Flexible and Efficient Exploration of Rated Datasets

by

Naresh Kumar Kolloju

B.Sc.,Indian Institute of Technology, Bombay, 2009

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

March 2013

Abstract

As users increasingly rely on collaborative rating sites to achieve mundane tasks such as purchasing a product or renting a movie, they are facing the data deluge of ratings and reviews. Traditionally, the exploration of rated data sets has been enabled by rating averages that allow user-centric, item-centric and top-k exploration. More specifically, canned queries on user demographics aggregate opinion for an item or a collection of items such as *18-29 year old males in CA* rated the movie `The Social Network` at 8.2 on average. Combining ratings, demographics, and item attributes is a powerful exploration mechanism that allows operations such as comparing the opinion of the same users for two items, comparing two groups of users on their opinion for a given class of items, and finding a group whose rating distribution is nearly unanimous for an item. To enable those operations, it is necessary to (i) adopt the right measure to compare ratings, and to (ii) develop efficient algorithms to find relevant <user,item,rating> groups. We argue that rating average is a weak measure for capturing such comparisons. We propose contrasting and querying rating distributions, instead, using the Earth Mover's Distance (`EMD`), a measure that intuitively reflects the amount of work needed to transform one distribution into another. We show that the problem of finding groups matching given rating distributions is NP-hard under different settings and develop appropriate heuristics. Our experiments on real and synthetic datasets validate the utility of our approach and demonstrate the scalability of our algorithms.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank my supervisor, Professor Laks V S Lakshmanan, for his great help and guidance through out this thesis. I am indebted to him for his patient guidance, insightful comments on my research and his support during the past few years.
I am also grateful to Ruben H Zamar (Professor at UBC statistics Department) and Sihem Amer Yahia (Director of research at CNRS), who worked with me throught out this thesis. Most ideas of this thesis came from the discussion among Laks, Sihem, Ruben and me. I feel lucky to have worked under guidance of these people.
I owe thanks to all members of the Database Management and Mining lab, and also Holly Kwan, Joyce Poon for making my study at UBC fun and memorable.

Chapter 1

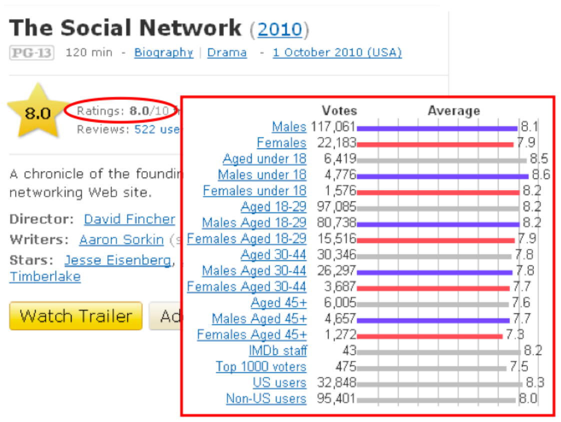
Introduction

1.1 Motivation

In collaborative rating systems such as Flixster, Epinions, and Yelp, users, in their role as producers, rate items they have experienced. These ratings form the basis for users, in their role as consumers, to explore items in order to find what they may be interested in, or to receive recommendations. In this paper, we focus on the kind of support a system needs to provide in order to make *the exploration of rated datasets* effective and useful. We argue that the ability to compare *rating distributions* of groups of ⟨user,item,rating⟩ tuples is central to such exploration, as it enables choosing between different rater populations, contrasting items with respect to the opinion of their raters, finding groups of ⟨user,item,rating⟩ tuples that nearly agree on their ratings for items in the group, etc. To achieve that, we develop a framework to compare and find such groups given conditions on users, items and ratings.

In the physical world, asking one's friends or acquaintances to compare sets of items, e.g., French and American comedies, Canon and Nikon cameras, is common practice. The same could be said of comparing the opinion of two users, Mary and John, or user sets, e.g., males and females, on a specific item or a class of items such as moderately priced Chinese restaurants in the Bay Area. Online, the most common way to do that is by contrasting rating averages. Averages are computed on ⟨user,item,rating⟩ groups that include one or multiple items, e.g., French Comedies, and where the user dimension represents all or some users, such as movie critics or 18-29 year old's. Figure 1.1(a) illustrates an example of a demographics breakdown on IMDb for the movie `The Social Network`. The example shows a global average as well as averages for canned ⟨user,item,rating⟩ groups where item refers to a single movie.

Several studies have established that users in different demographics, i.e., different rater populations, may have different preferences for the same item [6]. Table 1.1 shows average ratings for two different movies by two rater populations – those living in NYC and those living in Nashville. One can easily see that *average rating for a given population does not necessarily*



The Social Network (2010)

PG-13 120 min - Biography | Drama - 1 October 2010 (USA)

8.0 Ratings: 8.0/10

Reviews: 522 use

A chronicle of the foundi
networking Web site.

Director: David Fincher

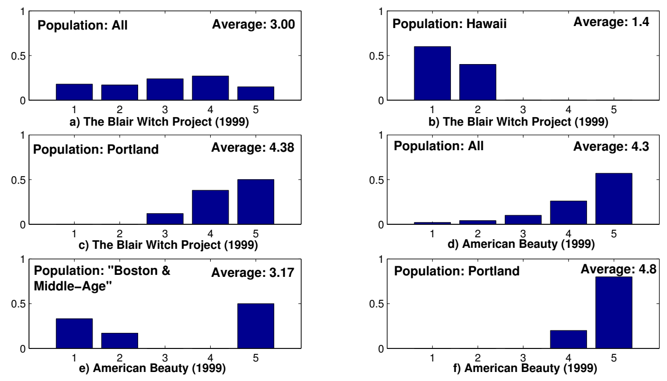
Writers: Aaron Sorkin (s

Stars: Jesse Eisenberg,
Timberlake

Watch Trailer Ad

	Votes	Average
Males	117,061	8.1
Females	22,183	7.9
Aged under 18	6,419	8.5
Males under 18	4,776	8.6
Females under 18	1,576	8.2
Aged 18-29	97,085	8.2
Males Aged 18-29	80,738	8.2
Females Aged 18-29	15,516	7.9
Aged 30-44	30,346	7.8
Males Aged 30-44	26,297	7.8
Females Aged 30-44	3,687	7.7
Aged 45+	6,005	7.6
Males Aged 45+	4,657	7.7
Females Aged 45+	1,272	7.3
IMDb staff	43	8.2
Top 1000 voters	475	7.5
US users	32,848	8.3
Non-US users	95,401	8.0

(a)



(b)

Figure 1.1: Figure showing (a) Demographics Breakdown of Ratings on IMDb (b) Movie Rating Distributions

reflect those of populations it contains. Indeed, while the first movie is preferred in both NYC and Nashville, its average rating overall is lower than the second movie's. This is also known as the Simpson's Paradox [1]. Such user groups are not always known in advance and need to be discovered. In particular, given the large number of groups that could be derived by sub-setting items and users in an input rated dataset ⟨user,item,rating⟩, a strategy is needed to determine the most relevant of such groups according to a query semantics. In this paper, we explore using *rating distributions* as input and finding groups that compare favorably to input distributions (referred to as *query distributions*).

[1] *http://en.wikipedia.org/wiki/Simpson's_paradox*

Group	**Movie1**	**Movie2**	**Winner**
New York	$10 = \frac{1000}{10}$	$8 = \frac{7200}{900}$	Movie1
Nashville	$3.5 = \frac{3150}{900}$	$1 = \frac{100}{100}$	Movie1
Total	$4.15 = \frac{4150}{1000}$	$7.3 = \frac{7300}{1000}$	Movie2

Table 1.1: An Example of Simpson's Paradox

To illustrate the benefit of comparing rating distributions directly instead of relying on their averages, we use real examples from MovieLens [14] and focus on specific groups of ratings from users in Hawaï, Boston and Portland for the movies `American Beauty` (`AB`) and `The Blair Witch Project` (`BWP`) (see Figure 1.1(b)). While `AB` was globally preferred to `BWP`, the average rating of people in Portland for `BWP` is higher than that of middle-aged users in Boston for `AB`. Moreover, while the overall population is neutral about `BWP`, giving it an average of `3`, people in Hawaii hated that movie while people in Portland clearly liked it. Finally, rating averages of middle-aged users in Boston for `AB` and of all users for `BWP` are very similar (resp., `3.17` and `3`), but their *rating distributions are quite different.* In the case of `AB`, users are *polarized* and in the other case, ratings are relatively *uniform*, i.e., random. Clearly, average ratings are too coarse to reveal interesting patterns and rating distributions enable finer comparisons of ⟨user,item,rating⟩ groups.

1.2 Goals and Challenges

Our thesis is that the exploration of rated datasets is made effective by the ability to identify ⟨user,item,rating⟩ groups that satisfy conditions on users, items and ratings. While conditioning users and items is straightforward and can be done with regular relational operators, *conditions on ratings are enabled with the ability to compare rating distributions.* For example, a user exploring a rated dataset may be interested in finding movies among new releases for which the ratings are (almost) *unanimously* very high. Or she may be interested in movies for which the ratings among teens, such as herself, are *polarized.* In these examples, an input rating distribution is used as a *query* in order to find ⟨user,item,rating⟩ groups that exhibit similar distributions. In each case, users could represent the entire rater population or a subset (teens) and the group could contain one or a class of items. Enabling those operations raises two important challenges.

The first challenge is choosing the right measure for gauging the proximity of rating distributions. Many distance measures for distributions have been used in previous work in various contexts. These include KL-divergence and cosine distance. We show that they suffer from a fundamental problem: they do not discriminate pairs of distributions which are intuitively close to each other from those that aren't. We propose to use Earth Mover's Distance (`EMD`) [16], a measure that intuitively captures the minimum amount of work required to transform one distribution into another. The closer the distributions the lower their `EMD` *penalty*.

Our second challenge is computational: Given a set of query distributions, our problem is to find a (potentially partial) partition of the rated dataset into reasonably sized $\langle$user,item,rating$\rangle$ groups such that each group in the partition enjoys a rating distribution that has a low `EMD` with respect to some query distribution. Such groups need to be *dynamically discovered from a potentially exponential search space* and have to be *describable*: "Middle aged Minnesotans in the construction business", "Horror movies from the 90's" or "Comedies rated by teenagers" are examples of describable groups. We show that the problem of finding such groups is NP-hard under two different settings and propose to adapt decision trees to work with `EMD` and split an input rated dataset along user and item attributes. It has been found in the area of classification that the approach of random forests, pioneered by Breiman [2], essentially an ensemble classifier, has a much better performance than any one decision tree. We exploit this intuition using `EMD`-based decision trees and explore different strategies for combining multiple `EMD`-based decision trees.

We make the following contributions.

- We formalize the exploration of a rated dataset as the problem of finding a (potentially partial) partition of the dataset into reasonably sized $\langle$user,item,rating$\rangle$ groups such that each group enjoys a rating distribution that has a low `EMD` with respect to some query distribution (Section 2).

- We show that well-known rating aggregation and comparison measures are not well-adapted for comparing rating distributions and propose using Earth Mover's Distance (`EMD`) for this purpose (Section 4.1).

- We show that the problem of finding $\langle$user,item,rating$\rangle$ groups that "match" query distributions is NP-hard under two different settings and develop appropriate heuristics. The first heuristic is based on adapting decision trees. The second one is based on the random forest approach. Our decision tree construction algorithm makes use of an elegant linear time algorithm for computing `EMD` along with clever pruning techniques inspired by the well-known `NRA` algorithm in top-k query processing. Our random forest approach explores different strategies for combining decision trees (Section 5.1).

- We run comprehensive experiments on two real data sets and a synthetic data set and demonstrate the effectiveness of querying by rating distributions using two kinds of offline quality evaluation. We also report the scalability of several competing algorithms (Section 6). Our results show that in general, random forest approaches are superior and examine in detail when decision tree algorithms perform better.

Related work is discussed in Section 7. We conclude the paper with future directions in Section 8.1.

Chapter 2

Data Model

A rated dataset consists of a set of users with schema $S_{\mathcal{U}}$, a set of items with schema $S_{\mathcal{I}}$ and a set of rating actions with schema $S_{\mathcal{R}}$. E.g., we could have $S_{\mathcal{U}} = \langle$`uid`, `age`, `gender`,`state`, `city`$\rangle$ and a user might be represented as $\langle$*u1*, *18*, *student*, *new york*, *nyc*$\rangle$. Similarly, restaurants on Yelp[2], can be described with $S_{\mathcal{I}}$=$\langle$`item_id`, `cuisine`, `attire`, `ambiance`, `price`$\rangle$, and an example restaurant represented as $\langle$*i2*, *Thai*, *Formal*, *Romantic*, *Moderate*$\rangle$. Finally, a rating action can be represented using the schema $S_{\mathcal{R}} = \langle$`uid`, `item_id`, `rating`$\rangle$, where the domain of attribute `rating` depends on the application and the dataset. E.g., in MovieLens, it is $\{1, ..., 5\}$ and in BookCrossing, $\{1, ..., 10\}$. An *instance* in our data model consists of relations $\mathcal{U}, \mathcal{I}$, and $\mathcal{R}$ of users, items, and rating actions over their respective schema.

2.1 Groups and Rating Distributions

Groups. A rated dataset effectively contains tuples of the form $\langle$user,item,rating$\rangle$. Collaborative rating sites typically contain millions of such tuples. We adopt the approach proposed and experimentally validated in [6], where sets of rating tuples of the form $\langle$user,item,rating$\rangle$, are viewed according to groups that are *structurally describable* using a conjunction of predicates on user and item attributes. E.g., a group may represent a set of rating tuples on Thai restaurants by young males in NY. We let g denote a group, *g.idesc* the set of item predicates, in this case {`cuisine`= `Thai`}, and *g.udesc* the set of user predicates, in this case {`gender`= `male` & `age` $\leq$ `35` & `state`=`new york`}, that are associated with g. In what follows, we use the term describable groups to mean groups which are describable using a conjunction of predicates of the form `Attr` op `val` where `Attr` is a user or item attribute, `val` is a domain value, and op is one of $=, <, \leq, >, \geq$.[3] For categorical attributes, we only allow the predicate

[2] *http://www.yelp.com*

[3] We found descriptions involving $\neq$ are not very intuitive unless they are combined with one of the other predicates above, so we disallow them w.l.o.g.

`Attr` = `val`. We abuse the notation and write $u \in g$, (resp., $i \in g$), to mean that user u, (resp., item i), satisfies all the user, (resp., item), predicates in $g.udesc$, (resp., $g.idesc$).

Rating Distributions. The set of all (describable) groups that contributed ratings in a rated set $\mathcal{R}$ is denoted $G^{\mathcal{R}}$. Given a group $g \in G^{\mathcal{R}}$, we define $ratings(g, \mathcal{R}) = \{\langle u, i, r\rangle \in \mathcal{R} \mid u \in g \,\&\, i \in g\}$ as the set of rating actions of all users in g on items in g, in the rated set $\mathcal{R}$. The rating distribution of g in $\mathcal{R}$ is defined as $dist(g, \mathcal{R}) = [1 : w_1 \ldots\ m : w_m]$ where the rating scale is $\{1, ..., m\}$ and $w_j = \frac{|\{\langle u,i,r\rangle \in ratings(g,\mathcal{R}) | r=j\}|}{|ratings(g,\mathcal{R})|}$ is the fraction of ratings with value j in $ratings(g, \mathcal{R})$. When it causes no confusion, we blur the distinction between g and $ratings(g, \mathcal{R})$ and speak of the tuples in g or the size $|g|$ of g.

Figure 1.1(b) contains many example rating distributions, including those of Portland users for two movies: [0 0 0.1 0.4 0.5] for `The Blair Witch Project` (`BWP`) and [0 0 0 0.2 0.8] for `American Beauty` (`AB`).

Query Distributions. Users might be interested in finding groups whose distributions are similar to *query distributions* of interest to them. Some example query distributions include unanimous distributions $U_1, ..., U_m$ where U_i denotes the distribution where the mass is concentrated at rating value i: $U_i(i) = 1$ and $U_i(j) = 0, j \neq i$; moderately unanimous distributions $U_{i,i+1}$, where the mass is concentrated on successive rating values $i, i+1$: e.g., $U_{i,i+1}(i) = U_{i,i+1}(i+1) = 0.5$ and $U_{i,i+1}(j) = 0, j \neq i, i+1$; and polarized distribution $P_{1,m}$ where mass is concentrated on the extreme ratings 1 and m: e.g., $P_{1,m}(1) = P_{1,m}(m) = 0.5$ and $P_{1,m}(j) = 0, j \neq 1, m$. Indeed, the user may use any distribution(s) as query distributions.

Example 1 (Querying by Distribution) Consider the set $\mathcal{S}$ of ratings of movie `BWP` by all users and let the query $Q = \{\rho_1, \rho_2\}$ contain the distributions $\rho_1 = [0, 0, 0, 0.4, 0.6]$ (high ratings) and $\rho_2 = [0.5, 0, 0, 0, 0.5]$ (polarized). In this case, we are interested in finding $\langle$user,item,rating$\rangle$ groups in $\mathcal{S}$ whose rating distributions on `BWP` are close to a distribution in Q, be it high or polarized. As a second example, Q may be the rating distribution of Portland users on `BWP`. In this case, we are asking for groups whose rating distribution on `BWP` is close to that of Portland raters. In a last example, the user, a New Yorker, may have first explored ratings of New Yorkers on `Avatar` and on `AB`. Treating these two distributions as query distributions, she might want to find groups whose rating distribution on `BWP` closely resemble any of the query distributions. □

Chapter 3

Problem Definition

3.1 Exploration Problem

In this paper, we are interested in developing a framework for helping users explore a rated dataset. At the core of this is the ability to compare rating distributions. Our framework aims to cover different scenario where two rating distributions may represent the opinion of the same users on different items or the opinions of different users on the same or different items. Also, the end user may wish to explore ratings at the granularity of individual items (e.g., a specific digital camera) or sets of items (e.g., the class of Nikon SLR digital cameras). Similarly, she may wish to compare the opinions of two specific users (her friends John and Mary) or two sets of users (teens in NY vs. teens in CA). In its general form, a user query contains conditions on items, conditions on users and a set of rating distributions, referred to as query distributions, that represent the desired rating distributions of returned $\langle$user,item,rating$\rangle$ groups.

Given the input rated dataset $\mathcal{R}$, it is easy to see that the application of conditions on items and users is straightforward as it amounts to finding the subset $\mathcal{S} \subseteq \mathcal{R}$ where each tuple $\langle$user,item,rating$\rangle$ satisfies input item and user conditions. Hence we focus on conditions on rating distributions. For now, let us assume a generic function `ratComp` that compares two rating distributions ρ_1, ρ_2 and returns a score to reflect how far apart ρ_1 and ρ_2 are. We will explore the choices for `ratComp` in Section 4.1. The idea is that given $\mathcal{S}$, we want to *discover* $\langle$user,item,rating$\rangle$ groups whose rating distribution on the item(s) in the group is very close to one of the given query distributions.

The resulting groups must be describable, reasonably sized and together cover as many of the input tuples $\langle$user,item,rating$\rangle \in \mathcal{S}$ as possible. Indeed, splitting $\mathcal{S}$ into describable groups whose ratings are close to query distributions does not guarantee that all tuples in $\mathcal{S}$ will be in the resulting groups. Also, shorter group descriptions are preferred as they are meant for end-user consumption. Hence, there are two ways of formulating our problem. Since we can only optimize one objective, we can either *maximize coverage* of tuples in $\mathcal{S}$ or *minimize the description length* of resulting groups. We call $\pi = [g_1, ..., g_m]$ a partial partition of $\mathcal{S}$ if g_i are pairwise disjoint and $\bigcup_i g_i \subseteq \mathcal{S}$.

Problem Statement: Maximum coverage. Given a rated dataset $\mathcal{S} \subseteq \mathcal{R}$, a group size threshold b, a set of query distributions $Q = \{\rho_1, ..., \rho_k\}$, and a rating proximity threshold θ, find a (possibly partial) partition $\pi = [g_1, ..., g_m]$ of $\mathcal{S}$, such that (i) each g_i is a describable group, (ii) $|g_i| \geq b$, $\forall i \in [1, m]$, (iii) $\forall i \in [1, m]$, $\texttt{ratComp}(dist(g_i, \mathcal{S}_i), \rho_j) \leq \theta$, for some $j \in [1, k]$, and (iv) $\frac{|\bigcup_i g_i|}{|\mathcal{S}|}$ is maximized.

The first condition states that each block in the partition is describable with at least one user attribute in $g_i.udesc$ or one item attribute in $g_i.idesc$. The second condition enforces that each group should not be too small, i.e., must contain at least b tuples. The third condition states that the rating distribution of each group g_i should be at a distance no more than θ to one of the query distributions ρ_j. The last condition says the fraction of input tuples $\mathcal{S}$ that are covered by π should be maximized.

Problem Statement: Minimum description length. Given a rated dataset $\mathcal{S} \subseteq \mathcal{R}$, a rating proximity threshold θ, and a set of query distributions $Q = \{\rho_1, ..., \rho_k\}$, find a (possibly partial) partition $\pi = [g_1, ..., g_m]$ of $\mathcal{S}$, such that (i) each g_i is a describable group, (ii) $\forall i \in [1, m]$, $\texttt{ratComp}(dist(g_i, \mathcal{S}_i), \rho_j) \leq \theta$, for some $j \in [1, k]$, and (iii) $\sum_{i \in [1,m]} |g_i.udesc \cup g_i.idesc|$ is minimized.

The main difference with the first problem is that instead of maximizing coverage, we want to minimize the total description lengths of the groups returned. Since intuitively, shorter descriptions contain more tuples, we don't constrain the group size.

In the rest of the paper, by partition, we mean a possibly partial partition.

Chapter 4

Rating Comparison Measures

4.1 EMD Vs. Other Measures

A key ingredient in the problem we study is the choice of the function `ratComp` that quantifies the proximity between two distributions. We first briefly review some obvious candidate choices for this function. Recall, the rating distributions we consider are normalized so they are probability distributions. KL-divergence [11] is a well-known measure used for gauging the proximity between probability distributions ρ_1 and ρ_2. It is defined as $D_{KL}(\rho_1||\rho_2) = \sum_j \rho_1^j log(\rho_1^j/\rho_2^j)$. Another candidate is a symmetric version of KL-divergence, called JS-divergence [11], and defined as $D_{JS}(\rho_1, \rho_2) = 1/2(D_{KL}(\rho_1||\rho_3) + D_{KL}(\rho_2||\rho_3))$, where $\rho_3 = 1/2(\rho_1 + \rho_2)$. Or we could interpret rating distributions as vectors (over rating values as dimensions) and use cosine distance or Euclidean distance. Unfortunately, all these measures have limitations which make them a poor choice, as we next illustrate with an example.

Example 2 (Rating Comparison Example) Consider three rating distributions $\rho_1 = [0.9, 0.025, 0.025, 0.025, 0.025]$, $\rho_2 = [0.025, 0.9, 0.025, 0.025, 0.025]$, and $\rho_3 = [0.025, 0.025, 0.025, 0.025, 0.9]$, illustrated in Figure 4.1. Say they correspond to ratings by the same set of users on three different digital cameras c_1, c_2, c_3. Intuitively, distributions ρ_1 and ρ_2 are more in agreement with each other than ρ_1 and ρ_3: the users have very similar opinions about cameras c_1 and c_2 and very different opinions about c_1 and c_3. Table 4.1 shows the considered distance scores for the two pairs of distributions. Notably, only `EMD` (described below) distinguishes between the two pairs. □

Earth Mover's Distance. The Earth Mover's Distance (`EMD`) is a widely used distance measure for distributions [16]. Intuitively, the `EMD` between two distributions is the minimum amount of work done per unit

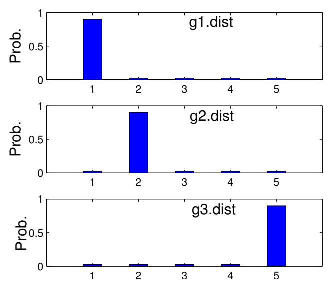


Figure 4.1: Examples of Rating Distributions

Measure	(ρ_1, ρ_2)	(ρ_1, ρ_3)
$Cosine$	0.058	0.058
$KL - Divergence$	3.13	3.13
$JS - Divergence$	0.53	0.53
$Euclidean$	1.24	1.24
`EMD`	1	4

Table 4.1: Distance Measures

mass in converting one distribution to another, where the distributions are viewed as piles of earth at various positions. If D is a discrete domain, `EMD` is computed based on a solution to the well-known transportation problem [16]. Let $\rho_1 = [1 : p_1, .., i : p_i, .., n : p_n]$, $\rho_2 = [1 : q_1, .., i : q_i, .., n : q_n]$ represent two probability distributions over a discrete domain $D = \{1, 2, 3, ..., n\}$. The amount of work required to convert ρ_1 to ρ_2 is defined as:

$$\min_F \texttt{Work}(\rho_1, \rho_2, F) \ = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} f_{ij}$$

subject to the constraints: $f_{ij} \geq 0 \quad 1 \leq i, j \leq n$; $\sum_{j=1}^{n} f_{ij} = p_i \quad 1 \leq i \leq n$; and $\sum_{i=1}^{n} f_{ij} = q_j \quad 1 \leq j \leq n$, where f_{ij} is the amount of mass moved from position i to j in the process of converting ρ_1 to ρ_2. $F = [f_{ij}]$ is the matrix representing the flows and d_{ij} is the ground distance from position i to j, which, for our setting, is defined as the absolute difference in positions, $|i - j|$.

A flow F is optimal if the work done is minimum in that flow. Therefore, the `EMD` is defined as:

$$\texttt{EMD}(\rho_1, \rho_2) \ = \frac{\min_F \texttt{Work}(\rho_1, \rho_2, F)}{\sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij}}$$

EMD is work done per unit mass in an optimal flow. In our setting, the region D over which EMD is calculated is always the whole domain of the distribution, so the value of the denominator in the above equation is 1. So we can ignore the denominator and speak of EMD as the work done itself.

The EMD definition clearly captures fine-grained differences between two rating distributions in the form of work done. This makes EMD particularly suitable for comparing rating distributions.

Chapter 5

Algorithms

5.1 Exploration Algorithms

Our problems, defined in Section 3.1, require to develop efficient algorithms for dynamically finding and comparing relevant groups. In the next subsection, we outline our approach and discuss the inherent complexity of the problems. In the following subsections, we discuss our algorithms.

5.1.1 Complexity of Problems

Given a rated dataset $\mathcal{S} \subseteq \mathcal{R}$, our first problem, `Maximum Coverage`, involves finding a partition of $\mathcal{S}$ whose groups are describable, are large enough, between them cover the maximum number of tuples of $\mathcal{S}$ and are close enough to one of the given query distributions. Let g be a group obtained from $\mathcal{S}$ and Q a set of query distributions. We abuse the notation and write $\mathtt{EMD}(g, Q)$ to mean the `EMD` between g and its *closest* query distribution in Q. We next show that `Maximum Coverage` is NP-hard.

Theorem 1 *The Maximum Coverage problem is NP-hard.*

Proof: We show the decision version of the problem to be NP-hard by reduction from 3-SET PACKING, defined as follows. Given an instance I consisting of a family $\{S_1, ..., S_n\}$ of sets containing exactly 3 elements and a number k, the question is whether I contains k pairwise disjoint sets. This is an NP-complete problem [8]. Given an instance I of this problem, we create an instance J of Maximum Coverage as follows. J consists of a single rating relation, corresponding to ratings on one item. Let $\bigcup_i S_i = \{x_1, ..., x_m\}$. Then J contains n categorical user attributes A_i corresponding to S_i and $2m$ tuples corresponding to $2m$ users; for each x_j, it contains two rated tuples r_j^+ and r_j^-. The values of the tuples are set as follows: whenever $x_j \in S_i$, we set $r_j^+[A_i] = a_i$ and $r_j^-[A_i] = b_i$. The rating value of all tuples of the form r_j^+ is 5 (max) while that of tuples of the form r_j^- is 1 (min). The values of all other attributes are distinct constants appearing

uid	A_1	A_2	A_3	rating
r_1^+	a_1			5
r_2^+	a_1			5
r_3^+	a_1	a_1		5
r_4^+		a_1	a_1	5
r_5^+		a_1	a_1	5
r_6^+			a_1	5

uid	A_1	A_2	A_3	rating
r_1^-	b_1			1
r_2^-	b_1			1
r_3^-	b_1	b_1		1
r_4^-		b_1	b_1	1
r_5^-		b_1	b_1	1
r_6^-			b_1	1

Table 5.1: Instance of Maximum Coverage obtained by reduction from 3-SET PACKING. Blanks represent distinct constants appearing nowhere else.

nowhere else. The query distributions are $Q = \{U_1, U_5\}$ and set the group size threshold to $b = 3$ and the rating proximity threshold to $\theta = 0$. It can be shown that I has k pairwise disjoint sets iff J admits a partition of describable groups of size $\geq b$ that covers exactly $6k$ elements. For example, if $I = \{\{x_1, x_2, x_3\}, \{x_3, x_4, x_5\}, \{x_4, x_5, x_6\}\}$, then $J = \{r_1^+, r_1^-, ..., r_6^+, r_6^-\}$, illustrated in Table 5.1.

- Given a solution for I, solution to J is obtained by grouping tuple corresponding to each set. In the above example solution to I is $\{\{x_1, x_2, x_3\}, \{x_4, x_5, x_6\}\}$, then corresponding solution to J is $\{\{r_1^+, r_2^+, r_3^+\}, \{r_4^+, r_5^+, r_6^+\}\}, \{r_1^-, r_2^-, r_3^-\}, \{r_4^-, r_5^-, r_6^-\}\}$.
- Given a solution for J one can obtain solution for problem I, by choosing corresponding sets in the input. The construction is designed in such a way that a group is describable iff it corresponds to a set in I. For instance in the above example set $\{r_1^+, r_2^+, r_3^+\}$ is describable using description $A_1 = a_1$, but the set $\{r_2^+, r_3^+, r_4^+\}$ is not describable as negations are not allowed in the attributes. In the above example solution to J is $\{\{r_1^+, r_2^+, r_3^+\}, \{r_4^+, r_5^+, r_6^+\}\}, \{r_1^-, r_2^-, r_3^-\}, \{r_4^-, r_5^-, r_6^-\}\}$, then solution to I is $\{\{x_1, x_2, x_3\}, \{x_4, x_5, x_6\}\}$. $\square$

Next, consider the problem of `Minimum Description Length`. We formalize this objective in terms of decision trees. More precisely, we seek to find partitions by constructing decision trees, with predicates over attributes used as split conditions, where the leaves correspond to the groups of the partition. The descriptions can be obtained by reading the predicates off the paths. We call such trees *partition decision trees.* Figure 5.2 shows an example. Ignore paths labeled by negative predicates such as $age \neq teen$ for now. We will discuss this later.

Given a rated dataset $\mathcal{S} \subseteq \mathcal{R}$, a rating proximity threshold θ, we want to find a partition of $\mathcal{S}$ containing groups whose descriptions are short and whose `EMD` w.r.t. some query distribution is $\leq \theta$. When formulated in terms of decision trees, this problem corresponds to minimizing the height of the decision tree. More precisely, let T be such a decision tree whose leaves form a partition $\pi = [g_1, ..., g_m]$ of $\mathcal{S}$. Then minimizing $\sum_i (|g_i.udesc| + |g_i.idesc|)$ is equivalent to minimizing the total length of all root-to-leaf paths. This total length is in turn minimized exactly when the height of T is minimized. We next show:

Theorem 2 *Given a rated dataset $\mathcal{S} \subseteq \mathcal{R}$, a set of query distributions Q and an EMD threshold θ, finding a minimum height partition decision tree for $\mathcal{S}$ where each group's EMD to some distribution in Q is $\leq \theta$ is NP-hard.*

Proof Sketch. We show the decision version to be NP-hard by reduction from the classic Minimum Height Decision Tree problem, defined as follows. Given a set I of n m-bit vectors and a number k, the question is whether there is a binary decision tree with height $\leq k$ such that, each of its leaves is a unique bit vector in I and internal nodes are labeled by binary tests on some bit. From this instance I, we construct an instance J as follows. J has m binary attributes $A_1, ..., A_m$ and a categorical attribute A_0. For each bit vector s_i, J has two tuples t_i^+ and t_i^-, $i \in [1, n]$, with $t_i^+[j]$ and $t_i^-[j]$ set to the j-th bit of vector s_i, $j \in [1, m]$. Assign $t_i^+[A_0]$ and $t_i^-[A_0]$ to two distinct constants appearing nowhere else. Finally, the rating value for t_i^+ (resp., t_i^-) is 5 (resp., 1). Set the `EMD` threshold $\theta = 0$ and let the query distributions be $Q = \{U_1, U_5\}$. E.g., if $I = \{011, 010, 100\}$ then J contains the tuples $(a_1, 0, 1, 1, 5), (b_1, 0, 1, 1, 1), (a_2, 0, 1, 0, 5), (b_2, 0, 1, 0, 1)$, $(a_3, 1, 0, 0, 5), (b_3, 1, 0, 0, 1)$, where the last value is the rating. We have:
Claim. I admits a decision tree of height $\leq k$ iff J admits a partition decision tree of height $\leq k + 1$ where each group at its leaf is describable and exactly matches U_1 or U_5.
Only If: Given a decision tree T for I, by definition, it contains a unique bit vector $s_i \in I$ at each leaf. If we apply this tree to J, we will get a tree each of whose leaves contains a group containing exactly the tuples $\{t_i^+, t_i^-\}$, $i \in [1, m]$. These groups do not match either of U_1, U_5. Applying a split based on $A_0 = a_i$ versus $A_0 = b_i$ divides this group into two singleton groups $\{t_i^+\}$ and $\{t_i^-\}$ which match U_5 and U_1. The groups are describable. This tree has height one more than that of T.

If: Let T be a partition decision tree of height $\leq k + 1$ for J. By definition, each leaf of T contains a unique tuple of J. Notice that none of the groups at the leaves can contain more than one tuple with the same rating value, as they are not describable (without distinction's or negations).

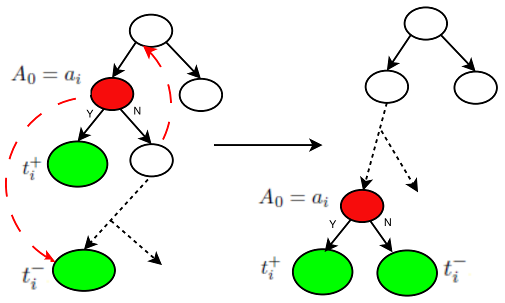


Figure 5.1: Figure shows how to push a node conditioned on attribute A_0 towards the leaf.

T must apply the predicates on attribute A_0 to separate tuples t_i^+ and t_i^-. Suppose T applies these tests after all other tests. Then the node at which $A_0 = a_i$ vs. $A_0 = b_i$ is applied must contain exactly the group $\{t_i^+, t_i^-\}$. By replacing that group by the corresponding bit vector s_i, we get a decision tree of height $\leq k$ for I. Suppose T applies one or more tests on A_0 before tests on other attributes A_i, where $i > 0$. Now we show that, we can "push down" these tests (or node) on A_0 so they are applied at the parent of leaf level, without increasing the height of the tree. Figure 5.1 illustrates how on can push the nodes conditioned on A_0. As shown in the figure 5.1, red color node represent the node conditioned on attribute A_0 and green color nodes t_i^+ and t_i^- represents two rating tuples which are separated on attribute A_0. Note that this transformation on tree, does not effect height of green nodes. Though it decreases height of the leaf nodes under red node, it may decrease the height of the tree but it wont increase its height. Modified tree is still a decision tree with required height $\square$

5.1.2 Group Discovery Algorithms

Given that finding partitions that either minimize description length or maximize coverage is NP-hard, a natural question is whether we can develop efficient heuristics, which we next address. Rather than tailor the heuristics specifically for minimizing description length or maximizing coverage, we argue that the algorithms we develop must in general favor short descriptions; in addition to improving understand-ability, they in turn tend to favor larger groups and thus make it more likely to satisfy the size threshold. In other words, they help improve the coverage too. We take a top-down approach for finding partitions, based on decision trees (recall the notion of partition decision trees defined in Section 5.1.1). Decision trees have the benefit that group descriptions can be directly read off the root-to-leaf paths. Whereas classic decision trees are driven by gain functions like entropy [4] and gini-index [5], a novelty in our case is that our decision trees are designed to discover groups whose distributions are close to query distributions. That has the benefit of exploiting `EMD` properties as will be shown in Section 5.2. In the next two subsections, we develop two algorithms – `DTAlg` and `RFAlg`.

Decision Tree Algorithm (`DTAlg`)

Our first algorithm, `DTAlg`, is based on partition decision trees. Algorithm 1 gives the pseudo-code. It takes as input a set of query distributions Q, a set $\mathcal{S}$ of rating tuples on any items, a group size threshold b and a rating proximity threshold θ. For simplicity, we assume all our attributes are categorical. Numerical attributes like `age` and `year` are binned appropriately.[6] Since attributes are binned, the only type of descriptions we need to focus on are conjunctions of `Attr` = `val`. Attribute values can be organized in a hierarchy. Figure 5.2(a), shows a partial hierarchy of attribute `age` from MovieLens dataset.

Algorithm 1 repeatedly divides rating set $\mathcal{S}$, in a breadth first manner, to find interesting groups with short descriptions and large enough sizes. At each node the algorithm checks if its corresponding group is *good*, i.e., it has good description, has size $\geq b$, and has `EMD` $\leq \theta$ to some distribution in Q. It adds good groups to the output list. Here, a description is good if consists of positive atoms of the form `Attr` = `val`. We disallow descriptions consisting of only negative atoms `Attr` $\neq$ `val` since they are hard to understand intuitively. If a group is not good, the algorithm makes the following decisions: the group is dropped if it is too small (line 8); if the group's `EMD` to the closest query distribution in Q is $> \theta$, the algorithm looks for a splitting attribute (using a procedure described in Section 5.1.2) and the group is split accordingly; if the `EMD` is $\leq \theta$ but the group has only negations in its description, the group is split again recursively using the *same* attribute until a description with no negation is found. This process is described in Section 5.1.2.

A key operation that is repeatedly performed by the algorithm is finding the `EMD` of a group w.r.t. a set of query distributions of which the smallest

[4] *http://en.wikipedia.org/wiki/Entropy*

[5] *http://en.wikipedia.org/wiki/Gini_index*

[6] In the real datasets MovieLens and BookCrossings, these attributes are available only as binned.

Algorithm 1 `DTAlg`$(\mathcal{S}, Q, b, \theta)$

```
1: Initialize queue $\mathcal{K} = \phi$ and list $Output = \phi$.
2: Add $\mathcal{S}$ to $\mathcal{K}$.
3: while $not\ empty(\mathcal{K})$ do
4:    $parent = \mathcal{K}.pop()$
5:    if goodGroup(parent) then
6:       Add $parent$ to $Output$.
7:    else if $|parent| < b$ then
8:       $continue$
9:    else if $not\ hasGoodEMD(parent, Q, \theta)$ then
10:      $Attribute\ \ attr =\ \ findBestAttribute(parent)$
11:      $Array\ \ child =\ \ split(parent, attr)$
12:      for $i = 1 \rightarrow size(child)$ do
13:         if $goodGroup(child[i])$ then
14:            Add $child[i]$ to $Output$.
15:         else
16:            Add $child[i]$ to $\mathcal{K}$.
17:         end if
18:      end for
19:   else if $not\ hasGoodDescription(parent)$ then
20:      $Array\ child = removeBadDecription(parent)$
21:      for $i = 1 \rightarrow size(child)$ do
22:         if $goodGroup(child[i])$ then
23:            Add $child[i]$ to $Output$.
24:         else
25:            Add $child[i]$ to $\mathcal{K}$.
26:         end if
27:      end for
28:   end if
29: end while
30: return $Output$
```

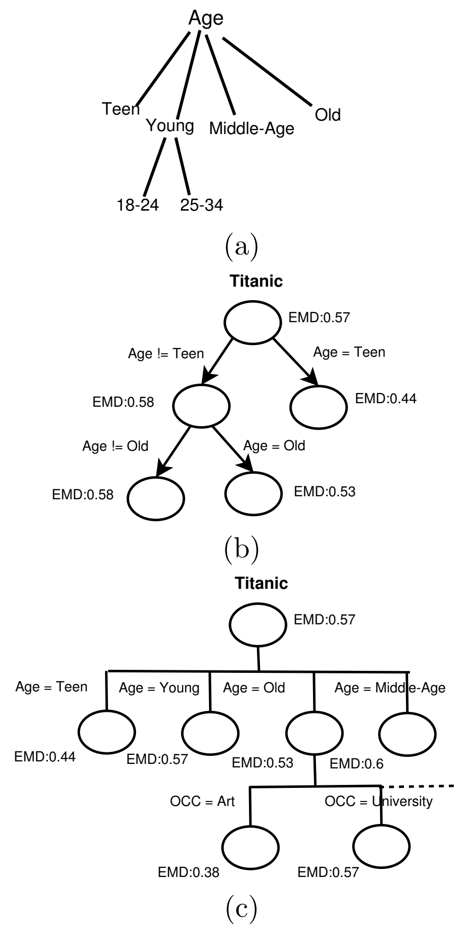


Figure 5.2: `DTAlg` (a) Hierarchy of Attribute Age (b) Binary Split (c) Categorical Split

`EMD` is picked. We anticipate users querying rated datasets with many possible query distributions. Thus it is important to minimize the work done for this operation. In Section 5.2.1, we give an elegant algorithm inspired by the top-k algorithm NRA [7] to accomplish this.

Group Splitting We explored two ways of splitting groups into subgroups that are describable: Given an attribute A_i and a value v_j, *binary splitting* on a group produces two child groups based on two complimentary predicates $A_i = v_j$ and $A_i \neq v_j$. Figure 5.2(b) shows an example of binary splitting on ratings of movie `Titanic` using attribute `age` and value *teen*.

This kind of splitting adds negation to group descriptions. For example, the leftmost node is described with `age` $\neq$ *teen* & `age` $\neq$ *old*.
Categorical splitting on the other hand, results in n child groups, one for each value $v_j \in V$ where $V = \{v_1 \dots v_n\}$ is the active domain of A_i. Figure 5.2(c) shows an example of categorical splitting on ratings of movie `Titanic` using attribute `age`, resulting in four child groups. Unlike binary splitting, categorical splitting does not introduce negation in the description of child groups. Thus, the check for goodness of descriptions performed by Algorithm 1 is needed only for binary splits.

Gain Functions In order to proceed with group splitting, Algorithm 1 calculates a gain for each attribute and picks the one with maximum gain. After experimenting with various choices, we propose using *minimum average* `EMD`. Suppose splitting a group g using an attribute A_i yields m child groups $c^i_1 \dots c^i_m$. The gain of A_i is defined as the inverse of average `EMD` of its child groups. If child groups have zero `EMD` then the gain is infinity. More formally:

$$\texttt{Gain}(A_i) = \frac{m}{\sum_{j=1}^{m} \texttt{EMD}(c^i_j, T)}$$

Finally, an attribute may not be useful for splitting a group, if all the rating tuples in it have the same value for that attribute. For example, if a group has ratings of a single movie `Titanic` then none of the movie attributes are useful for splitting the ratings. Such attributes, if discovered at any group are discarded and are not considered for further splitting of any of the children groups. Algorithm 2 describes an algorithm for categorical splitting.

Algorithm 2 : $findBestAttribute(g)$

```
Require: $Array\ of\ attributes : \ attribs$
1: $maxGain = 0$
2: $maxGainAttribute = -1$
3: for $i = 1 \to size(attribs)$ do
4:    $curGain = findGain(g, attribs[i])$
5:    if $curGain > maxGain$ then
6:       $maxGain = curGain$
7:       $maxGainAttribute = i$
8:    end if
9: end for
10: return $attribs[maxGainAttribute]$
```

Removing Bad Descriptions As discussed earlier, the description of a group is said to be bad if it only contains negative predicates. Suppose group g contains negation predicate $A_i \neq v_i$ for an attribute A_i. Then we can modify the description of the group depending on two situations. In the first situation, if all the rating tuples in the group have a unique value v_j for the attribute $j \neq i$, then the condition on the attribute A_i is replaced with $A_i = v_j$. In the second situation, if there are multiple values for A_i in the group, we split the group recursively until all child groups have a unique value for the attribute A_i. Algorithm 3 describes an pseudo code for handling bad descriptions.

Algorithm 3 : $removeBadDecription(g)$

```
Require: $Array\ of\ attributes:\ attribs$
1: for $i = 1 \rightarrow size(attribs)$ do
2:   if $hasNegation(description(g, attribs[i]))$ then
3:     if $hasUniqueValue(g, attribs[i])$ then
4:       $description(g, attribs[i]) = uniqueValue$
5:       return $R$
6:     else
7:       $Array\ childs = split(g, attribs[i])$
8:       return $childs$
9:     end if
10:  end if
11: end for
```

The Random Forest Approach (`RFAlg`)

One of the shortcomings of the `DTAlg` algorithm is that the condition used to split the root will be shared by all groups. E.g., in the Figure 5.2(b) and (c), every group obtained will have a condition on attribute `age`. Although not a goal in itself, the diversity in the descriptions of groups that belong to the same partition may matter. In particular, groups sharing the same root attribute will have lower diversity, as measured using, e.g., the Jaccard distance between their descriptions. To mitigate this, we draw inspiration from the work of Breiman [2] who proposed the approach of *Random Forests* (`RF`) to dramatically improve the performance of decision tree based classifiers. The idea is given n predictor attributes, a classical decision tree examines all n of them to pick the best split attribute and split point at each stage. The `RF` approach consists of two main steps.

In step 1, the classical decision tree algorithm is run m times for some parameter m, where each run examines a random subset of k predictor attributes at each splitting node, a default value for k being $\sqrt{n}$. This generates m decision trees. In step 2, the decisions of these m trees are combined, e.g., by voting, to yield an ensemble classifier.

We adapt random forests to our setting. Step 1 of the `RF` approach remains the same: we run the `DTAlg` algorithm on a random subset of $\sqrt{n}$ user/item attributes available m times. For us, step 2 should produce a partition, not a classifier. In the next section, we examine different strategies for combining the m partitions from step 1 into a single partition.

Combining Partitions In this section, we describe five heuristics for combining the m partitions $\pi_1, ..., \pi_m$ produced in step 1 of the `RF` approach into a single partition.
`1. RF-Cluster`: Each partition π_i intuitively captures a (possibly partial) clustering. For each pair of rated tuples $r_i, r_j \in \mathcal{S}$, we can determine the fraction of partitions in which they are clustered together. Let k_{ij} be the number of partitions in which r_i and r_j are clustered together. Then we can define the distance between r_i and r_j as $d_{ij} = k_{ij}/m$. Now, we can use any standard clustering algorithm to obtain a clustering of $\mathcal{S}$ with this distance measure. After examining several alternate clustering algorithms, we chose hierarchical clustering. As the desired number of clusters, we chose the average number of groups in the partitions $\pi_1, ..., \pi_m$. Once groups are obtained, we discard groups g for which $|g| < b$ or $\texttt{EMD}(g, Q) > \theta$ holds. The resulting groups do not necessarily have a natural exact description. We take a pattern mining approach to solve this issue. Viewing each rated tuple as a transaction and each (user or item) attribute as an "item", we obtain maximal frequent patterns, by setting the support threshold to 90%. Any maximal frequent pattern serves as an approximate description of the group, with an accuracy of at least 90%.
`2. RF-Description`: A second strategy favors a partition with diverse groups w.r.t. their descriptions. Define the Jaccard distance between groups g_i, g_j as $Jaccard(g_i, g_j) = |desc(g_i) \cap desc(g_j)|/|desc(g_i) \cup desc(g_j)|$. Start with an empty output partition and successively add a group whose total Jaccard distance to existing groups in the output is maximum until we cannot add any more groups. The first group is picked at random. This strategy favors diversity among the group descriptions. We will verify that in our experiments in Section 6.

A common theme among the remaining strategies is that we order the groups from the various partitions $\pi_1, ...\pi_m$ using some criterion. Then we add these groups to the output partition one by one: if a group overlaps with existing groups in the output, we discard it. Thus, the only difference is in the way groups are ordered.

3. `RF-Random`: Order the groups in a random manner.
4. `RF-Size`: Order the groups in decreasing order of their size. The rationale is to favor larger groups, which may help with the coverage.
5. `RF-EMD`: Order the groups in increasing order of their `EMD` to their closest query distribution in Q. This favors a solution where groups closer to some query distribution are preferred.

We expect any of the `RF` algorithms to take more time than `DTAlg` since m decision trees are constructed and further time is incurred for combining their groups. Among them, RF-Cluster is by far the most expensive since it requires finding the distance between all pairs of tuples in $\mathcal{S}$.

5.2 EMD Algorithms

A key operation that is repeatedly performed by both `DTAlg` and `RFAlg` is finding the `EMD` between a group and its closest query distribution and it is important to do this efficiently. In general, the calculation of `EMD` between two given distributions can be done using the Hungarian algorithm and takes time $O(n^3 log\, n)$ where n is the domain size of the distribution [10]. However, in our setting, the distributions are probability distributions and are over the same domain (rating scale), thus it is possible to compute their `EMD` in linear time [16]. A similar observation was also exploited by Li et al. [12] in a different context. The key insight is to use a stack to manipulate the *flow* that corresponds to the amount of mass moved to transform one distribution to another. The stack helps minimize the work needed to find the closest distribution to a given distribution, from among a set of query distributions.

5.2.1 Computing EMD of Two Distributions

Suppose we want to measure the `EMD` between two rating distributions ρ_1 and ρ_2. We make one pass over distributions ρ_1, ρ_2 starting from the left-most positions (1). A position i is *excess* (resp., *deficit*, *equal*) position if $\rho_1[i] > \rho_2[i]$ (resp., $\rho_1[i] < \rho_2[i]$, $\rho_1[i] = \rho_2[i]$). We keep track of each position as we scan it. It moves mass such that ρ_1 converts to ρ_2, and keeps track of the mass flow $F[i, j]$ from position i to j. 4,5 together describe pseudo code of the algorithm.

Algorithm 4 : `EMD`$(\rho_1[1,n], \rho_2[1,n])$

```
work = 0 // work done
Stack $S = \phi$
$state = equal$
for $i = 1 \to n$ do
   if $\rho_1[i] > \rho_2[i]$ then
      if $state = excess\ or\ equal$ then
         S.push($\{i, (\rho_1[i] - \rho_2[i])\}$)
      else if $state = deficit$ then
         work += updateStack(i)
      end if
   else if $\rho_1[i] < \rho_2[i]$ then
      if $state = deficit\ or\ equal$ then
         S.push($\{i, (\rho_2[i] - \rho_1[i])\}$)
      else if $state = excess$ then
         work += updateStack(i)
      end if
   end if
end for
return $work$
```

Algorithm 5 : `updateStack`(i)

```
work = 0;
mass = $|\rho_1[i] - \rho_2[i]|$
while mass >0 and S.isEmpty = false do
    $\{k, kmass\}$ = S.pop()
    if mass $\geq kmass$ then
        work = work + ($kmass$)($|i - k|$)
        updateFlow($kmass$,i,k,state)
        mass -= $kmass$
    else
        work = work + (mass)($|i - k|$)
        updateFlow($mass$,i,k,state)
        mass = 0;
        S.push($\{k$, (kmass$-mass$)$\}$)
    end if
end while
if mass >0 then
    state = !state //invert
    S.push($\{i, mass\}$)
end if
if S.isEmpty = true then
    state = equal
end if
return $work$
```

step	stack	state	work
0	$\{ \phi \}$	`equal`	0
1	$\{ (1, 0.2) \}$	`excess`	0
2	$\{ (1, 0.2) \}$	`excess`	0
3	$\{ (3, 0.1) \}$	`deficit`	0.4
4	$\{ (3, 0.1), (4, 0.2) \}$	`deficit`	0.4
5	$\{ \phi \}$	`equal`	0.8

Table 5.2: Running Example of EMD Computation.

We may first encounter excess or deficit or equal positions. Equal positions are just ignored. Say we first see an excess position. We store it on a stack. The stack is now said to be in *excess state*. Future excess positions are pushed to the stack while deficit positions are processed by flowing mass out of the top excess position on the stack. The stack may transition to *deficit state* as deficit positions are seen. Thus the stack is always in a well-defined state – excess or deficit. It reaches an *equal state* when it is empty. For each position type, we perform the following actions.
Excess Position: This means $\rho_1[i] > \rho_2[i]$. Set the flow $F[i, i] = \rho_2[i]$. If the stack is in equal or excess state, push the entry $(i, \rho_1[i] - \rho_2[i])$ onto stack. This is the excess mass available at i. If the stack is in deficit state, pop the top element, say (j, δ), i.e., there is a deficit of δ at j. If $\mu =_{def} \rho_1[i] - \rho_2[i] > \delta$, set $F[i, j] = \delta$ and decrement δ from μ. Repeat this for remaining deficit positions on stack until excess mass is left in μ. If μ becomes $< \delta$, set $F[i, j] = \mu$, and set $\delta = \delta - \mu$ and $\mu = 0$. In the end, the stack may remain in deficit state, move to equal state (i.e., become empty). If position i remains an excess position push its entry on stack and the stack moves to excess state.
Deficit Position: This means $\rho_1[i] < \rho_2[i]$. It is the mirror analog of the above case and we omit the obvious detail.
Equal Position: In this case, since $\rho_1[i] = \rho_2[i]$, we set the flow $F[i, i] = \rho_1[i]$ and simply move to the next position.

Table 5.2 illustrates the algorithm on a particular example with $\rho_1 = [0.2, 0, 0.2, 0.3, 0.3]$ and $\rho_2 = [0, 0, 0.5, 0.5, 0]$. As shown in the table, in step 1 since there is excess mass of 0.2 at $\rho_1[1]$, and it is pushed to stack and the stack state changes from equal to excess. In step 2, since $\rho_1[2]$ is an equal position, the algorithm simply moves to the next position. $\rho_1[3]$, is a deficit position, so we use excess positions on the stack to flow mass to $\rho_1[3]$. But since there is not enough mass, $\rho_1[3]$ is added to stack with a deficit of 0.1

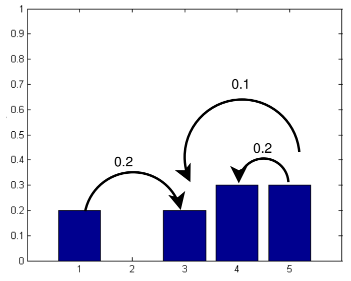


Figure 5.3: Figure showing flows for example in Table 5.2

and the state of the stack is changed to deficit. Since there is a mass flow of 0.2 from position 1 to 3, 0.4 work is added to current work. In the next step, $\rho_1[4]$ is a deficit position so it is pushed to stack. Since $\rho_1[5]$ is an excess position mass is moved from it to previous deficit positions. Figure 5.3 shows the corresponding flows. It is easy to see that $EMD(\rho_1, \rho_2)$ is 0.8.

We can show:

Theorem 3 *Work computed by algorithm to convert ρ_1 to ρ_2 is optimal and is equal to* `EMD`(ρ_1, ρ_2).

The proof, omitted here for lack of space, appears in a technical report, not cited for reasons of anonymity. It is easy to see that the algorithm takes time $O(n)$ where n is the length of the distribution, which in our application is the size of the rating scale.

5.2.2 EMD for Multiple Distributions

Our algorithms for finding partitions, `DTAlg` and `RFAlg`, involve multiple evaluations of the following query: *given a distribution ρ (corresponding to a group) and a set of query distributions $Q = \{\rho_1, ..., \rho_k\}$, what is the* `EMD` *of ρ to its closest distribution in Q?* We anticipate the application of our algorithm for any rated dataset where the rating scale may be large (e.g., Yahoo!Music has a rating scale of 1–100) and where the number of query distributions k may be large (e.g., the user may have seen several distributions while exploring items and may want to know which groups exhibit similar distributions on a given class of items). To make this possible, it is essential to provide efficient support for the above query. A naïve approach is to evaluate `EMD`$(\rho, \rho_i), i \in [1, k]$ and pick the smallest `EMD`. But this is inefficient. In this section, we develop a pruning strategy that relies on maintaining a lower bound ℓ_i and an upper bound u_i on the possible values of

$\texttt{EMD}(\rho, \rho_i), \forall i$. Then whenever $\ell_i > min\{u_j \mid j \in [1,k]\}$, we can safely discard ρ_i as a candidate for being the closest query distribution to ρ. To develop the bounds, consider an algorithm that maintains a list L of promising candidates to be the closest `EMD`-neighbors of the given distribution ρ (equiv., group). Similar to our earlier algorithm for calculating the `EMD` for a pair of distributions, this algorithm makes one concurrent pass over ρ and the query distributions ρ_i. It maintains excess/deficit positions on a stack as before. We assume there are n rating values in the rating scale.

We can show:

Lemma 1 *After the algorithm has examined the first i positions, let δ be the total mass on the stack, j, k be positions on the top and bottom of the stack. Let $\gamma = \sum_{t=1}^{i} \rho_p[t] = \gamma$ be the cumulative mass of distribution $\rho_p \in Q$. Let Δ be the work done so far for converting ρ into ρ_p. Then $\ell_p = \Delta + \delta \times (i+1-j) \leq EMD(\rho, \rho_p) \leq u_p = \Delta + \delta \times (n-k) + (1-\delta-\gamma) \times (n-i-1)$.*

Proof: The key intuition is that after examining i positions, in the best scenario, all the mass (say excess) on the stack just needs to move to the next ($(i+1)$-th) position. Thus, the `EMD` cannot be smaller than the current work done plus this amount. The upper bound corresponds to the worst case where all the mass on stack (say excess) has to move the furthest, i.e., from the bottom position to the very last (n-th) position. Additionally, it is possible that there is excess mass left over at the last position n, which needs to move to position $i+1$. The maximum possible value of this excess mass is $1-\delta-\gamma$. $\square$

We illustrate the pruning algorithm with a simple example. Consider a given distribution $\rho = [0.8, 0.2, 0, 0, 0]$ and the query distributions $Q = \{\rho_1 = [1,0,0,0,0], \rho_2 = [0,1,0,0,0], \rho_3 = [0,0,1,0,0], \rho_4 = [0,0,0,1,0], \rho_5 = [0,0,0,0,1]\}$. After examining the first two positions, the current work done for the various query distributions is 0.2, 0.8, 0, 0, 0. The bounds are: $\ell_1 = u_1 = 0.2$; $\ell_2 = u_2 = 0.8$; $\ell_3 = \ell_4 = \ell_5 = 1$; $u_3 = u_4 = u_5 = 4$. At this point, The lower bounds for $\rho_2, ..., \rho_5$ exceed the upper bound for ρ_1 and we can drop $\rho_2, ..., \rho_5$ from further consideration.

Chapter 6

Experiments

The goal of our experiments is two-fold: validate the quality of partitions obtained by our algorithms and study the scalability of our algorithms. In order to assess the quality of obtained partitions, we perform offline experiments that report measures used to validate clustering performance since our partitioning can be viewed as a clustering of the input rated dataset. We also examine coverage, description length and diversity of descriptions for groups generated using our algorithms. Finally, we evaluate the scalability of our algorithms as a function of the number of ratings and of trees.

6.1 Experimental Setup

We use two rated datasets, 1 million MovieLens (ML) and Book crossing (BC) summarized in Table 6.1. ML contains { *gender, age, occupation, location*} as user attributes. We combine it with IMDb to obtain attributes {*title, actor, director, writer*} for each movie. Similarly BC provides {*location, age*} for users and { *title,author,year,publisher*} for books. Some attributes have a hierarchy. For example, the hierarchy of *location* is *Country* $\rightarrow$ *State* $\rightarrow$ *City*. This information is readily available for every user in BC, but for ML, we queried Yahoo! Maps to get this information. We manually created hierarchies for attributes *age, year* and *occupation.* Other attributes like *director, gender, author* have trivial hierarchies (i.e height =0).

All experiments were done on a Xeon 2.5GHz Quad CoreWindows Server 2003 machine with 16GB RAM and a 128GB SCSI hard disk. All code is written in Java using JDK/JRE 1.6.

Unless mentioned otherwise, `EMD` threshold, $\theta = 0.2$,(2 for BC), number of trees is 4 , and minimum group size is $b = 5$. All our results are averages of 3 runs. For ML, we used the query distribution $Q = \{U_1, U_2, ..., U_5, U_{1,2}, U_{2,3}, U_{3,4}, U_{4,5}, P_{1,5}\}$ and for BC, $Q = \{U_1, U_2, ..., U_{10}\}$.

	MovieLens (+IMDB)	Book Crossing
about	movie ratings	book ratings
users	6040	38511
items	3900	260
ratings	1000209 $(mil\tilde{l}ion)$	196842
Rating Scale	1 to 5	1 to 10

Table 6.1: Summary of datasets

Location/Age	Teen	Young	Middle	Senior	Count
East	$\tilde{U_1}$	U	U	U	2000
Central	U	$\tilde{U_5}$	U	U	1000
West	U	U	U	$\tilde{U_2}$	1000
Count	2000	1000	1000	1000	5000

Table 6.2: Rating distributions of various groups in synthetic data

6.2 Offline Quality Evaluation

6.2.1 Purity, Rand-Index and F-Measure

Since our algorithms partition a rated dataset into groups, we propose to borrow standard clustering evaluation measures like "purity", "Rand-Index" and "F-Measure", to evaluate the quality of our partitions. All those measures require a gold standard. We therefore construct a gold standard by generating synthetic data using known distributions.

We generated 5000 rating tuples for an item, in which users have three attributes *age, occupation, location.* Our gold standard consists of the three groups shown in Table 6.2. For example ratings by user in group, *[age = Teen,Location= East]* are generated from slightly perturbed U_1 distribution represented as $\tilde{U_1}$. We do not use exact U_i, to make the data look more realistic and also to make it more difficult for our algorithms. For this experiment, the query distribution is $Q = \{U_1, U_2, ..., U_5\}$.

`Purity` is defined as the fraction of rating tuples which are "correctly classified" according to our gold standard. Consider a rating tuple r, let g_i be the group to which r belongs in the gold standard and g_j the group to which it belongs in the partition obtained from our algorithm. Let the nearest query distribution (using `EMD`) for g_i be ρ_i and for g_j be ρ_j. In this setting r is correctly classified iff $\rho_i = \rho_j$. Purity is defined as:

$$\texttt{Purity} = \frac{\text{No. of correctly classified tuples}}{\text{Total no. of tuples}}$$

`Purity` measures whether an individual rating tuple r belongs to a group with the same rating distribution. It does not measure pairwise co-occurrence of tuples r_i and r_j. This is measured by `RandIndex` and F_β

Algorithm	Purity	RandIndex	F_1 score	F_2 score	$F_{0.5}$ score
DTAlg	0.9955	0.9058	0.8542	0.6540	0.8842
Rf-Size	0.9923	0.9687	0.9570	0.9129	0.9729
Rf-Description	0.9891	0.9420	0.9177	0.8691	0.9624
Rf-EMD	0.9891	0.8646	0.7707	0.6347	0.8620
Rf-Rand	0.9891	0.8712	0.7904	0.7002	0.8620

Table 6.3: Performance of various algorithms measured using `Purity`, `RandIndex`, `F-Measure`

`score`. These measures look at all pairs of rating tuples and compare their groups in the gold standard and in the algorithm output. Depending on various combinations, each pair is recognized as True Positive (TP) or True Negative (TN) or False Positive (FP) or False Negative (FN). TP, TN, FP, FN have the same meaning as in classic clustering theory. `RandIndex` and F_β `score` are defined as:

$$\texttt{RandIndex} = \frac{TP+TN}{TP+TN+FP+FN}$$
$$\texttt{F}_\beta\texttt{score} = \frac{(1+\beta^2)TP}{(1+\beta^2)TP+\beta^2 FN+FP}$$

Values of all three measures are bounded in the range $[0,1]$. Table 6.3 shows the performance of various algorithms. We do not measure these values for algorithm `RF-Cluster` as it takes a long time to terminate (as shown later). Results show that all algorithms perform equally good in terms of purity. In fact all algorithms nearly achieve maximum value of 1 for purity. Togetherness of rating tuples is measured by `RandIndex`, F_β measure. Overall, the `DT` algorithm may find a smaller child group instead of a big parent group. For example, it may not identify the group *[age = Teen, Location= East]* , but may return all smaller groups in it like *[age = Teen, Location= East, Occupation=lawyer]*. This may happen because in generating trees in `DTAlg`, *Occupation* might have been chosen before *Location*. For any choice of gain function, this can always happen for some input. It is because of this that `DTAlg` does not perform well in terms of these measures. `RF` algorithms overcome this by creating many trees. Clearly, `RF-Size` and `RF-Description` are better heuristics than `RF-EMD` and `RF-Rand`. In fact `RF-EMD` and `RF-Rand` perform worse than `DTAlg`. This shows that the heuristic used in combining output from various `DT` runs of `RF` plays an important role. Our experiments show that `RF-Size` and `RF-Description` are better heuristics with `RF-Size` being the best.

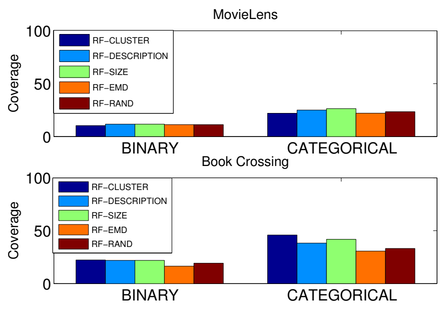


Figure 6.1: Figure showing coverage for two types of split Categorical and Binary on two datasets

6.2.2 Coverage, description, diversity and EMD

We compare the quality of found groups using their *coverage*, *description diversity*, *description length* and EMD. Description diversity and description length are good indicators of the effectiveness of an algorithm, even though the former is not a goal in itself. *Diversity* of a partition is defined as the average pairwise Jaccard between groups in the partitions. That is $diversity(\pi) = \frac{\sum_{i \neq j, i=1}^{m} Jaccard(g_i, g_j)}{m}$.

In our experiments we observed that RF has better coverage when we use categorical splitting. As shown in the figure 6.1, coverage has low value for both the dataset when split type is binary. On close observation of partitions, we found that, partitions obtained from binary splitting overlaps a lot. One possible explanation of this is, in binary split, an attribute can get picked many times. As a result an attribute which has good gain for many attribute values, has more chance of getting picked again and again. Because number of attributes are very few in number (4-5) in both the dataset, this can happened a lot. As a result, partitions from different decision trees overlap a lot. The coverage remains almost same or improves slightly when compared to a partition from a single tree. In rest of the experiment we use categorical splitting. We found that the gain function based on minimum average EMD is the best.

Figure 6.2(a) shows coverage on inputs of various sizes. It can be noticed that, all RF approaches have higher coverage than DTAlg on both datasets. One can notice that the margin between DTAlg and RF approaches increases as input size grows. For large input sizes, coverage by RF algorithm is almost double that of DTAlg. This shows that, single trees may not be sufficient to find non intersecting groups. Among different variants of RF, RF-Cluster has slightly more coverage than others in BC. Finally, RF-Size and RF-Cluster have better coverage than other variants.

Figure 6.2(b) demonstrates the dependence of coverage on query distributions. Items with low variance in their ratings have higher coverage, and coverage decreases as variance increases. Recall that the query mostly consists of either distributions of type U_i or of type $U_{i,i+1}$. For items with lower rating variance, most of the population gives the same rating value, so the rating distribution for such items will be close to one of the query distributions. For such items, coverage is higher. As variance increases, rating distributions get very far from query distributions. For these items, coverage is less. Also obtained group size is larger for items with low variance.

Figure 6.3(a) shows how coverage varies as emd threshold increases. Coverage increases in all cases as threshold increases. It is expected because, now many in the algorithm qualify. Also note that gap between decision tree algorithm and RF-Size initially increases as emd increases, but later both of them converge. It is because, initially as emd increases,coverage of RF-grow rapidly because it discover more number of new blocks than a single decision tree algorithm. But as threshold gains large value, almost all subsets make it to final set. Thus coverage of both the algorithms converge to 100. Figure 6.3(a) shows how coverage changes as no. random trees increase. initially coverage increase as number of random trees increase, but it converge very quickly to a constant value. We found it experimentally that no. of trees at which coverage converge actually depends on size of the input.

Figure 6.2(c) shows the diversity of discovered groups as input size varies. Notice that RF-Cluster has high diversity in description. It is because the description for each set is chosen independently using maximal frequent patterns. DTAlg in general has very low diversity for small input sizes. It is because the height of DTAlg is small and leaves share many nodes resulting in low diversity. Surprisingly, RF-Desc does not have diversity much higher than other RF approaches. Diversity of RF approaches is almost same. We suspect that is because the number of attributes in the datasets is small.

Figure 6.4(b) shows average EMD obtained for various algorithms increases with the number of random trees generated for RF approaches. Since DTAlg is independent of the number random trees, its curve is constant. RF-EMD as expected has the smallest average EMD as the number of trees increases. When the number of random trees is one, average EMD is slightly smaller for DTAlg in ML, but almost the same for BC. The gap between average EMD for RF approaches and DTAlg is higher in case of ML. For BC, the values are almost identical. That is probably because of the difference in rating values in the two datasets.

Algorithm/No. of Ratings	500	1000	2000
DTAlg	1.2710	1.6670	1.7360
RF-CLUSTER	1.1750	1.3810	1.6340
RF-DESCRIPTION	2.2750	2.5230	2.6630
RF-SIZE	2.2950	2.5540	2.8240
RF-EMD	2.4040	2.5540	2.8610
RF-RAND	2.3860	2.6020	2.8290

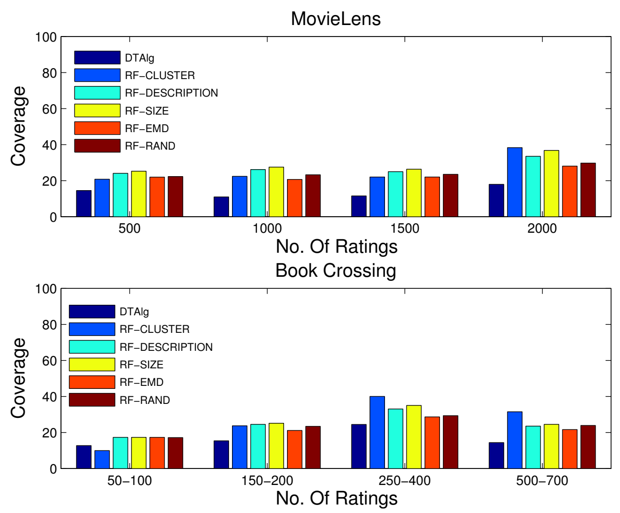
Table 6.4: Table describing how description length varies with input size on ML dataset

Table 6.4 shows the variation in average description length of groups obtained as the input size grows (ML dataset). `DTAlg` has the least description length compared to other `RF` approaches except `RF-Cluster`. Recall that the description of `RF-Cluster` is approximate. In general description length increases as input size increases, although not dramatically.

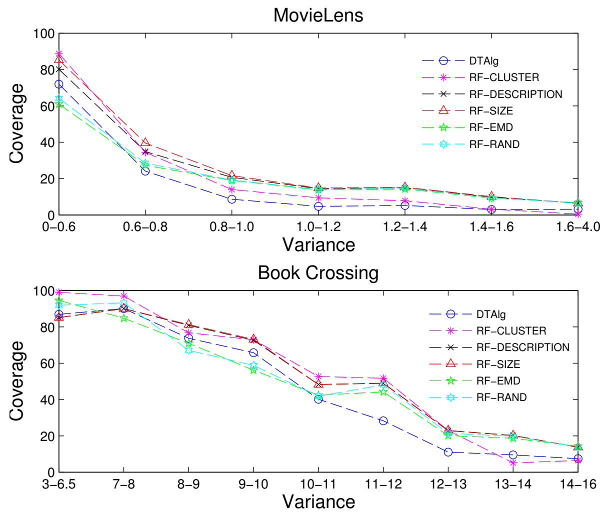
6.3 Scalability Evaluation

Figure 6.4(a) shows the running times of various algorithms as input size grows. For `DTAlg` the increase is unnoticeable. The time taken to create random trees is same for all `RF` approaches. The difference in running times is due to different strategies in combining partition from different trees. For `RF-Cluster`, though it has good quality in terms of coverage and diversity, its running time is very long. For large input sizes, it takes minutes. `RF-Size`, `RF-Rand`, `RF-EMD` have low running times. `RF-Description` time increases as input size grows as expected as finding groups which are far apart in terms of description is a costly step. Compared to previous work in finding groups [6] our algorithms are scalable.

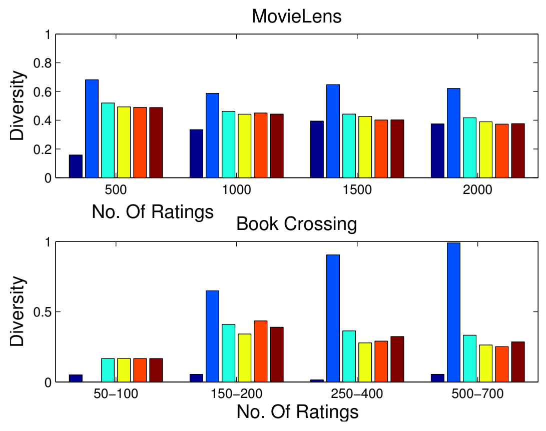
Since rating distribution length (rating scale) of groups is small, to observe the effect of pruning in `EMD` calculation algorithms, algorithm has to perform many such operations. This is only possible on very large datasets. In order to understand the effect of pruning, we compare time taken by `RF` approaches as the number of random trees grows since the number of `EMD` calculations increases. Effect of pruning can be seen from the plot. As the number of random trees grows, the gap between two curves increase which justifies that bounds can prove helpful when distributions are very long. Improvement is not very large. It is probably because, the constant factor in linear time algorithm also increases due maintenance of bounds.



(a)

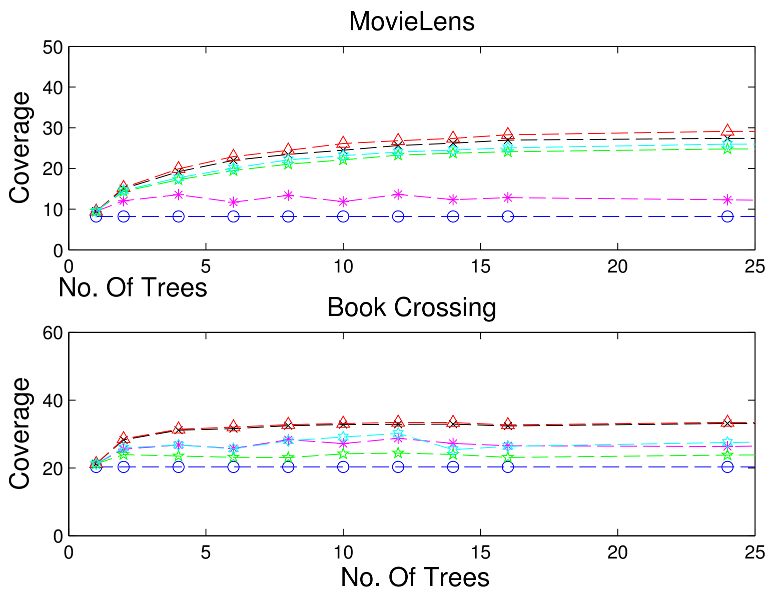


(b)

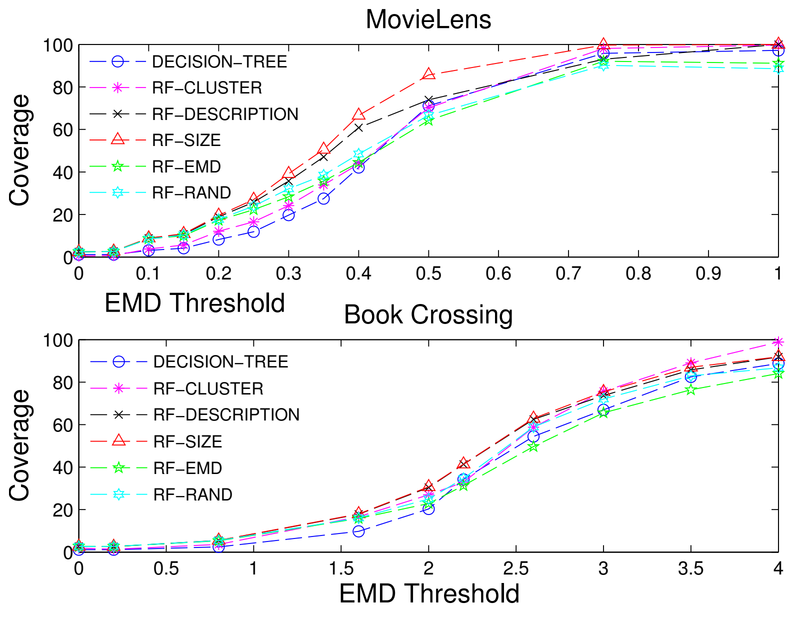


(c)

Figure 6.2: Figure showing plots of coverage and diversity in description for movie Lens and Book Crossing. (a)Coverage Vs. Input Size (b)Coverage Vs. Variance (c)Diversity Vs. Input Size(Plot legend is same as in (a))

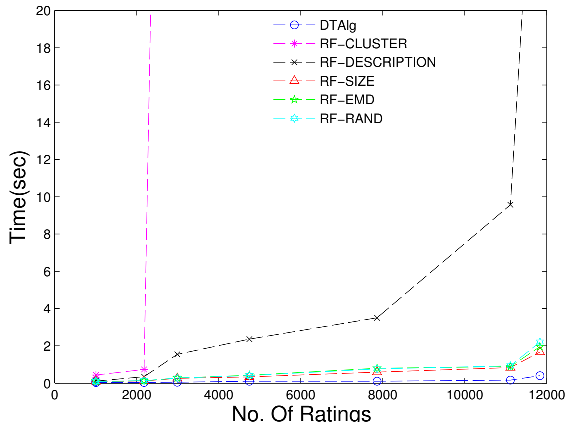


(a)

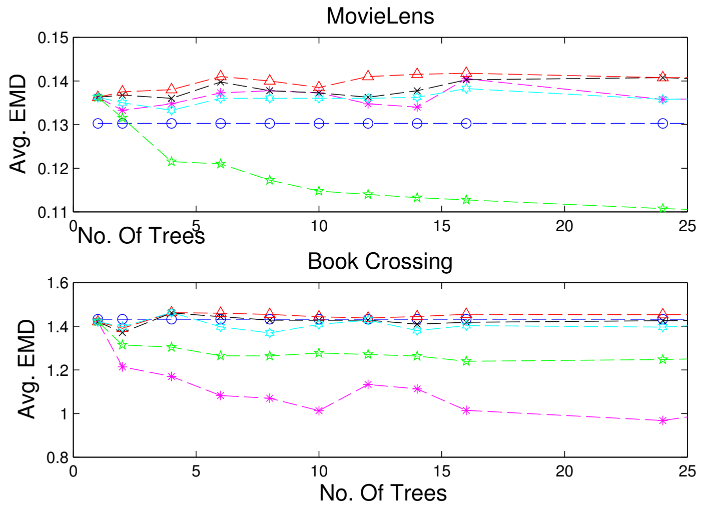


(b)

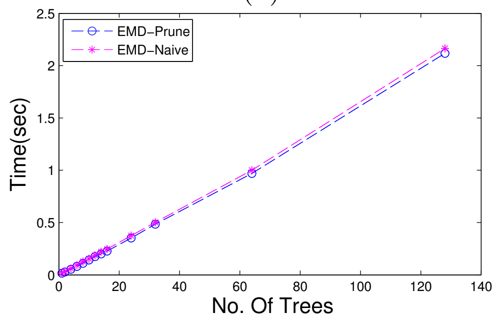
Figure 6.3: Figure showing plots of coverage for movie Lens and Book Crossing. (a)Coverage Vs. EMD Threshold (b)Coverage Vs. No. of Trees (Plot legend is same as in (a))



(a)



(b)



(c)

Figure 6.4: Figure showing plots of various experiments on both the datasets: (a) Time Vs. Input size (b)Avg. EMD Vs. No. of Trees (Plot legend is same as in (a)) (c) EMD-Naive vs EMD-Prune

Chapter 7

Related Work

Recently, the work of Das et al. [6] introduced complex mining of rated datasets with the goal of extracting meaningful demographic patterns that describe groups with biased opinions. They study the problem of finding groups that are uniform or polarized in their ratings. Our work can be seen as a generalization of their problems as we propose to query the rating distributions of an input dataset and discover describable groups that are close to input query distributions. Our query distributions could have any shape including uniform and polarized ones. Moreover, unlike ours, the algorithms proposed in that paper rely on data cubes. Exploring their applicability to our setting is an interesting future direction.

Some data-driven studies have indicated that the observed diversity of opinion can be the result of demographic groups reacting differently to external events. Choudhury et al. [5] examined opinion biases in blogosphere communities, relying on entropy measure as an indicator of diversity in opinions. Alternatively, Varlamis et al. [17] propose clustering accuracy as an indicator of the blogosphere opinion convergence. Using this measure, they detected a divergence in topics near few major events.

Several dimensionality reduction techniques, such as Subspace Clustering and Principle Component Analysis (PCA), were developed in order to describe a large structured dataset as labeled clusters. In particular, subspace clustering has been used extensively for data exploration in a variety of domains, see [9, 15] for reviews. While Subspace Clustering may be extended to handle the detection of groups in our setting, it needs to be modified to account for our rating distribution comparison measure (`EMD`) and for describability and scalability. CLIQUE [1] is the first bottom-up subspace clustering algorithm that relies on a global notion of density - the percentage of the overall dataset that falls within a particular subspace. ENCLUS [4] uses information entropy as the clustering objective, and shows a relationship between entropy and density, correlation, and coverage. Several extensions of the original algorithm were developed: CLTree [13] uses a decision-tree approach to identify high-density regions, while Cell-Based Clustering [3] improves scalability by partitioning the data so as to produce fewer clusters. Finally, PCA relies on pre-determining the set of attributes to use to describe clusters instead of discovering them on the fly, as in our work.

Chapter 8

Conclusion

8.1 Conclusion

In this paper, we present the first work that enables scalable exploration of rated datasets in a rating distribution-centric manner. Our solution returns a set of <user,item,rating> groups whose rating distribution is as close as possible to input query distributions. We show that the problem is NP-hard under two different settings and develop two heuristics: the first one is based on adapting decision trees and the second is based on the random forest approach. At the center of our work is the use of Earth Mover's Distance, a measure that intuitively captures the minimum amount of work required to transform one distribution into another. In future work, we would like to explore the relationship between different definitions of diversity and coverage and describability. We would also like to investigate the applicability of other algorithms such as subspace clustering for finding groups.

Bibliography

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data. *Data Min. Knowl. Discov.*, 11(1):5–33, 2005.

[2] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001. 10.1023/A:1010933404324.

[3] J.-W. Chang and D.-S. Jin. A new cell-based clustering method for large, high-dimensional data in data mining applications. In *SAC*, pages 503–507. ACM, 2002.

[4] C. H. Cheng, A. W.-C. Fu, and Y. Zhang. Entropy-based subspace clustering for mining numerical data. In *KDD*, pages 84–93, 1999.

[5] M. D. Choudhury, H. Sundaram, A. John, and D. D. Seligmann. Multi-scale characterization of social network dynamics in the blogosphere. In *CIKM*, pages 1515–1516, 2008.

[6] M. Das, S. Amer-Yahia, G. Das, and C. Yu. Mri: Meaningful interpretations of collaborative ratings. *PVLDB*, 4(11):1063–1074, 2011.

[7] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

[9] H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *TKDD*, 3(1), 2009.

[10] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[11] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:49–86, 1951.

[12] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, pages 106–115, 2007.

[13] B. Liu, Y. Xia, and P. S. Yu. Clustering through decision tree construction. In *Proceedings of the ninth international conference on Information and knowledge management*, CIKM '00, pages 20–29, New York, NY, USA, 2000. ACM.

[14] *MovieLens dataset, http://www.grouplens.org/data/*, as of 2003.

[15] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: a review. *SIGKDD Explorations*, 6(1):90–105, 2004.

[16] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40:99–121, 2000. 10.1023/A:1026543900054.

[17] I. Varlamis, V. Vassalos, and A. Palaios. Monitoring the evolution of interests in the blogosphere. In *ICDE Workshops*, pages 513–518, 2008.

Appendix A

Algorithm Optimality

In this section we prove that `EMD` calculation algorithm 4 is indeed optimal and correct. That is we prove the following statement:

Theorem 4 *Distance computed by algorithm 4 is optimal and is equal to* `EMD`

Suppose you we are calculating `EMD` between distributions ρ_s (source) and ρ_t(target). Algorithm 4 makes a pass over distribution ρ_s and keeps track of deficit and excess positions using a stack. As algorithm while keeping track of workdone, uses positions in the stack to move mass such that ρ_s is converted to ρ_t. Its a greedy algorithm because, it moves masses between nearest position (top element of stack)

A.1 Notations/Definitions

Before going into details of proof, we explain various notation used in the proof:

1. $\rho_s[i, j]$ - represents an array of probability values from i to j in ρ_s
2. $\rho_s[i : j] = \sum_{k=i}^{j} \rho_s[k]$, i.e sum of the probabilities for values i to j
3. In process of converting ρ_s to ρ_t, mass is flown between the positions. Such flows are represented using a flow matrix. F represent any flow matrix which convert ρ_s to ρ_t.
4. F^{opt} represent flow matrix of the optimal solution
5. G represent flow matrix obtained from greedy algorithm 4
6. If F is the flow matrix, $F_{ij} = F[i, j]$ =flow from position i to position j.
7. $W(F)$ represents work done due to flow F
8. Given ρ_s, ρ_t, the region $\rho_s[1, i]$ is

- `self-sufficient(SS)` if $\rho_s[1:i] = \rho_t[1:i]$
- `deficit` if $\rho_s[1:i] < \rho_t[1:i]$
- `excess` if $\rho_s[1:i] > \rho_t[1:i]$

9. $F[1,k][1,k]$ represents submatrix of F of size $k \times k$ and it has flows between position 1 to k

A.2 Lemmas & Facts

In this section we state lemmas and proofs which are useful in proving the theorem 4

Fact 5 *If region $\rho_s[1,l]$ is* `deficit` *then there must be inflow to this region from other positions. That is $\exists k, j$ s.t $F^{opt}_{kj} > 0, k > l, 1 \leq j \leq l$*

Fact 6 *If region $\rho_s[1,l]$ is* `excess` *then there must be outflow from this region to other positions. That is $\exists k, j$ s.t $F^{opt}_{jk} > 0, k > l, 1 \leq j \leq l$*

Lemma 2 *If region $\rho_s[1,l]$ is* `deficit or SS` *then in an optimal flow F^{opt}, there cannot be any out flow from this region. That is if $\rho_s[1,l]$ is* `deficit or SS` *then*

$$\forall F^{opt}, F^{opt}_{ik} = 0 \ \forall k > l \ \& \ 1 \leq i \leq l \tag{A.1}$$

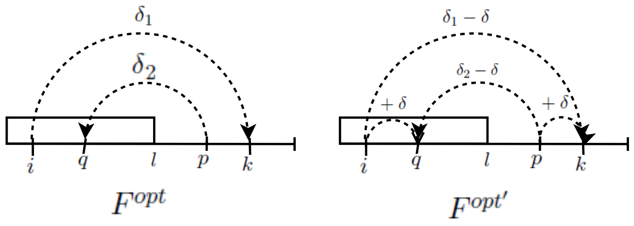


Figure A.1: Figure shows how to modify optimal flow F^{opt} to obtain new flow $F^{opt'}$

Proof: We use contradiction to prove that, such a flow does not exist. Let us suppose in optimal solution, there exists a flow such that $F^{opt}_{ik} = \delta_1 > 0$ s.t $k > l$. Since region $P[1,l]$ is `deficit or SS` , by fact 5 $\exists$ p, q *s.t* $F^{opt}_{pq} = \delta_2 > 0$, $p > l$ and $1 \leq q \leq l$. Now we show that, F^{opt} is not optimal flow W.l.og let $q > i$ and $k > p$. Similar contradiction can be shown for other three cases. Let $\delta = \min(\delta_1, \delta_2)$. Intuition here is to disturb the optimal flow such that total amount of work done decreases. Let the new Flow matrix obtained be $F^{opt'}$. Figure A.1 describes how optimal flow is modified to obtain new flow. The values in the $F^{opt'}$ are as follows:

$$\begin{aligned}
F^{opt'}_{ik} &= \delta_1 - \delta \\
F^{opt'}_{pq} &= \delta_2 - \delta \\
F^{opt'}_{pk} &= F^{opt}_{pk} + \delta \\
F^{opt'}_{iq} &= F^{opt}_{iq} + \delta
\end{aligned}$$

At other position, flow of $F^{opt'}$ are identical to flows of F^{opt}. Lets calculate the change in the work

$$\begin{aligned}
W_{F^{opt}} - W_{F^{opt'}} &= \delta_2(k-i) + \delta_1(p-q) \\
&\quad - [(\delta_2 - \delta)(k-i) + (\delta_1 - \delta)(p-q) \\
&\quad + \delta(q-i) + \delta(k-p)] \\
\triangle W &= \delta(k - i + p - q - q + i - k + p) \\
\triangle W &= 2\delta(p-q) \\
\triangle W &> 0
\end{aligned}$$

Thus change in the work is positive. It means, $W_{F^{opt}} > W_{F^{opt'}}$ which is a contradiction because F^{opt} is an optimal solution. Hence lemma is proved

Lemma 3 *If region $\rho_s[1,l]$ is excess or self-sufficient then in an optimal flow F^{opt}, there cannot be any in flow to this region. That is if $\rho_s[1,l]$ is* `excess or SS` *then*

$$\forall F^{opt}, F^{opt}[k,i] = 0 \ \forall k > l \ \& \ 1 \le i \le l \tag{A.2}$$

Proof: This lemma can be proved following similar steps as in proof of Lemma 2. So proof is not written in detail.

Corollary 7 *Given, ρ_s, ρ_t, there can either be an inflow or outflow from position 1, in optimal solution. More precisely:*

1. *$\rho_s[1] > \rho_t[1]$, then $F^{opt}[j,1] = 0 \ \forall j \neq 1$*
2. *$\rho_s[1] < \rho_t[1]$, then $F^{opt}[1,j] = 0 \ \forall j \neq 1$*

A.3 Proof

In this section we give proof of the theorem using lemmas and facts proved above. Recall that G represents the flows obtained from our greedy algorithm.

Theorem 8 *Let k be the maximum value such that,* $G[1,k] \geq 0$ *or* $G[k,1] \leq 0$. *If* F_{opt} *is an optimal flow matrix of* $\texttt{EMD}(\rho_s, \rho_t)$ *such that*

$$F^{opt} = \left[\begin{array}{c|c} F^{opt}[1,k][1,k] = A & F^{opt}[1,k-1][k+1,n] = B \\ \hline F^{opt}[k+1,n][1,k] = C & F^{opt}[k,n][k+1,n] = D \end{array}\right]$$

then

1. $F^{opt}[1,k-1][k+1,n] = B = 0$
2. $F^{opt}[k+1,n][1,k-1] = C = 0$
3. *Workdone due to flow* $W(F^{opt}[1,k][1,k])$ = *work done due to flow* $W(G[1,k][1,k])$
4. $F^{opt}[k,n][k,n]$ *is the optimal flow of another smaller sub-problem* $\texttt{EMD}(\rho_s^{'}[k,n], \rho_t^{'}[k,n])$

where $\rho_s^{'}[k] = \rho_s[k] - \sum_{j=1}^{k-1}(F^{opt}[j,k] + F^{opt}[k,j]), \rho_s^{'}[k+1,n] = \rho_s[k+1,n]$ *and* $\rho_t^{'}[k] = \rho_t[k] - \sum_{j=1}^{k-1}(F^{opt}[j,k] + F^{opt}[k,j]), \rho_t^{'}[k+1,n] = \rho_t[k+1,n]$

Proof: Theorem states that, if k is the farthest position from 1 such that there is either inflow or outflow between positions 1 and k in greedy solution, then 1,2,3,4 are true.
(1) & (2) : $\rho_s[1,k]$ is a `deficit or SS` region. Thus using lemma 2 $F^{opt}[1,k-1][k+1,n] = 0$. Similarly, then using lemma 3 $F^{opt}[k+1,n][1,k] = 0$

We use induction technique to prove statement (3) is true $\forall k = m, 1 \leq m \geq n$.

- $k = 1$: It implies $\rho_s[1] = \rho_t[1]$, thus there is no out-flow or in-flow at position 1. Thus $W(F^{opt}[1,1]) = W(G[1,1])$, proving the statement is true.

- $k = m - 1$: Let workdone due to $W(F^{opt}[1, m-1][1, m-1])$ is equal to workdone due to $W(G[1, m-1][1, m-1])$

- $k = m$: W.l.og let $G[1, m] > 0$. That is first position is excess mass position. Now idea is to move excess mass at position 1 to position 2 and use induction hypothesis to prove the statement. To be more precise, form a new sub problem, such that all flows from $\rho_s[1]$ appear as flows $\rho_s[2]$. That is move all the extra mass at from $\rho_s[1]$ to $\rho_s[2]$. Let resulting distribution be $\rho_s^{''}$. Length of $\rho_s^{''}$ is $m-1$. Using induction hypothesis one can say that statement (3) is true for corresponding optimal ($F^{opt''}$) and greedy ($G^{''}$) flow matrices of `EMD(`$\rho_s^{''}, \rho_t[2, n]$`)`. One can extend and modify flow matrice $F^{opt''}, G^{''}$ of EMD($\rho_s^{''}, \rho_t[2, n]$) to obtain flow matrices F^{opt}, G of EMD(ρ_s, ρ_t) by just adjusting flows at position one. It is easy to argue that such increase in workdone for both optimal and greedy flows is same amount and it is equal to workdone in moving at extramass at position 1 to 2. This proves that statement (3) is for F^{opt}, G.

(4) states that this problem exhibits optimal substructure property, which is essential for a greedy algorithm. That is Solution to `EMD(`$\rho_s^{''}, \rho_t^{''}$`)` is equal to $W(F^{opt}[k, n][k, n])$. Otherwise one can derive a contradiction. Proof of (4) completely shows that solution obtained by greedy algorithm is optimal.