\)

Ratio of Intersection to Minimal Support Value (RatioToMin) for Viewing Data

We include another similarity score, related to the Jaccard index, in our algorithm. Recall that the Jaccard index compares the size of the intersection of two sets, \(A\) and \(B\), to their union. Suppose that we are screening all movies for their similarity to \(A\), and that we reject from consideration all movies with a Jaccard index less than some threshold, i.e. \(\frac{|A \cap B|}{|A \cup B|} \leq T_{\text{Jaccard}}\). With this filter, we run a risk of ignoring movies that simply have a low overall support relative to the support of \(A\), i.e. if \(|A \cap B| \leq |B| \leq T_{\text{Jaccard}} \cdot |A \cup B|\), as

\(^{16}\)Note that, unlike other methods that make assumptions about the centering of data in order to justify the use of 0 entries, this formulation does not depend on any such assumptions.
opposed to \( T_{jaccard} : |A \cup B| \leq |B| \). To that end, we introduce a new similarity calculation, \( d_{RatioToMin} \), as shown in Equation (3.16), below. Note that we need not calculate the minimal support if we have already ordered our matrix by support: the set (e.g. movie column vector) with a higher index will have a lower support by definition.

\[
d_{RatioToMin}(V_A, V_B) = \frac{|A \cap B|}{\min(|A|, |B|)} = \frac{N_{AB}}{\min(N_A, N_B)}
\]

**Partial Chi-squared Test**

For another simple similarity distance, we turn to statistical tests for the independence of two indicator vectors, say, \( V_A \) and \( V_B \), each of length \( N \). Although a chi-squared test statistic for testing the independence of these two vectors could be appropriate, for our purposes (namely, to eliminate items that are not sufficiently similar to a given anchor movie) we found it sufficient to calculate only one of the summands for this statistic.

If we assume the two vectors respectively have support of \( N_A \) and \( N_B \) (i.e. their total non-zero entries, which happens to be their \( L_1 \), as they are indicator vectors), then we may test the difference between the observed number of co-occurrences versus the expected number of co-occurrences, as follows:

\[
p_A = \frac{N_A}{N} \quad e_{AB} = p_A \cdot N_B \quad d_{pchi}(V_A, V_B) = \frac{(V_A^T \cdot V_B) - e_{AB}}{\sqrt{e_{AB}}} \]

From the perspective of Table 3.7, we are calculating \( \frac{N_{AB} - \frac{N_A N_B}{N}}{\sqrt{\frac{N_A N_B}{N}}} \). Note that we could calculate the chi-squared test statistic by summing \( [d_{pchi}(V_A, V_B)]^2 + [d_{pchi}(V_A, -V_B)]^2 + [d_{pchi}(-V_A, V_B)]^2 + [d_{pchi}(-V_A, -V_B)]^2 \), where \( - \) indicates the logical negation operation. There are two reasons we think this is just as, if not more, effective as fully calculating the chi-squared statistic: first, we are using several other similarity distances in conjunction with this, which already gives us a fairly well screened set of similar movies; the second is that the chi-squared statistic is affected by the large proportion of viewers that haven’t watched either film.

**Additional Screening**

We also pursued several additional similarity distances, many of which are nearly monotonically associated with the above distances, including cosine similarity (very similar to the Pearson correlation coefficient), mutual information, and a variety of tweaks on \( L_1 \) and \( L_2 \) distances. However, we found few instances where additional screening helped refine lists of neighbors of a given movie.
3.7.3 Anchoring Example

We will present our full algorithm in Section 3.8, but we would like to demonstrate usage of these similarity calculations at this point, in order to give an understanding of their use, both in our application and as an example of the relation to neighbor-based methods.

Part of our algorithm involves what we call “Anchoring”, which involves selecting a representative item in a cluster, and then screening all other items for inclusion in the cluster. Our primary interest is in movie-based anchoring. To illustrate anchoring, we have selected several movies and will step through the anchoring process. First, we have calculated similarities according to Jaccard and correlation, and the results are presented in Table 3.8.

For this example, we sort movies by their total squared error, in order to focus on movies that contribute most to the RMSE. In this case, “Miss Congeniality” is, ironically, the film that is most popular, i.e. most rated, though it is not necessarily the most well-liked; it is also the movie with the largest total squared error (TSE). This movie anchors our first factor. Let us suppose that our Jaccard threshold is 0.2 and our correlation threshold is 0.4, for identifying movies that are in the same cluster as our anchor film. In Table 3.8, we see that the Jaccard similarity with the next highest TSE movie, “Pearl Harbor” is quite high: 0.48, but the correlation is below our threshold. Already, we know now that “Pearl Harbor” will anchor the second factor. Continuing through the list, we also skip “Lost in Translation” (negatively correlated) and “Independence Day”, and then add “Sweet Home Alabama” and “The Wedding Planner” to our first factor. The movies that have been assigned to the first factor are now marked “inactive” (these appear stricken out in the table), and “Pearl Harbor” is then compared with the remainder of the active set. In this case, “The Patriot”, another war film, based on the American Revolutionary War, with a romantic sub-plot, is added to the same factor. Notably, “Independence Day”, a seemingly patriotic action film, with a more comedic, than romantic, sub-plot, is not included in this factor. Ratings for “Lost in Translation” are also negatively correlated with ratings for “Pearl Harbor”, so it will become its own anchor. We found that “Lost in Translation” tended to anchor a factor with popular, but slightly eccentric movies; such movies tended to be thought-provoking, but not necessarily emotionally provocative. The film “Independence Day” tended to anchor factors with very large block-buster action films, including “Con Air” and “The Rock”.

In conclusion, our example clusters are as follows:

1. Miss Congeniality, Sweet Home Alabama, The Wedding Planner
2. Pearl Harbor, The Patriot
3. Lost in Translation, The Royal Tenenbaums, The Life Aquatic with Steve Zissou
Table 3.8: Anchoring example, for Section 3.7.3. Entries represent (Jaccard, Correlation) similarities for the pair of movies. **Bold** values indicate similarity scores that exceed thresholds of (0.2, 0.4), for the Jaccard index or correlation, respectively. Anchoring begins with “Miss Congeniality”. Once a movie is assigned, its similarity scores have been stricken through for subsequent columns.

### 3.7.4 Matrix Factorization

Matrix factorization is a sub-class of general matrix decomposition methods, which includes methods such as Singular Value Decomposition (SVD), Principal Component Analysis (PCA), QR decomposition, and others. Our particular interest involves decomposing the ratings matrix, $X$, into two matrices that reflect user and movie projections onto a set of “factors”. Although it is popular in the Netflix literature to refer to these matrices as $U$ and $M$, referring to users and movies, we prefer a more general notation of $L$ and $R$, to reflect the left and right factor matrices.

As matrix factorization was a critical component of many algorithms in the competition, the literature by competitors is exhaustive, covering dozens of variations. Algorithm 3 describes our distillation of the essential steps of most algorithms, though it does cover some of the more unusual variations of matrix factorization. In general, variations among the algorithms occur in initialization, Step 2, and the updates to $L$ and $R$, Step 3.

<table>
<thead>
<tr>
<th></th>
<th>Miss Congeniality</th>
<th>Pearl Harbor</th>
<th>Lost in Translation</th>
<th>Independence Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total squared error (TSE)</td>
<td>249598</td>
<td>217127</td>
<td>236094</td>
<td>202689</td>
</tr>
<tr>
<td>Miss Congeniality</td>
<td>(1.00, 1.00)</td>
<td>(0.48, 0.35)</td>
<td>(0.29, 0.20)</td>
<td>(0.52, 0.34)</td>
</tr>
<tr>
<td>Pearl Harbor</td>
<td>(0.48, 0.35)</td>
<td>(1.00, 1.00)</td>
<td>(0.24, 0.25)</td>
<td>(0.47, 0.39)</td>
</tr>
<tr>
<td>Lost in Translation</td>
<td>(0.29, 0.20)</td>
<td>(0.24, 0.25)</td>
<td>(1.00, 1.00)</td>
<td>(0.30, 0.20)</td>
</tr>
<tr>
<td>Independence Day</td>
<td>(0.52, 0.34)</td>
<td>(0.47, 0.39)</td>
<td>(0.30, 0.20)</td>
<td>(1.00, 1.00)</td>
</tr>
<tr>
<td>Sweet Home Alabama</td>
<td>(0.48, 0.44)</td>
<td>(0.37, 0.37)</td>
<td>(0.29, 0.16)</td>
<td>(0.30, 0.30)</td>
</tr>
<tr>
<td>The Wedding Planner</td>
<td>(0.46, 0.50)</td>
<td>(0.34, 0.49)</td>
<td>(0.25, 0.20)</td>
<td>(0.37, 0.32)</td>
</tr>
<tr>
<td>The Patriot</td>
<td>(0.51, 0.26)</td>
<td>(0.50, 0.43)</td>
<td>(0.29, 0.21)</td>
<td>(0.50, 0.37)</td>
</tr>
<tr>
<td>The Royal Tenenbaums</td>
<td>(0.31, 0.20)</td>
<td>(0.25, 0.27)</td>
<td>(0.44, 0.46)</td>
<td>(0.32, 0.20)</td>
</tr>
<tr>
<td>The Life Aquatic with Steve Zissou</td>
<td>(0.13, -0.19)</td>
<td>(0.13, -0.23)</td>
<td>(0.23, 0.40)</td>
<td>(0.15, -0.17)</td>
</tr>
<tr>
<td>Con Air</td>
<td>(0.50, 0.32)</td>
<td>(0.51, 0.37)</td>
<td>(0.25, 0.18)</td>
<td>(0.52, 0.44)</td>
</tr>
<tr>
<td>The Rock</td>
<td>(0.45, 0.25)</td>
<td>(0.47, 0.31)</td>
<td>(0.25, 0.14)</td>
<td>(0.49, 0.42)</td>
</tr>
</tbody>
</table>
Algorithm 3 General Matrix Factorization Algorithm

Inputs:

- \(X\) - Matrix of Ratings
- \(K\) - Dimensionality of factors (i.e. \(L\) and \(R\) are of size \(N \times K\) and \(K \times P\), respectively)
- \(\lambda\) - The “learning rate”, used to adjust step size in gradient descent
- \(\nu\) - An \(L_1\) regularization penalty to apply in the update step
- \(N\) - The number of users
- \(P\) - The number of movies
- \(\mathcal{R}\) - The set of all ratings pairs, i.e. \(\{(i,j)\}\), for which there are ratings in \(X\).
- Stopping criterion (e.g. tolerance for updates to \(L\) and \(R\), number of iterations, tolerance for changes in RMSE, etc.)

Steps:

1. Center the ratings matrix, \(X\) [Defined per method; basic approach is a weighted average of movie means and user means.]
2. Initialize \(L\) and \(R\) [Defined per method; basic approach is to use uniformly distributed random values between \([-1,1]\).]
3. For each factor, \(k = 1 : K\)
   
   (a) Calculate the error matrix: \(E = X - LR\)
   
   (b) (Option 1: Gradient descent on \(L_2\) loss) For each user, \(i \in \{1, \ldots, N\}\), and movie, \(j \in \{1, \ldots, P\}\), pair in the ratings set, i.e. for each \((i,j) \in \mathcal{R}\), update the entries in \(L\) and \(R\) according by a step in the opposite direction of the gradient. Instead of iterating over every user, we may take the sum of updates per user or movie for the given factor as follows:

   - Update \(L_{i,k}\) as: \(L'_{i,k} = L_{i,k} + \lambda \left( \sum_{(j; (i,j) \in \mathcal{R})} E_{i,j} R_{k,j} \right) - \nu L_{i,k}\)
   - Update \(R_{k,j}\) as: \(R'_{k,j} = R_{k,j} + \lambda \left( \sum_{(i; (i,j) \in \mathcal{R})} E_{i,j} L_{i,k} \right) - \nu R_{k,j}\)
   
   (c) (Option 2: Alternating Least Squares) Hold the \(L\) matrix constant and recompute the corresponding entries in \(R\) as a least squares regression, with optional \(L_1\) and \(L_2\) shrinkage. After this, fix \(R\) and recompute entries in \(L\).

4. Repeat 3 until stopping criterion is reached. Upon termination, return \(L\) and \(R\).

Although the algorithm appears quite simple, there are a number of variations and a few inherent problems in this approach. In particular, we would like to address two: the a priori assumption of the dimensionality of the factors, \(K\), and the loop process.\(^{17}\)

\(^{17}\)Depending on the algorithm, other concerns include: the solution may not be unique (this is especially true when the initialization of \(L\) and \(R\) is based on random or arbitrary values), instability in estimates of
Regarding the number of factors, $K$, most practitioners attempt to identify $K$ via trial and error based on the performance of the matrix factorization algorithm. We found this quite unsatisfying, as we believe that $K$ can be “discovered” in the process of the decomposition, by increasing $K$ subject to additional parameters used in fitting $L$ and $R$.

A more serious concern arises in the updating step, 3b, where each factor is updated with the sole consideration of reducing error. This is unsettling, as residuals that “remain” after updating a given factor carry over as the error to be fitted by the next factor. We will elaborate on this issue via a thought experiment.

Suppose that we have only a few movies, and that $R$ consists solely of two orthogonal rows, i.e. two distinct factors that are orthogonal to one another. In essence these define weights for two distinct clusters or groups of movies. We may suppose that these films are for two mutually exclusive user populations. If there is also no overlap in the user set for weights on these two factors, there is no constraint against merging, or summing, the two orthogonal vectors as one vector (i.e. factor), and collapsing the users’ weights to a single vector (i.e. one factor). Suppose that we merge these factors, with disjoint movies, and subsequently add to our population a few users who have rated movies from these previously distinct film clusters (e.g. relatively eccentric viewers or hyperactive raters). As we update such users’ weights for the distinct movies, updates to $L$ for these users will, in turn, affect updates to $R$. However, those updates in $R$ will impact subsequent updates in $L$ for other users. We initially assumed that the two factors are disjoint, yet now we see that updates to weights in one subset can lead to updates in the other subset; this is because we have made no constraints to enforce orthogonality.

Another question arises: suppose we see improvements in the overall RMSE, is non-orthogonality of factors bad? I.e. should factors indicate cluster membership or should we merely seek globally optimal solutions to our gradient descent on $L_2$ loss? We sought to answer this question, and our basis for concern rested on earlier work with sparse data and sparse models. In our experience with sparse data and iterative fitting algorithms, we have observed that during fitting, overall error may decrease, but errors can “propagate” across dimensions, as the iterative fitting algorithm optimizes for a subset of observations with a given feature. In other words, these iterative updates may improve some predictions, while other predictions may worsen. In the case of matrix factorization, methods such as those in Algorithm 3, Step 3c, and similar approaches known as alternating least squares ($L$ is held constant while $R$ is updated, and vice versa), such error propagation behavior would be observed as errors shift across users and movies. For instance, during the loop, Step (3) and Step (3b), of Algorithm 3, users who have watched many movies of a given factor would have a greater weight in loss calculations for the factor than users who have observed fewer such movies (a similar claim may be made for the movies). Thus, the $R$ matrix would be updated more often to accommodate more ratings from a higher-volume user, which would have an impact on the corresponding entries in $L$ for lower-volume users, or, under different optimization functions, a gradient-based update for an entry in the $R$ matrix may be substantially larger for a low-volume user.

the factors, and over- or under-fitting low-support users and movies.
Iterative SVD (Funk)

In [6], Brandyn Webb, under the alias “Simon Funk”, described several methods that, when combined, yield a rapid and strong algorithm. The core method is an iterative approach to calculating an SVD for matrices with missing (unobserved) data, with several types of regularization that improve the accuracy. This appears to be the first known use of iterative matrix factorization methods for collaborative filtering or recommendation systems, and built upon earlier work by Gorrell [7], which presented an iterative method for singular value decompositions for natural language processing, where, as we have seen in Chapter 1, the data can be very sparse.

Before describing Funk’s algorithm, we need to address the notation and assumptions. For a centered matrix, $X$, the method factors it into the product of two matrices, $U$ (user) and $M$ (movie), with $K$ factors. For simplicity and generality, we have chosen a different naming scheme, using $M$ to represent the product matrix, and $L$ and $R$ to represent the left and right factorizations, respectively.

To predict an observation $M_{i,j}$, one need only calculate $\sum_{k=1}^{K} L_{i,k} \ast R_{k,j}$. The algorithm for fitting $L$ and $R$ is described in Algorithm 4. One could express this as $M = LR$, but recall that the observations are very sparse, both in the training and qualifying sets, and a full matrix multiplication would consume far more space and computational effort than is necessary.

In Funk’s description of Algorithm 4, he described terminating the updates for a given factor when it was “as good as it’s going to get”, which was not defined, but is likely determined based on cross validation against the probe set, i.e. one stops when overfitting begins to occur.
Algorithm 4 Iterative SVD (Funk, [6])

1. Select initialization parameters:
   (a) The number of factors, $K$, to use (e.g. Funk tried $K = 40$)
   (b) A scaling rate, $\lambda$ (Funk recommended $\lambda = 0.001$)
   (c) Values to use for initialization of $L$ and $R$ (Funk used 0.1 for all of the entries)

2. Initialize $L$ and $R$ according to the initialization parameters, identify as $L^{(0)}$ and $R^{(0)}$.

3. Let $M^{(0)} = L^{(0)}R^{(0)}$

4. Calculate $E^{(0)} = M - M^{(0)}$.

5. Produce a set of triplets $(i, j, k)$, combining all ratings $(i, j) \in \mathcal{R}$, and all factors $k \in 1, \ldots, K$, and order the triplets by $k$, then from the oldest to newest ratings (i.e. starting with factor 1, list all ratings by date, then repeat the list for factor 2, and so on).

6. Update via iterative descent: Sort the training entries by factor, then by date, and iteratively update entries of the factorizations matrices, $L$ and $R$, over triplets of (user, movie, factors), indexed by $(i, j, k)$, where $(i, j) \in \mathcal{R}$, as follows:
   (a) Update $L_{i,k}$ as: $L'_{i,k} = L_{i,k} + \lambda E_{i,j}R_{k,j}$.
   (b) Update $R_{k,j}$ as $R'_{k,j} = R_{k,j} + \lambda E_{i,j}L_{i,k}$.
   (c) Stop iterating over updates for the factor when the changes to $L$ and $R$ are below some tolerance, e.g. $\max_{i,j} \{|E_{i,j}R_{k,j}|, |E_{i,j}L_{i,k}|\} \leq \tau$, for some chosen $\tau > 0$.

7. Let $L^{(1)}$ and $R^{(1)}$ be the resulting estimates for $L$ and $R$ after these updates, and recalculate $M^{(1)}$ and $E^{(1)}$.

8. Repeat steps 4 and 5 $t$ times, until a stopping criterion is reached, which may be one of:
   (a) $t = T$, a pre-chosen number of iterations, or
   (b) The change in RMSE, $E^{(t)} - E^{(t-1)}$, is below some tolerance, or
   (c) The RMSE for the held out probe or qualifying set, $Q$, has begun to increase (indicating over-fitting)

Funk also suggests a variation of the algorithm, applying $L_1$ regularization in the update step 6 of Algorithm 4, described in (3.18), where $\nu = 0.02$ was reported to work well; we will refer to this, Algorithm 5, as Regularized SVD.

Regarding the initialization step in Algorithm 4, in [16], members of the team “Gravity” suggested initializing the $L$ and $R$ matrices with random values. In practice, this could be beneficial for generating models based on different initial random values, which might be aggregated (e.g. averaged) in order to potentially produce better predictions than any one decomposition might generate.

Parameters for the “learning rate”, $\lambda$, and regularization values, $\nu$, were explored exten-
\begin{algorithm}
\textbf{Regularized SVD (Funk) - Update equations}
\begin{align}
L_{i,k}' &= L_{i,k} + \lambda E_{i,j} R_{k,j} - \nu L_{i,k} \\
R_{k,j}' &= R_{k,j} + \lambda E_{i,j} L_{i,k} - \nu R_{k,j}
\end{align}
\end{algorithm}

sively by others. In [13], Paterek confirms that the parameters suggested by Funk were very hard to improve, with $\lambda = 0.001$, $\nu = 0.02$, and $K = 96$ factors (or “features”). In [16], Takacs et al., members of “Gravity”, also recommend setting $\lambda$ and $\nu$ to small values.

In [16], Takacs et al. also explored the use of indicator values in the $R$ matrix, such as whether or not particular words, such as “Season”, appear in the title of a DVD. In this way, the $R$ matrix can be augmented with a discretized feature space.

\section*{Other Matrix Factorization Methods}

After the post by “Simon Funk”, other competitors produced many variations on matrix factorization. Indeed, the original Funk posting included variations on the method. We present below a brief summary of the types of variations encountered. The most extensive sources of variations appear in [6, 13, 17].

- Regularized MF (RegMF): In this algorithm, updates are regularized with $L_1$ shrinkage, as, for example: e.g. $L_{i,k}' = L_{i,k} + \lambda * L_{i,k} * R_{k,j} - \nu L_{i,k}$. This was introduced by Funk [6], and described by Paterek [13], as “RSVD1”.

- MF with biases (InterceptMF): Several methods for introducing indicators or intercept terms have been proposed. The most explicit, by Takacs et al. (aka the team “Gravity”) [17], simply sets the first column of $L$ equal to 1, and the second row of $R$ equal to 1.

- Bag-of-Movies (BOM-MF): In [13], Paterek described a different type of matrix factorization method, where a user is considered a “bag of movies”, just as a document might be considered a “bag of words” in an information retrieval context. In this scenario, two movie factor matrices are calculated, say, $P$ and $Q$; for predicting a given $x_{i,j}$, the rating by user $i$ for movie $j$, the method involves comparing $j$ to other movies rated by $i$. This model can be represented as $\hat{x}_{i,j} = c_i + d_j + e_i \sum_{k=1}^{K} p_{k,j} \sum_{m \in J_i} q_{k,m}$, where $J_i$ are the movies rated by user $i$, and $e_i = \frac{1}{|J_i|+1}$, as in the LM model. This approach is interesting in that it calculates predictions over an explicit neighborhood of similar movies. As such, it is a very atypical approach to factorization. One possible shortcoming is that it gives very little insight into subsets of users, as all of the major parameters are movie-based. Several extensions to this model have been developed:

- Paterek [13] also proposed letting $P = Q$ and eliminating the weights, as: $\hat{x}_{i,j} = c_i + d_j + \sum_{k=1}^{K} p_{k,j} \sum_{m \in J_i} p_{k,m}$, which reduces the number of parameters to estimate.

- In [2], Team BellKor’s first Progress Prize paper, Bell and Koren extended these ideas. For each factor in $P$, the score of movie $j$ (i.e. over $k \in 1 : K$, $p_{k,j}$ is extracted), is multiplied by a function of the factor weights in $Q$ for all other movies rated by the user.
user. The formulation is: 
\[
\hat{x}_{i,j} = \sum_{k=1}^{K} P_{k,j} \cdot \sigma(b_k + \sum_{m \in J_i} w\text{score}(i,m)Q_{k,m}),
\]
where \(\sigma\) is the logistic function, and all other elements are to be fitted through gradient descent to minimize the MSE of \(\{\hat{x}_{i,j}\}\). Variations on this include those by Paterek, namely: setting \(P = Q\), setting the weights, \(w\), equal to 1, letting \(w\) be determined by residuals, and using combinations of \(Q\) and \(P\) in the inner function (e.g. replace the inner summation with \(\sum_m Q_{k,m} + \sum_m E_{i,m}P_{k,m}\)), where \(E\) is the matrix of residuals.

In contrast to these methods, we:

- Enforce sparse models even before applying an \(L_1\) penalty, as is done by “RegMF” and its variants, by creating factors that are already quite sparse, unlike the dense initial models created by all other methods.

- Link the structure of the \(L\) and \(R\) matrices: by assigning movies to factors, we define the sparsity structure of \(R\) and \(L\) prior to their initialization.

- Incorporate intercepts by using a fixed “anchor” film for each factor, rather than via arbitrary assignment, as in “InterceptMF” models.

- Incorporate similarities as an integral part of the factor initialization, rather than as part of the update phase (as in the BOM-MF variants) or as a post-processing step, as suggested by the BellKor team in their papers [1, 2, 4].
3.8 Our Methodology

Our principal interest in this research is to reduce overall, or global, prediction error by way of constraining errors associated with local clusters. In particular, we seek to avoid the “spread” of error associated with iteratively fitting models on a sparse data set, especially as we may have reason to assume the orthogonality of various subspaces. The inspiration for our method was a realization that projecting unrelated and disjoint\textsuperscript{18} movies onto the same “factor”, as all other matrix factorization methods do, means that that factor’s use in predictions would be optimized across effectively disjoint subsets. By separating movies into clusters, we seek to relate factors to clusters, and adapt matrix factorization in such a way that orthogonality arises by considering clusters distinct; this incorporates agglomerative clustering into matrix factorization. As we relax constraints on cluster membership, we would increase the density of matrix factors until orthogonality is no longer enforced. This method incorporates a “layering” of clustering methods and factorization, by an additive ensemble of matrix factorizations.

In order to achieve our goal of a new approach to matrix factorization, we re-examined centering and the matrix factorization algorithms described earlier. This section describes our approach as three stages: centering, anchoring of clusters, and factorization. The key addition here is the use of “anchors”, which are films that are representative members of a cluster, and we then define factors as vectors of weights associated with members of corresponding clusters. Although movies are assigned to just one cluster, making clusters disjoint from each other, we may repeat this entire cycle with different optimization parameters, allowing for movies to inhabit multiple clusters (albeit, one per iteration of the cycle). This allows us to leverage strong associations between movies in initial iterations of the algorithm, and weaker associations in later iterations.

We will also demonstrate that these methods provide a tunable relationship between agglomerative clustering and matrix factorization.

3.8.1 Anchoring - Finding representatives of Clusters or Factors

It’s notable that as different movies have different variances and numbers of ratings, and thus different impact on total squared error and overall RMSE, we need not focus on performing well over all movies. In fact, it is easily observed that at most 20% of the movies account for 20% of the total squared error (TSE), or about 10% of the RMSE that we wish to reduce. As some movies are vastly more popular than others, with sufficiently large variance, a method would be very competitive if it could perfectly predict just this smaller set of films. Although this is an extreme perspective, it points to our first step in “anchoring”: we order films by their impact on the total squared error (e.g., we use a movie’s residual $L_2$-norm, after centering with a linear combination of user and movie means as the initial baseline prediction model), and we seek to do best in predicting their ratings and leveraging the information associated with their ratings to predict ratings for similar movies.

\textsuperscript{18}By “disjoint” we mean two movies with no common users in their sets of ratings. If two movies have quite many users and the intersection is very small, we might say that they are effectively disjoint, i.e. they are independent if the intersection count is approximately the expected count from independent Bernoulli trials.
The general approach is that we base our factors on clusters: for each new cluster, we anchor the cluster with a film with the largest total error based on our predictions, and subsequently screen other films for inclusion in the same cluster based on similarity to the anchor film. (Note that we may compare candidates for a given factor with all other films already included in the factor, but we can consider sufficiently strong association with the anchor film to be a surrogate, and address this discrepancy via an ensemble of matrix factorization models.) We initially consider all films “inactive”, and consider them “active” after inclusion in a cluster, at which point a film is removed from consideration and removed from the “inactive” list. After screening all available films for inclusion in a cluster, we create a new cluster with the most “erroneous” movie remaining, and repeat the process.

Our anchoring algorithm is described in detail in Algorithm 6. Upon completion of the anchoring, we can now anchor elements of the $R$ matrix. Let $\delta$ be a small initial value, e.g. $\delta = 0.03$, then we let $R_{k,j} = \delta$ if $j \in (A(k) \cup Z(k))$, and 0 otherwise. Initialization of both $R$ and $L$ is more fully addressed in section 3.8.3.
### Algorithm 6 Anchoring Algorithm

**Inputs**

- **N** - The number of users
- **P** - The number of movies
- **X** - Matrix of Ratings, of dimension $N \times P$.
- **V** - Companion “Indicator” matrix to $X$, also of dimension $N \times P$, with entries from $\{0, 1\}$, where $V_{i,j} = 1$ if and only if user $i$ has rated movie $j$.
- **R** - The set of ratings pairs, i.e. $(i,j) \in R$ if and only if $V_{i,j} = 1$.
- **Thresholds**: $T_{\text{Jaccard}} > 0$, $T_{\text{Correlation}} > 0$, $T_{\text{RatioToMin}} > 0$
- **Stopping criteria** - either: C1: Stop when all movies have been assigned to clusters, or C2: Stop when a maximum number of clusters has been identified.

**Steps**

1. Calculate error matrix $E$. For the initial pass, $E = X$. In general, $E$ is the matrix of residuals that we are attempting to fit.
2. For each column $j$ of $E$, let $E_{i,j} = \frac{1}{|\{(i,j) \in R\}|} \sum_{(i,j) \in R} E_{i,j}$, and $S_j = \sum_{(i,j) \in R} (E_{i,j} - E_{i,j})^2$.
3. Initialize our “inactive” set as $I = \{j : j \in 1, \ldots, P\}$; although it is not necessary, it is helpful if the $I$ list is sorted such that $S_{I(1)} \geq S_{I(2)} \geq \ldots \geq S_{I(P)}$.
4. Let $A(1) = \arg \max_{j \in I} S_j$ [i.e. $A(1) = I(1)$, if $I$ has been sorted by $S$ values]. This will be the index for the movie with the largest RMSE for the error matrix.
5. Update $I \leftarrow I \setminus A(1)$. [$A$ is chosen to represent “Anchor” - this is a list of movies that anchor clusters.]
6. For $j \in I$, calculate $d_{\text{Jaccard}}(V_{A(1)}, V_{j})$.
7. Let $Z(1) = \{j \in I : d_{\text{Jaccard}}(V_{A(1)}, V_{j}) \geq T_{\text{Jaccard}}\}$. This is a list of movies that have passed the Jaccard index threshold.
8. For $j \in Z(1)$, calculate $d_{\text{RatioToMin}}(V_{A(1)}, V_{j})$. Remove from $Z(1)$ those $j$ with $d_{\text{RatioToMin}} < T_{\text{RatioToMin}}$, i.e. we retain only those entries that pass our rudimentary statistical test.
9. For $j \in Z(1)$, calculate $d_{\text{Correlation}}(V_{A(1)}, V_{j})$. Remove from $Z(1)$ those $j$ with $d_{\text{Corr.}} < T_{\text{Corr.}}$, i.e. we retain only those entries that have sufficient correlation with the anchor film.
10. Update our inactive set: $I \leftarrow I \setminus Z(1)$.
11. Repeat Steps 4-9, selecting $A(2)$ from the remaining “inactive” movies, in the list $I$. **Terminate loop** when all movies have been included in some cluster or when a maximum number of clusters is reached.
12. Let $K = \{|k : A(k) \neq \emptyset|\}$. This determines the dimensionality of the factor matrices, $L$ and $R$.
13. Upon completion, we return a set of lists (or a list of lists), $\{Z^*(k) = \{A(k), Z(k)\}\}$, $k = 1, \ldots, K$. The first entry in list $Z^*(k)$ is the index of the anchor movie, $A(k)$, and the subsequent movies in the list are those assigned to the same cluster, i.e. $Z(k)$.

At this point, we have an $R$ matrix whose density pattern is easily understood: each factor represents clusters of movies. Because of our screening criteria, movies in the same factor tend to be watched by the same population of people (due to Jaccard and “RatioToMin” distances), and their ratings tend to be correlated. If we have used other filters, we can add
additional claims to each factor. In addition, we defined the ordering of movies to anchor each cluster based on the anchor film’s contribution to overall mean squared error, and we can now examine the clusters (or factors) from this perspective.

There are two additional variations involving the ordering in Step 2 and the anchoring in Step 9. Instead of weighting by $L_2$ norm in the training set, we may change Step 2 to reweight the columns by their expected total squared error in the qualifying set (i.e., we can estimate this based on each movie’s total number of entries in the qualifying set multiplied by its mean squared error in the training set), in order to focus more aggressively on the qualifying performance rather than the training RMSE. We examined this and believe it could be fruitful, but focused our efforts on mastering the training and probe data sets. Another variation involves changing the anchoring to include movies that could be negatively correlated with the anchor film, i.e., change Step 9 to include films with $|d_{\text{Corr}}(V_{iA(i)}, V_{i,j})| \geq T_{\text{Corr}}$, with appropriate changes in the initialization of the $R$ matrix, as discussed in section 3.8.3. We found this approach quite appealing, though we do not address the results in this chapter. One issue that can arise is that allowing negatively correlated movies in a factor may reduce the number of factors, and distinctions between factors may become less clear; from a clustering perspective, this is a problem that is akin to adding elements into a cluster that are especially distant from the cluster’s “seed” element. It is possible to include this in a later cycle of matrix factorization; we address matrix re-factorization in Section 3.8.4.

### 3.8.2 Centering the Matrix

Now that we have identified representative “anchor” films and assigned related films to their corresponding clusters, we may center the matrix per cluster. This step is optional, though it extends earlier work by introducing cluster-level offsets. Our approach is as follows:

1. Let $\Omega(k) = \{(i, j) : j \in Z^*(k)\}$
2. For $k = 1, \ldots, K$, let $C_k = \frac{1}{|\Omega(k)|} \sum_{(i,j) \in \Omega(k)} X_{i,j}$, i.e., the per-factor mean rating.
3. For $k = 1, \ldots, K$, for $(i, j) \in Z^*(k), (i, j) \in \mathcal{R}$, let $X_{i,j}^{(1)} = X_{i,j} - C_k$.

Throughout this chapter, we use $X^{(t)}$ to indicate the $t$-th iteration matrix that we seek to model; we will address iterations of matrix factorization (which we call “matrix re-factorization”), in Section 3.8.4. Our “centered” matrix is now $X^{(1)}$, and our “centering” matrix is $X^{(0)} = X - X^{(1)}$. We shall see in Section 3.9 that optimal initial centering is to have a maximal number of clusters, i.e., center per movie. However, extensions of this work can involve changing user-centering to incorporate their mean rating per cluster, which would necessitate a smaller number of clusters, otherwise the training data will be perfectly overfitted.

Extensions to the centering method include:

- Per cluster and per user, calculating the user’s mean rating of movies in each $Z^*(k)$, and subtracting this from $X$. We call this the “user perspective” for the cluster. (Note: as a cluster is inherently based on movie-movie relationships, there isn’t a natural movie-perspective of the cluster.)
• Iteratively recentering with the above centering algorithm and the user-perspective centering algorithm.

We only briefly explored these extensions. One of the first issues to arise is that centering should be based on the user support for the cluster. Enthusiasts or fans of a cluster are inclined to rate the movies more highly, as well as rate more movies in a cluster. People with a strong aversion to a cluster may also rate many movies from that cluster, and the most notable such example is a vigorous set of users who have given a 1 to all movies by Michael Moore. In general, though, people unenthused by a particular cluster may rate few movies and give a lower rating to their first exposure to movies in the cluster. In our work, we often see indicators of “gateway” films: if a user likes the gateway film, they are more likely to rate other movies in the same cluster than those who do not enjoy the gateway film\(^9\). As “enthusiasts” or fans of movie clusters tend to give higher ratings, these users, if they become such fans, may similarly give higher ratings to other movies in the same cluster.

### 3.8.3 Initialize and Update L and R Matrices

Our next step is to initialize the $R$ and $L$ matrices, and iteratively update each of them. Because the $K$ factors are based on $K$ distinct clusters, they are non-overlapping and we may be computationally efficient by updating the factors within each matrix in parallel across at least $K$ processors.\(^{20}\)

Based on $Z(k)$, we can now identify the non-zero entries in the $R$ matrix, and initialize them as follows, in Algorithm 7. One variation is to allow the initialization of $R_{k,j}$ to be scaled by the correlation with the anchor film. Also, as mentioned in section 3.9.1, the anchoring algorithm may be modified to allow for negative correlation, thus we may initialize $R$ as $R_{k,j} = \delta \cdot \text{sign}(d_{Corr}(V_{A(i)}, V_{j}))$.

---

\(^9\)We will address gateway films in future work.

\(^{20}\)Note, too, that the matrices can be reordered to produce block diagonal matrices; e.g. $R$ can become $K$ blocks, each of size $1 \times |Z^*(k)|$.\(^{20}\)
**Algorithm 7** Algorithm for initializing $R$ and $L$ matrices

**Inputs**

- $\{Z^*(k)\}$ - The set of movie clusters
- $N$ - The number of users
- $P$ - The number of movies
- $K$ - The number of unique clusters
- $\delta_R$ - An initial value to assign to entries in $R$, e.g. $\delta_R = 0.03$ in some of our experiments.
- $\delta_L$ - An initial value to assign to entries in $L$, e.g. $\delta_L = 0.03$ in some of our experiments.
- $E$ - An error matrix to fit, e.g. $E^{(1)} = X^{(1)} = X - X^{(0)}$.

**Steps**

- Initialize $L = 0_{N \times K}$ and $R = 0_{K \times P}$.
- For $k = 1, \ldots, K$:
  - For $j \in Z^*(k)$
    1. Let $R_{k,j} = \delta_R$
    2. For $\{i : (i,j) \in R\}$, let $L'_{ik} = L_{ik} + \delta_L \cdot E_{ij}$
- Terminate when $k = K$. Return $L$ and $R$.

Algorithm 8 describes our method for holding $R$ or $L$ constant while updating the other - an alternating least squares method for updating the $L$ and $R$ matrices. As stated before, although this loops over the $K$ factors, each can be done in parallel, independent of the others; this is because each movie belongs to only one cluster, and clusters define factors, making factors orthogonal to each other.

A number of variations are feasible for this algorithm. As noted within Algorithm 8, Steps 1 and 2, updates can be either gradient-based or based on fixed step sizes. Although our general inclination is toward fixed step sizes, in order to control the $L_1$ magnitude of the vectors based on the number of steps, we found that controlling the step sizes was a tedious and time consuming effort, and opted to regularize via a multiple, $\lambda$, of the gradient. Our algorithm also incorporates optional Lasso-style penalties.

Another notable consideration is that each of the loop steps, 1 and 2, can be ordered according to the dates of the ratings. As we did not yet investigate dates, we did not modify the loop procedure for this choice.
Algorithm 8 Algorithm for updating $R$ and $L$ matrices

**Inputs**

- $\{Z^*(k)\}$ - The set of movie clusters
- $N$ - The number of users
- $P$ - The number of movies
- $K$ - The number of unique clusters
- $\delta_R$ - A fixed step size for incrementing entries in $R$ (optional)
- $\delta_L$ - A fixed step size for incrementing entries in $L$ (optional)
- $E$ - An error matrix to fit, e.g. $E^{(1)} = X - X^{(0)} = X^{(1)}$
- $\lambda_L, \lambda_R$ - The "learning rate" for $L$ and $R$ updates, respectively; these may be identical.
- $\nu_L, \nu_R$ - The $L_1$ shrinkage penalties; these may be identical and may be set to 0.
- Stopping criterion (e.g. tolerance for updates to $L$ and $R$, maximum number of update iterations, tolerance for changes in RMSE, or maximum magnitude of values in either $L$ or $R$)

**Steps**

1. Fix $R$ matrix
   - For each user $i = 1, \ldots, N$
     - For each $j$ such that $(i, j) \in R$
       * Select $k$ such that $j \in Z^*(k)$
       * Let $L'_{ik} = L_{i,k} + \lambda_L E_{i,j} R_{k,j} - \nu_L \cdot \text{sign}(E_{i,j} R_{k,j}) - \nu_L L_{i,k}$ [Fixed step-size alternative: $L'_{ik} = L_{i,k} + \delta_L \cdot \text{sign}(E_{i,j} R_{k,j}) - \nu_L L_{i,k}$]

2. Fix $L$ matrix
   - For each movie $j = 1, \ldots, P$
     - Select $k$ such that $j \in Z^*(k)$
     - Loop over users $i$ in the set $\{i : (i, j) \in R\}$
       * Let $R'_{k,j} = R_{k,j} + \lambda_R E_{i,j} L_{i,k} - \nu_R \cdot \text{sign}(E_{i,j} L_{i,k} - \nu_R R_{k,j})$ [Fixed step-size alternative: $R'_{k,j} = R_{k,j} + \delta_R \cdot \text{sign}(E_{i,j} L_{i,k} - \nu_R R_{k,j})$

3. Repeat Steps 1 and 2 until stopping criterion is reached. Return $L$ and $R$.

3.8.4 Layering: Matrix Re-factorization of Residuals

After centering and factorization, we are now able to estimate the original matrix $X$. After centering $X$ with $\hat{X}^{(0)}$, we have $X^{(1)} = X - \hat{X}^{(0)}$. We estimate this via matrix factorization, via Algorithms 6, 7, and 8, as $\hat{X}^{(1)} = L^{(1)} R^{(1)}$; thus, our summed estimate is $\hat{X} = X^{(0)} + X^{(1)} = X^{(0)} + L^{(1)} R^{(1)}$. (Note that the initial centering need not use factor-based
centering, as we shall see in Section 3.9.) However, there will still be residuals, and we have not addressed denser factorizations associated with relaxation of our factor selection criteria, i.e. we may have originally chosen small clusters in order to ensure strong association with the anchor film of each cluster. Small clusters correspond to factors that are sparse; however, in order to leverage weaker associations, we would like to allow for greater density. To that end, we can produce a sequence of matrix refactorizations as follows, for some odd number of steps:

- For each step \( t = 2q \), with \( q = 0, \ldots, q_{MAX} \), our matrix \( X^{(t)} \) is a centering operation, for which we need to anchor factors and generate sets \( Z^{*(t+1)}(k) \)

- For step \( t = 2q + 1 \), for \( q = 0, \ldots, q_{MAX} \), matrix \( X^{(t)} \) is factored into matrices \( L^{(t)} \) and \( R^{(t)} \), where the density pattern of \( R^{(t)} \) is specified by \( Z^{*(t)}(k) \).

In this way, we can produce successively denser matrix factorizations: by relaxing constraints for membership in clusters, we produce larger clusters, thereby making each factor denser and responsible for more predictions (because more movies will share factors), while the RMSE is pushed down with each iteration.

At the most “permissive” extreme, with no thresholds for inclusion into a factor, at each pass through the entire algorithm, we simply produce matrices that consist of single vectors, and a sequence of passes produces a standard matrix factorization that is fully dense. In short, the matrix refactorization approach allows us to encapsulate many other matrix factorization algorithms as a special parameter setting for this method.

At the opposite extreme, if we allow only one movie per cluster, the method is the same as movie-based centering. Thus, this method encapsulates a spectrum of sparse to dense models, from centering to matrix factorization.

We may also “reuse” previous steps in the process: we could update the \( L^{(t)} \) and \( R^{(t)} \) matrices from a given step or we could reuse their density patterns and fit new \( L \) and \( R \) matrices. The latter approach may be more useful if we are reluctant to change a key component of a model, but would like to better update changes that occur in a streaming environment.
3.9 Results

In this section, we review the results from the major modeling phases and diagnostics relating to the building of the models and their performance. In Section 3.9.1, we address our factor anchoring step, and the effect of different parameters on the resulting number of factors. In Section 3.9.2, we report the results of our matrix factorization method, including the results of successive layers in the ensemble.

Our approach to matrix factorization had two components: factorization based on sparse factors produced through the anchoring step, and successive “layers” that involve re-fitting the residuals. The use of multiple layers may not be worthwhile, as the models did not improve much with successive layers. We further studied the evolution of the models, iteration-by-iteration and layer-by-layer, across different values for the anchoring parameters. Those results are addressed in Section 3.9.3.

In Section 3.9.4, we introduce the notion of different types of model inadequacy: the problems that arise both in fitting our models to sparse data and in making predictions with such models. In particular, we encounter sparsity in both columns and rows, and may be impacted by either or both types of sparsity. In Section 3.9.5, we illustrate the difference in the levels of support, i.e. the number of ratings, for users and movies in the training and qualifying sets. In particular, the two samples have different distributions of support across the (user,movie) pairs, which can further limit the strength of the models.

3.9.1 Anchoring

The first stage of our process involved anchoring the factors, which form groups of movies, as described in Algorithm 6. The grouping of movies determines the structures of the $R$ and $L$ matrices, which are subsequently initialized. In this section, we report on our work in using fixed values for similarity thresholds when adding movies to a factor, based on their similarity to an anchor film. Our work began by using fixed values, but we encountered far greater sparsity than desired in many factors. Instead, as described below, we opted to use quantiles, based on similarity values obtained per anchor, in order to tune the sparsity and interpretability of factors in a meaningful way, across all of the anchors.

We initially determined parameters for thresholding based on absolute similarity scores, such as a Jaccard index exceeding 0.3, which makes the initial groupings appear reasonable. Recall that the groups or factors are “anchored” by the movies that are as-yet-unassigned to a factor and have the highest total squared error in the training set, among the unassigned movies. In Table 3.9, we report the number of factors produced by using just two screening criteria: the Jaccard index and Pearson correlation.

The results are rather striking: although the intermediate values produce factors that are plausible, i.e. the films in each grouping are reasonably well related in terms of genre, the total number of factors is almost as large as the number of films (17770), indicating that each factor is very sparsely populated. In addition, we observed that anchoring results are sensitive to the choice of initial centering methods: the assignment to factors can vary based on which movie anchors a factor. If the ordering of total squared error is changed slightly, then factors anchored and filled earlier will affect the composition of the candidate list for anchoring later factors, as well as the composition of those later factors. When the screening
Table 3.9: Number of factors generated via the anchoring process, for different Jaccard (columns) and Correlation (rows) threshold values. Notice that the thresholds are increasing as we move down and to the right; the result is that fewer movies exceed the thresholds in terms of similarity with the anchor films. As a result, the factors are sparse, and more factors are needed to cover the full set of movies. The loosest thresholds are the upper left (0.10, 0.10), and the tightest are in the lower right (0.50, 0.50). This is addressed more fully in Section 3.9.1.

<table>
<thead>
<tr>
<th>Jaccard \ Correlation</th>
<th>0.10</th>
<th>0.20</th>
<th>0.25</th>
<th>0.30</th>
<th>0.35</th>
<th>0.40</th>
<th>0.45</th>
<th>0.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.10</td>
<td>7996</td>
<td>8000</td>
<td>8100</td>
<td>8339</td>
<td>8840</td>
<td>9524</td>
<td>10218</td>
<td>10983</td>
</tr>
<tr>
<td>0.20</td>
<td>13464</td>
<td>13479</td>
<td>13525</td>
<td>13648</td>
<td>13890</td>
<td>14179</td>
<td>14458</td>
<td>14738</td>
</tr>
<tr>
<td>0.25</td>
<td>14872</td>
<td>14894</td>
<td>15001</td>
<td>15145</td>
<td>15332</td>
<td>15491</td>
<td>15649</td>
<td>15850</td>
</tr>
<tr>
<td>0.30</td>
<td>15845</td>
<td>15859</td>
<td>15869</td>
<td>15922</td>
<td>16025</td>
<td>16131</td>
<td>16211</td>
<td>16291</td>
</tr>
<tr>
<td>0.35</td>
<td>16485</td>
<td>16480</td>
<td>16493</td>
<td>16533</td>
<td>16587</td>
<td>16637</td>
<td>16684</td>
<td>16722</td>
</tr>
<tr>
<td>0.40</td>
<td>16932</td>
<td>16934</td>
<td>16939</td>
<td>16954</td>
<td>16980</td>
<td>16999</td>
<td>17022</td>
<td>17040</td>
</tr>
<tr>
<td>0.45</td>
<td>17225</td>
<td>17228</td>
<td>17233</td>
<td>17237</td>
<td>17248</td>
<td>17256</td>
<td>17270</td>
<td>17279</td>
</tr>
<tr>
<td>0.50</td>
<td>17448</td>
<td>17449</td>
<td>17451</td>
<td>17452</td>
<td>17454</td>
<td>17457</td>
<td>17460</td>
<td>17464</td>
</tr>
</tbody>
</table>

criteria are made more stringent, the number of factors generally increases: fewer movies are sufficiently similar to the anchor film to be included in the same factor. However, notice the decrease in the number of factors between the entry for (0.35, 0.10) and (0.35, 0.20): there are 5 fewer factors, while every other setting for the Jaccard threshold involves an increase of between 1 and 22 factors for the same change in the correlation threshold. In this case, we assume that one or more films were no longer added to factors and instead anchored their own (very, very slightly) denser factors.

From our work, we conclude that fixed values for the similarity thresholds tend to result in many films that exceed the thresholds for similarity with the first few anchors, i.e. in Algorithm 6, the sets $Z(i)$ are large for small values of $i$, and most subsequent anchors are paired with very few other films. The impact is that the initial factors, e.g. the first hundred factors, are much denser than the subsequent factors, which may have only one or a few members. Furthermore, we were dissatisfied by reducing the similarity thresholds to produce even fewer factors: this tended to produce initial factors that appeared less coherent (e.g. grouping together romantic comedies and intense action films).

As an alternative to using fixed values, we examined the use of quantiles for the similarity scores, allowing us to tune the proportion of movies that could pass the thresholds for anchoring. For instance, if we set the threshold, per anchor film, for Jaccard index and correlation to be the top decile of its respective similarity scores with other movies, and so on for each successive anchor (using the thresholds for similarity with all other movies, not just those remaining in the candidate set), we obtain 5276 factors, as shown in Table 3.10. At the extreme, by relaxing the thresholds to the median similarity scores for Jaccard and correlation, the anchoring process yields only 62 factors.

Although we explored other screening thresholds, we used these two and a third, called $\text{RatioToMin}$, described in Section 3.7.2. As mentioned, increasing the number of screening criteria generally increases the number of factors, as it tends to increase the sparsity per
factor. We found that these three criteria produced reasonable, i.e. topically related, groupings that could be applied across multiple “layers” (i.e. re-anchoring for successively fitting the residuals). As an anchoring method, we were satisfied that this quantile-based screening would produce reasonable factors and capped the number of factors at 500, in order to make the initial factorizations and layers less computationally expensive. After 500 factors, the remaining movies were ignored for a given factorization layer, i.e. the model was not fitted for these movies. As successive layers would relax the thresholds, all movies were assured of entering the factorizations.

### 3.9.2 Prediction Performance

In order to demonstrate the algorithms of this chapter, we worked with the first “mod 10” slice of data (i.e. corresponding to the first row of 3.1), which is 10% of the users and very slightly more than 10% of the training data; for evaluation, we used the corresponding slice of the qualifying data set, aggregating both quiz and probe. This size allowed for exploring many different fitting parameters within a modest amount of time of a few days, rather than limited to the once per day submission limits of the competition.

In fitting the models, we had three stages: a linear model, the first matrix factorization, based on the residuals of the linear model, and then a method for creating successive layers, i.e. re-fitting the residuals of one matrix factorization model with a subsequent matrix factorization model. In each of the matrix factorization passes, we needed to determine the optimal anchoring parameters. In the initialization phase, Algorithm 7, the $\delta_L$ and $\delta_R$ values were set to 0.1. In the updating phase, referring to the updating algorithm, Algorithm 8, we fixed the number of update iterations at 100 (this is our stopping criterion); in addition, we set the values of $\lambda_L$ and $\lambda_R$ at 0.3, and the shrinkage parameters, $\nu_L$ and $\nu_R$ at 0.05.

The first layer was based on the “Linear 2” model, $X_{tij}^{(0)} = \beta_{\text{global}} + \beta_1 \text{userAverage}(i) + \beta_2 \text{movieAverage}(j)$, a model that incorporates a global intercept, unlike the “Linear 1” model.

<table>
<thead>
<tr>
<th>Jaccard quantile \ Correlation quantile</th>
<th>0.90</th>
<th>0.80</th>
<th>0.75</th>
<th>0.70</th>
<th>0.65</th>
<th>0.60</th>
<th>0.55</th>
<th>0.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.90</td>
<td>5276</td>
<td>2489</td>
<td>1730</td>
<td>1223</td>
<td>884</td>
<td>638</td>
<td>430</td>
<td>357</td>
</tr>
<tr>
<td>0.80</td>
<td>3907</td>
<td>1352</td>
<td>903</td>
<td>605</td>
<td>426</td>
<td>293</td>
<td>244</td>
<td>195</td>
</tr>
<tr>
<td>0.75</td>
<td>3410</td>
<td>1084</td>
<td>704</td>
<td>471</td>
<td>324</td>
<td>211</td>
<td>178</td>
<td>140</td>
</tr>
<tr>
<td>0.70</td>
<td>3027</td>
<td>900</td>
<td>563</td>
<td>377</td>
<td>261</td>
<td>169</td>
<td>142</td>
<td>109</td>
</tr>
<tr>
<td>0.65</td>
<td>2739</td>
<td>783</td>
<td>493</td>
<td>324</td>
<td>224</td>
<td>137</td>
<td>109</td>
<td>96</td>
</tr>
<tr>
<td>0.60</td>
<td>2528</td>
<td>727</td>
<td>456</td>
<td>302</td>
<td>198</td>
<td>127</td>
<td>96</td>
<td>87</td>
</tr>
<tr>
<td>0.55</td>
<td>2415</td>
<td>686</td>
<td>433</td>
<td>282</td>
<td>192</td>
<td>109</td>
<td>87</td>
<td>73</td>
</tr>
<tr>
<td>0.50</td>
<td>2332</td>
<td>671</td>
<td>423</td>
<td>264</td>
<td>173</td>
<td>100</td>
<td>76</td>
<td>62</td>
</tr>
</tbody>
</table>

Table 3.10: Number of factors for different Jaccard (columns) and Correlation (rows) threshold quantiles, based on quantiles of similarity values for each anchor film. Notice that as the quantiles are reduced, more films will exceed the threshold, making factors denser and fewer (i.e. fewer factors are needed to cover the entire set of films). Moving down and to the right, we have looser thresholds, and only 62 factors are created. This is addressed more fully in Section 3.9.1.
Between each layer, we re-anchored the data. To choose the final model for each anchor, we used the best performing anchoring parameters, of a set of 16 that were searched, with a different set of 16 parameters for each layer.

Our anchoring criteria are denoted as the “RatioToMin” criterion (simRTM), the Jaccard index criterion (simJM), our “partial chi-squared” test, which we refer to as binomial similarity (simBM), and the Pearson correlation (signCorr)\(^{21}\) of ratings. Further work on analyzing our anchoring parameters is described in detail in Section 3.9.3; for the first two layers of this evaluation, we searched over the same parameters as for the first layer in Section 3.9.3. For the third layer, we chose simRTM from \{0.25, 0.50\}, simJM from \{0.25, 0.50\}, simBM from \{0.70, 0.80\}, and signCorr from \{0.50, 0.65\}.

Results on training and qualifying data sets are presented in Table 3.11.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Train RMSE</th>
<th>Qualifying RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cinematic</td>
<td>0.9525</td>
<td></td>
</tr>
<tr>
<td>Centering with global mean</td>
<td>1.085</td>
<td>1.1306</td>
</tr>
<tr>
<td>Linear 1: OLS with user &amp; movie means, no intercept</td>
<td>0.9604</td>
<td>1.016</td>
</tr>
<tr>
<td>Linear 2: OLS with user &amp; movie means, with intercept</td>
<td>0.9240</td>
<td>0.9902</td>
</tr>
<tr>
<td>Anchored MF - 1 layer</td>
<td>0.9171</td>
<td>0.9895</td>
</tr>
<tr>
<td>Anchored MF - 2 layers</td>
<td>0.8189</td>
<td>0.9747</td>
</tr>
<tr>
<td>Anchored MF - 3 layers</td>
<td>0.7704</td>
<td>0.9734</td>
</tr>
</tbody>
</table>

Table 3.11: Results for several algorithms, described in Section 3.9.2.

What is most notable about these results is that the improvements in the test set became negligible after just two layers. This was both contrary to our expectations and time-consuming to investigate, due to the complexity of the method\(^{22}\). Further investigation revealed that several issues affected this result; we will discuss these further in the Section 3.10. The two primary issues were fundamentally methodological issues relating to data sparseness and model sparseness:

1. The decrease in total squared error varied among users based on their support within a factor (i.e. how many movies they had seen). The same was true for movies: the level of support affected the change in TSE with each update.

2. Due to the sparsity of the models, some \((user, movie)\) pairs could not be fitted well. This is addressed in more detail in Section 3.9.4.

Although most of the computations were quite fast, we also discovered that our most important computational cost was not floating point operations, e.g. addition, multiplication,\(^{21}\)This is called “signCorr” (signed correlation) because we also considered unsigned correlation, which we called “absCorr” (absolute correlation), in order to aggregate movies with strong opposing ratings patterns, but early investigations indicated that that aggregation produced unfavorable factorizations.

\(^{22}\)As a result of this, we are not enthusiastic about programming intricate algorithms in languages without good development tools. The size of the data set certainly affected computational speed, but the time to investigate implementation issues is related to the complexity of code, proficiency of the programmer, and the tools available for software development.
etc., but memory access: looking up entries in the \( L \) and \( R \) matrices consumed a greater proportion of the time than did multiplication of the values obtained. We exerted considerable effort to substantially re-implement the algorithms in order to improve the efficiency of memory “lookup” costs, but further work is necessary in order to produce the much larger number of layers that would be necessary if the factors were to become much denser on each iteration. We will address this in Section 3.10.

3.9.3 Layer-by-Layer Analysis

In Table 3.11, we reported the results per layer, with selections based on the best model candidate after 100 update iterations. Due to our earlier implementation decisions, we faced constraints that limited our model analysis to just a few models, namely those at the final stage of updates. The two biggest constraints were: storage of the intermediate \( L \) and \( R \) matrices, and multiplication of these matrices in order to make predictions. In order to examine the behavior of the evaluations at many iterations, we needed to substantially rewrite our data structures and implementation. We also took the opportunity to try different modeling parameters, to examine whether we could improve upon the prior results.

We summarize each layer below, reporting on the anchoring thresholds used for candidate factorization models for each layer, the performance of those models with each update step, and comments on the selection of the best model for each layer. For all of the layers, we limited the maximum number of factors to 500, and used up to all 4 of our similarity screening criteria. The similarity screening thresholds below represent an “acceptance quantile” \((a_{q_{sim}})\): the threshold for similarity with a given anchor, \( T_{sim} \), is determined from the quantile \( a_{q_{sim}} \). For example, if we use the threshold \( simRTM = 0.67 \), then we set the “RatioToMin” threshold for a given anchor at the 0.67 quantile of the similarity values calculated for that movie with all other movies. An acceptance quantile of 0 indicates that that similarity index is not used to screen for related movies. As we used a machine with 16 cores, we opted to allow 2 values, high and low, when 4 criteria were used. In later layers, as we discovered that some criteria seem to have little effect, we removed those from screening.

For each layer, we offer a few brief comments. At a higher level, our interest was in smaller clusters, with closer association to the anchor film, for the factors in the initial layers, and in fewer, denser factors for the later layers, where the clusterings are less bound to the anchor film. Unlike the models of 3.9.2, the selection criterion for selecting the best model for each layer was not the lowest RMSE after the maximum number of iterations; instead, we used an “early stopping” criterion: for each candidate model, its optimal stopping point was defined as the iteration with the greatest decrease in the RMSE relative to the prior iteration, and we chose the candidate with the lowest RMSE at its respective stopping point. In other experiments, we found that this early stopping method seemed to give better layering results on the qualifying set than by using a fixed maximum number of iterations. Our interest in this subsection is to explore the relationship between changes in the screening parameters and the performance of different candidate models over a series of updates; we think that

\[ a_{f_{sim}} = 1 - a_{q_{sim}} \]

\[ a_{f_{sim}} = 0.67 \] is the same as \( a_{f_{sim}} = 0.33 \), but the terms differ based on whether the threshold is thought of as arising from similarity values or as arising from the proportion of films that should pass the threshold (i.e. acceptance fraction).
Table 3.12: Factor anchoring parameters for layers 1 (left) and 2 (right).

(a) Layer 1 anchoring parameters.  
(b) Layer 2 anchoring parameters.

<table>
<thead>
<tr>
<th>Model #</th>
<th>simRTM</th>
<th>simJM</th>
<th>simBM</th>
<th>signCorr</th>
<th>Model #</th>
<th>simRTM</th>
<th>simJM</th>
<th>simBM</th>
<th>signCorr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.67</td>
<td>0.67</td>
<td>0.9</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>0.75</td>
<td>0.8</td>
<td>0.65</td>
</tr>
<tr>
<td>2</td>
<td>0.34</td>
<td>0.67</td>
<td>0.9</td>
<td>0.75</td>
<td>2</td>
<td>0.25</td>
<td>0.75</td>
<td>0.8</td>
<td>0.65</td>
</tr>
<tr>
<td>3</td>
<td>0.67</td>
<td>0.34</td>
<td>0.9</td>
<td>0.75</td>
<td>3</td>
<td>0.75</td>
<td>0.25</td>
<td>0.8</td>
<td>0.65</td>
</tr>
<tr>
<td>4</td>
<td>0.34</td>
<td>0.34</td>
<td>0.9</td>
<td>0.75</td>
<td>4</td>
<td>0.25</td>
<td>0.25</td>
<td>0.8</td>
<td>0.65</td>
</tr>
<tr>
<td>5</td>
<td>0.67</td>
<td>0.67</td>
<td>0.8</td>
<td>0.75</td>
<td>5</td>
<td>0.75</td>
<td>0.75</td>
<td>0.7</td>
<td>0.65</td>
</tr>
<tr>
<td>6</td>
<td>0.34</td>
<td>0.67</td>
<td>0.8</td>
<td>0.75</td>
<td>6</td>
<td>0.25</td>
<td>0.75</td>
<td>0.7</td>
<td>0.65</td>
</tr>
<tr>
<td>7</td>
<td>0.67</td>
<td>0.34</td>
<td>0.8</td>
<td>0.75</td>
<td>7</td>
<td>0.75</td>
<td>0.25</td>
<td>0.7</td>
<td>0.65</td>
</tr>
<tr>
<td>8</td>
<td>0.34</td>
<td>0.34</td>
<td>0.8</td>
<td>0.75</td>
<td>8</td>
<td>0.25</td>
<td>0.25</td>
<td>0.7</td>
<td>0.65</td>
</tr>
<tr>
<td>9</td>
<td>0.67</td>
<td>0.67</td>
<td>0.9</td>
<td>0.65</td>
<td>9</td>
<td>0.75</td>
<td>0.75</td>
<td>0.8</td>
<td>0.55</td>
</tr>
<tr>
<td>10</td>
<td>0.34</td>
<td>0.67</td>
<td>0.9</td>
<td>0.65</td>
<td>10</td>
<td>0.25</td>
<td>0.75</td>
<td>0.8</td>
<td>0.55</td>
</tr>
<tr>
<td>11</td>
<td>0.67</td>
<td>0.34</td>
<td>0.9</td>
<td>0.65</td>
<td>11</td>
<td>0.75</td>
<td>0.25</td>
<td>0.8</td>
<td>0.55</td>
</tr>
<tr>
<td>12</td>
<td>0.34</td>
<td>0.34</td>
<td>0.9</td>
<td>0.65</td>
<td>12</td>
<td>0.25</td>
<td>0.25</td>
<td>0.8</td>
<td>0.55</td>
</tr>
<tr>
<td>13</td>
<td>0.67</td>
<td>0.67</td>
<td>0.8</td>
<td>0.65</td>
<td>13</td>
<td>0.75</td>
<td>0.75</td>
<td>0.7</td>
<td>0.55</td>
</tr>
<tr>
<td>14</td>
<td>0.34</td>
<td>0.67</td>
<td>0.8</td>
<td>0.65</td>
<td>14</td>
<td>0.25</td>
<td>0.75</td>
<td>0.7</td>
<td>0.55</td>
</tr>
<tr>
<td>15</td>
<td>0.67</td>
<td>0.34</td>
<td>0.8</td>
<td>0.65</td>
<td>15</td>
<td>0.75</td>
<td>0.25</td>
<td>0.7</td>
<td>0.55</td>
</tr>
<tr>
<td>16</td>
<td>0.34</td>
<td>0.34</td>
<td>0.8</td>
<td>0.65</td>
<td>16</td>
<td>0.25</td>
<td>0.25</td>
<td>0.7</td>
<td>0.55</td>
</tr>
</tbody>
</table>

Further research should focus on better factorization methods, rather than the choice of iterations for stopping the updates.

The results of this section are for insights, rather than competitive purposes, as we have evaluated many candidate models for each layer against the qualifying set, which would have been difficult if not infeasible given the rules of the competition. Nonetheless, our tendency for successive layers was to substantially decrease the acceptance quantile, perhaps more rapidly than we expected in our layering model. This tends to support the idea that the initial layers may be overly sparse.

Layer 1

In manual testing, we found that higher thresholds for simBM and signCorr appeared to give interpretable factors. For simRTM and simJM, we tested both low acceptance quantile thresholds (0.34) and high quantile thresholds (0.67). In Figure 3.2, we can see that the RMSE drops from above 0.988 to below 0.976 for the best models, or a reduction of more than 0.0120 in the RMSE, or 120 “basis points”.

\footnote{We would like to further caution the reader about perceiving dramatic slopes in the decrease of RMSE in some plots, by pointing out that the baseline for each plot is still quite high: above 0.9700 for the RMSE in every case. The key issues are that differences in parameters lead to quite different trends, but we have yet to pin down the type of model sparsity or density that yields competitive results.}

\footnote{Netflix allowed one submission per day, per team. Some skirted this by being members of multiple teams. In addition, our evaluations are calculated to more than the 4 digits reported by Netflix, which would not have allowed us to produce the smooth trends shown in plots shown below.}
Layer 2

For layer 2, we increased the range of values for simRTM and simJM, and decreased the acceptance quantile for simBM and signCorr, as it seemed that these tended to produce slightly better models in the first layer. The result was that we had greater segregation in the models’ performance for this layer, though the change was on the order of about 0.0050 for the RMSE, or 50 “basis points”. This is less than half of the change in the first layer, and concerned us that we may need quite many layers to reach our goal.

Layer 3

With this iteration, the acceptance thresholds were generally moved even lower; simBM did not seem to have much effect on results in the previous layer, so we have widened the range of values for it, as well. The result was a reduction of about 0.0020 in the RMSE, which is rather low relative to prior layers and to the effort in adding an additional layer.

Layer 4

For this layer, we have moved the acceptance quantiles to the lowest values seen yet, for all but simBM, with values of 0.1 and 0.5 for the other criteria. The performance results are quite peculiar, with increasing RMSE for a number of iterations for some of the models. The reduction in RMSE was again about 0.0020. We suspect that the results for the fourth layer (right-hand plot in Figure 3.3) are likely the result of successive iterations causing some overfitting for some subgroups and better fitting for others, with the rise and fall of RMSE a result of the balance between these two over/better-fitting regions. In addition, we decided to perform 200 update iterations, as our best models seemed to occur later in the process.
Table 3.13: Factor anchoring parameters for layers 3 (left) and 4 (right).

<table>
<thead>
<tr>
<th>Model #</th>
<th>simRTM</th>
<th>simJM</th>
<th>simBM</th>
<th>signCorr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4</td>
<td>0.4</td>
<td>0.75</td>
<td>0.50</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>0.4</td>
<td>0.75</td>
<td>0.50</td>
</tr>
<tr>
<td>3</td>
<td>0.4</td>
<td>0.2</td>
<td>0.75</td>
<td>0.50</td>
</tr>
<tr>
<td>4</td>
<td>0.2</td>
<td>0.2</td>
<td>0.75</td>
<td>0.50</td>
</tr>
<tr>
<td>5</td>
<td>0.4</td>
<td>0.4</td>
<td>0.25</td>
<td>0.50</td>
</tr>
<tr>
<td>6</td>
<td>0.2</td>
<td>0.4</td>
<td>0.25</td>
<td>0.50</td>
</tr>
<tr>
<td>7</td>
<td>0.4</td>
<td>0.2</td>
<td>0.25</td>
<td>0.50</td>
</tr>
<tr>
<td>8</td>
<td>0.2</td>
<td>0.2</td>
<td>0.25</td>
<td>0.50</td>
</tr>
<tr>
<td>9</td>
<td>0.4</td>
<td>0.4</td>
<td>0.75</td>
<td>0.25</td>
</tr>
<tr>
<td>10</td>
<td>0.2</td>
<td>0.4</td>
<td>0.75</td>
<td>0.25</td>
</tr>
<tr>
<td>11</td>
<td>0.4</td>
<td>0.2</td>
<td>0.75</td>
<td>0.25</td>
</tr>
<tr>
<td>12</td>
<td>0.2</td>
<td>0.2</td>
<td>0.75</td>
<td>0.25</td>
</tr>
<tr>
<td>13</td>
<td>0.4</td>
<td>0.4</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>14</td>
<td>0.2</td>
<td>0.4</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>15</td>
<td>0.4</td>
<td>0.2</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>16</td>
<td>0.2</td>
<td>0.2</td>
<td>0.25</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Model # | simRTM | simJM | simBM | signCorr |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
<td>0.9</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>0.1</td>
<td>0.5</td>
<td>0.9</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>0.1</td>
<td>0.9</td>
<td>0.5</td>
</tr>
<tr>
<td>4</td>
<td>0.1</td>
<td>0.1</td>
<td>0.9</td>
<td>0.5</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>6</td>
<td>0.1</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>7</td>
<td>0.5</td>
<td>0.1</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>8</td>
<td>0.1</td>
<td>0.1</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>9</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>10</td>
<td>0.1</td>
<td>0.5</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>11</td>
<td>0.5</td>
<td>0.1</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>12</td>
<td>0.1</td>
<td>0.1</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>13</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>14</td>
<td>0.1</td>
<td>0.5</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>15</td>
<td>0.5</td>
<td>0.1</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>16</td>
<td>0.1</td>
<td>0.1</td>
<td>0.5</td>
<td>0.1</td>
</tr>
</tbody>
</table>

(a) Layer 3 anchoring parameters.  
(b) Layer 4 anchoring parameters.

Figure 3.3: RMSE versus iteration, for model candidates for layers 3 (left) and 4 (right).
Layer 5

For layer 5, we have relaxed one of the screens, by letting simJM = 0, and have returned to higher and lower values for simBM and signCorr. We also accept most matches based on simRTM.

Layer 6

Finally, with layer 6, we have also decided to do no screening based on simRTM, i.e. let it equal 0, and have instead tested a larger range of values for signCorr. As this will be our last layer, we have decided to use 400 update iterations.
3.9.4 Inadequacy of Sparse Models for Sparse Data

As seen in our earlier work in Chapter 1, sparse models can have a unique inadequacy: for sparse test data, sparse models may not be able to make a useful prediction when they have no support (i.e. non-zero values) in the dimensions that are non-zero in the test data. In the case of matrix factorization, sparse $L$ and $R$ matrices, when multiplied, may have regions in the product matrix that are 0. However, we may want to make predictions for (user, movie) pairs that overlap with those sparse regions. This presents a problem, which we need to carefully examine.

When evaluating sparse factorizations with the qualifying data, we have identified three cases for (user, movie) pairs in the test set, which we describe below. Note that we use $(i,j,k)$ to refer to the user index, the movie index, and the index of the factor to which the movie is assigned.

1. Model adequacy (or $L$, $R$ adequacy): Entries in the $L$ and $R$ matrix are non-zero for $(i,k)$ and $(k,j)$, respectively. In this case, the user has already rated other movies in the same factor.

2. $R$-inadequacy: The movie has not been assigned to any factor, i.e. $(k,j)$ is 0 for rows in column $j$ of the $R$ matrix.

3. $L$-inadequacy: A non-zero entry exists in $R$ for $(k,j)$, though the entry in $L$ for $(i,k)$ is 0. In this case, the user has not rated other movies from the same factor, so this movie is the first rating within this factor by this user.

For the first case, the situation is optimal, nothing needs to be done. The second case, of a movie that has not been factored, is unpleasant, but not entirely unique to our method: as the inventory of movies grows, new additions will suffer the same problem; the same is true for new users and the third case. Nonetheless, our approach is more likely to have this problem than an approach with fully dense $L$ and $R$ factors. We can ameliorate the second and third situations in a number of ways already addressed: group movies in appropriate ways (i.e. assign all movies to factors, where one is a “catch-all” not based on an anchor film); use multiple layers, with some that have very few, very dense factors, so that users and movies are likely to inhabit at least one shared factor, in at least one layer; and via other means, such as ensembles with “standard” dense methods.

3.9.5 Differences in (User, Movie) Support Quantiles

In future work, we intend to explore the relationship between performance of the models and both their sparsity (or density) and the support for both users and movies, as we suspect that this is where we may pursue better modeling methodologies and understand the differences between our methods and denser methods.

To examine the relationship between the support (i.e. # of ratings) per user and per movie, we first mapped the users and movies to their quantiles of support in the training set. This mapping allows us to represent all of the points on a $(0,1) \times (0,1)$ rectangle; the density of the $(user,movie)$ points can be examined by using hexagonal binning, as illustrated in the plots in Figure 3.5, showing the frequency of these pairs in the training and qualifying sets.
Figure 3.5: Quantiles of support in the training set, for (user, movie) pairs in the training and qualifying sets.

In the case of the training set, the greatest intensity in the plot is for the users with many ratings who rated movies with many ratings, i.e. the dark region in the upper right - this indicates that a significant portion of training ratings arise from frequent raters who rated popular movies. In the test set, the darkest region is nearly constant across the users, i.e. the ratings were concentrated across users, for movies with higher support. This indicates that movies that were popular in the training set are popular in the test set, independent of the users’ support quantile. Thus, there is a mismatch between the two sets, training and testing, regarding the weights of the users across support quantiles. Further, the region of greatest density is slightly decreasing as one looks from left to right, indicating that the higher support users rated movies that had lower support than did users with lower support. This pattern suggests that in the training set, frequent raters tended to rate more popular movies than they did in the qualifying set, where they may rate slightly more esoteric movies than did the users who had fewer ratings.

These tendencies are quite difficult to capture in our current framework, as temporal factors are ignored in our method. Thus, in addition to examining the density of ratings among users or movies, and the density of factors, we believe that further work will require a better approach to modeling changes over time, which our method does not address.

### 3.10 Discussion

#### 3.10.1 Results

For the first two layers, the cumulative RMSE reduction is between 0.015 and 0.02; given that the target reduction is about 0.10, we had hoped that the method could work well in
just a few layers. Once we modeled additional layers, we found it surprising to see that subsequent layers do not appear to have the same effect - the RMSE improvements taper quickly to about 0.0005 to 0.0020, or about 1/100th the improvements of the first two layers.

Thus far, we have not conclusively identified why this occurs, and will explore this in future work. Preliminary analyses suggest that the support of the users (i.e. the number of ratings by a given user) and the movies (i.e. the number of ratings per movie) plays a role in the improvements to the RMSE for each. This has been demonstrated in other work, such as [9], yet we know of no work that has investigated the relationship between a user or movie’s support and the effect on improvements in RMSE with each update or iteration.\footnote{In practice, this could be a resource-intensive task, due to the number of intermediate models that would need to be retained.} Although model diagnostics are beyond the scope of this particular work, we intend to explore diagnostics more fully in later research.

3.10.2 Sample Size

In the work for this chapter, we used the full data set to create the similarity scores and to do anchoring for the first layer, and a 10\% subsample for all of the modeling and evaluation, along with anchoring for the subsequent layers. Due to our method of sampling, we believe similar results would be achieved if we used the entire data set, or perhaps better results due to better anchoring in the subsequent layers.

Although we were not constrained for RAM or hard drive space, the slowest part of our calculations was the time that it takes to look up values in memory or to store updated values. If we wished to do work with the full data set, we would re-implement much of the algorithm in C++, especially the inner loop of the updating step.

3.10.3 Algorithmic Extensions

Finally, our work with sparse models can be extended to matrix factorization for dense data. In spite of our concerns for the performance of the models, which we will seek to improve through a better mix of dense and sparse models, we believe that further work, with denser models and with sparse models applied to dense data, could lead to better insights on how these methods may be applied and extended.

To apply our methods to dense data, very little needs to be changed: our only modeling dependence on sparsity is in the anchoring phase, with the use of the Jaccard index and our “Ratio-to-Min” calculation. We can ignore these filters, and include others, such as $L_2$ and $L_1$ distances, rank correlation, or many other choices. Alternatively, we could sparsify dense data in several ways, usually via some form of stratification:

- Stratify and identify strata with indicator values (i.e. embed the data into a slightly higher dimensional space, by partitioning selected dimensions)
- Squash values that are “close” to 0, after re-centering.
- Stratify to only consider outliers.
- Project into sparse subspaces via Random Linear Projections, and sparsify as above or else use sparse projections.

We also found that the contents of factors were both interpretable in their own right, as related films, due to the screening methods in place. We believe this kind of interpretability has significant utility in real world usage of a recommendation system, as methods like the Jaccard index can be used to suggest why a particular recommendation was made (e.g. “Because you rated ...”), and correlation may be used for other interactions (e.g. “Because you liked ...”).

We believe that a great deal more work is ahead in the field of ensembles. We have explored this in separate work, and found that the clusters can be informative in combination, and that the combination of factorizations (e.g. from different methods) will be a very fruitful area for further research.

3.10.4 Computational Considerations

In developing this algorithm, we primarily sought to develop the statistical methodologies involved in matrix factorization; however, we also found that there were computational benefits and costs associated with the algorithm and its implementation. As we have implemented and applied this algorithm several times, in Matlab and R, across several different types of computers, we believe that the following guidance will help in future implementations of similar algorithms:

1. Dividing the work into stages that could be easily parallelized was very helpful. Parameter searching is an obvious example, but updating the $R$ and $L$ matrices can be done in parallel, as movies were members of only one factor per layer. Most steps in the algorithm, with the exception of anchoring, were parallelizable or distributable, and we made use of this to speed up our computations. This approach makes scaling of the implementation easier, allowing for the use of large numbers of computers; in addition, this supports the ability to handle a large numbers of factors. We believe also that the modeling of the $R$ matrix could be done in parallel across multiple slices of the set of users. This may confer additional benefits via bagging or other ensembling methods, though we did not investigate them in this work.

2. Environments such as R and Matlab offer excellent support for most numerical operations, especially those involving dense matrices. However, it is important to utilize optimized linear algebra libraries, such as BLAS, that are especially efficient in their implementation of fundamental linear algebra functions, as well as tailored to optimize performance on the types of CPUs used to execute one’s code.

3. For sparse matrices, memory access, rather than floating point operations, may be a rate determining step. To that end, it is important to have extremely efficient usage of

\footnote{Note that we do not advocate the same algorithms due to the accuracy issues we have discovered, especially the issues of excessive sparsity. Instead, most of this guidance is directed toward \textit{computationally efficient} implementations of a chosen modeling method, rather than toward the accuracy of the method: these insights are relevant to other methods than ours. The learning curve addressed here is the reader’s or programmer’s, not the algorithm’s.}
the memory caches. Unfortunately, abstract high level languages such as Matlab, R, Python, and Java do not provide much direct support for efficient memory access and usage. Instead, many of the packages that we found most useful were in C or C++, and, should we re-implement this or a related algorithm, we would do so in C++.

4. The methodological research of this chapter was conducted during the Netflix competition, though the final implementation occurred afterward, as implementation was more difficult than anticipated.\textsuperscript{28} At the outset of the competition, the best choice languages for statistical and matrix modeling were R and Matlab, both of which had severe limitations that were made clear by working on this project\textsuperscript{29}. Since that time, Python has, in our opinion, eclipsed Matlab for a number of reasons. The development of numerical computing libraries, most notably NumPy and extensions in SciPy, has made available a large set of fast and extensive tools for linear algebra. The development of strong integrated development environments (IDEs) makes the development of new code much easier. Finally, many other contributions, such as packages for machine learning (e.g. scikit-learn), and the user community (e.g. there appears to be a very large number of Python users who are Matlab refugees), are all important considerations. Beyond these is the issue of cost: open source environments such as R, and most languages, are easily and freely deployed on many computers, such as Amazon’s EC2 cloud computing infrastructure, unlike a proprietary system such as Matlab. We found that it was essentially impossible to deploy Matlab on EC2 in an “on-demand” manner. While there are alternative options, such as compiled Matlab code or a stripped-down open source alternative, such as Octave, these were inadequate and incomparable for our needs. Just as we did not use Python for its limitations at a given time, our initial choice to use Matlab was a result of balancing ease of implementation against its idiosyncratic limitations, and our later migration to R was a result of evolving needs and resources. At this point in time, we would not re-implement anything in Matlab or Perl, and would recommend Python instead.\textsuperscript{30} We remain proponents of R, but have discovered speed limitations common to all of these languages (i.e. Matlab, R, and Python). For that reason, we do not recommend solely using a higher-level language, such as R, Python, or Matlab, for such intense computing purposes.

5. The inner loops of all such code need to be as fast as possible. While R, Python, and Matlab are excellent for research and exploration, we would not recommend implementation of the inner loops of our algorithm in any of these languages. Instead, we would recommend C++, for both the previously-mentioned improvements in memory

\textsuperscript{28}Although this matrix factorization work was our primary research emphasis during much of the competition, the author’s work with The Ensemble team primarily involved other modeling methods as well as work on building ensembles of models.

\textsuperscript{29}As an example, R lacked easy-to-use functionality for mapping large matrices directly to disk – a functionality known as memory mapped files. However, the lack of such functionality inspired others working on the competition to develop it, resulting in the “bigmemory” package by Kane and Emerson [8].

\textsuperscript{30}In many ways, some of this functionality is very competitive with the options in R, as well. The key differentiator between R and Python is the extent to which the former supports statistical analyses and visualization, along with an extensive community focused on such usage. In almost every other way, Python has functionality comparable, or superior, to similar functionality in R, primarily due to its much larger base of developers and users.
access (as mentioned in our second point, above), as well as other reasons, including reusability and speed: such code can be easily re-used from within other environments (i.e. Matlab, R, Python, etc. can all pass data to and from C++ code), and there are numerous high performance code libraries and easy tools for profiling (we have made extensive use of R and Matlab code profilers; the former is painful, the latter is tolerable, though weak in comparison to the options for C and C++).

3.11 Conclusion

In this work on matrix factorization, we have extended our work in regularized linear models into matrix decomposition, by constructing the decomposition as a set of linear projections; in addition, by clustering the column-space, i.e. movies, we have separated the columns into disjoint spaces, so that we could create sparser decompositions.

Our melding of similarity distances and matrix factorization is novel. Although we were disappointed with the results due to extreme sparsity, we look forward to further research on tunable sparse decompositions. As we can eliminate our sparsity constraints and obtain approaches similar to others, we believe that more work can be done to determine the optimal balance between sparse factors with tight similarity constraints and denser factors that do not have such strong relationships between elements.

We have also explored several methods for speeding up the computations involved in the algorithm and some of the underlying methods. Unlike traditional similarity calculations or implementations of such, we have taken different approaches in order to make the similarity screening faster and more efficient, as constraints on memory and time required faster calculations. Traditionally, statistics evolved (and evolves) to answer various questions, with particular assumptions and constraints. In this work, on the basis of novel assumptions and constraints, we have explored new statistical methodologies. On their own, the results are not competitive with the denser approaches to matrix factorization, though we look forward to exploring how we may combine both dense and sparse decompositions for better results than either alone.

We have also introduced a simple way to allow for meaningful subsampling, by reordering the data by support. We did not find this in the literature, but it was very helpful for our own research process. Because the method allows for easier parallelization of modeling, we also look forward to using this sampling methodology in computationally distributed implementations of ensembles of models.
Bibliography


