EXPLORATORY MINING OF COLLABORATIVE SOCIAL CONTENT

by

MAHASHWETA DAS

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2013

To my Didi, who is the reason why I am in Computer Science

ACKNOWLEDGEMENTS

"*Silent gratitude isn't much use to anyone*"

- Gladys B. Stern

I would like to take this opportunity to thank a handful of people, without whom this dissertation would not have seen the light of day. This is just not a trite etiquette, but my fair chance of translating my heartfelt gratitude towards each of you to words.

My life is a living testimony to the saying: "*When God closes a door, He opens another*". I met my PhD advisor, Dr. Gautam Das at a time when life was unjust to me and I was looking for an anchor. Unquestionably, Dr. Das is the first person I would like to acknowledge here. The research in this dissertation would not have been possible without his emphatic encouragement, steadfast support, meticulous mentoring, and candid critiquing. He taught me everything about research - from problem insemination to technical formulation, from solution designing to result dissemination, from paper writing to conference presentation - and yet never constrained me. Just like a parent, he took me under his wings during my early PhD days and then gradually turned over the reins to help me brace the role of an independent and mature researcher.

On this note, I would like to express my deepest gratitude to all my mentors in the past four years: Dr. Sihem Amer- Yahia for her unflinching enthusiasm at each stage of my PhD journey; Dr. Vagelis Hristidis for hearing me out at all times and patiently responding to my research queries (however trivial); Dr. Cong Yu for partaking in extended research discourses with me amidst his busy schedule; Dr.

Aris Gionis for teaching me the significance of conscientiousness and perfection in performing a task; and Dr. Gagan Agrawal for leading me to the right career path. I would like to thank Dr. Hristidis, Dr. Vassilis Athitsos, and Dr. Chengkai Li for serving in my dissertation committee. I have spent great summers at IBM Research India, Technicolor Research USA, and Yahoo! Research Spain, and I am grateful to my hosts for offering me those amazing industry research experiences. Special thanks go out to my DBXLAB labmates Senjutidi, Saravanan, Habibur, and Azade. This dissertation is an outcome of my association with all these exceptional researchers.

I am grateful to the National Science Foundation for supporting my research through grants 0812601, 0915834, 1018865, as well as Texas Higher Education Coordinating Board, Microsoft Research and Nokia Research for providing financial supports throughout my graduate study. Any opinions, findings, and conclusions or recommendations expressed in this dissertation are those of the author and, if applicable, his advisor and collaborators, and do not necessarily reflect the views of the sponsors. I would also like to thank the Department of Computer Science and Engineering, University of Texas at Arlington for providing me the platform and resources to perform. On this note, my almae matres: South Point, where I spent the first fifteen years of my enriching student life; Jadavpur University, where I took my baby steps to pursuing a career in Computer Science; and Ohio State University, where I acquired academic research experience for the first time, deserve mention too.

It would have been a lonely experience without my friends - you know who you are :). All our conversations over coffee, Facebook, Gchat, and phone have kept me ticking. I would specially like to mention Senjutidi, Kausikda, Arnab, Nandish, and Upa who made my stay at Arlington more memorable than I could ask or imagine.

Finally, this token of acknowledgement cannot be complete without mentioning my family - my circle of love and life. I am indebted to Appa and Ma for instilling

in me the values of integrity, perseverance and self-respect, and the attitude of a champion. However, the greatest gift my parents have given me is my Didi - my big sister, my friend, and my inspiration, who is the reason why I am writing this manuscript. Words are not enough to express the debt of gratitude that I owe her. In spite of being just four years older than me, she endured the uncertainty and experimented with the unknown to ensure me a smooth and successful ride in life. She has been my "*Guardian Angel*" in the true sense of the term. A special thanks to my awesome brother-in-law Kdada for being the big brother I never had. I am glad that my "*Haate Khori*", i.e., introduction to research in Computer Science, happened through him. I believe I am the luckiest girl in this world to have an adoring husband like Abon in my life, whose unyielding support of my pursuits and unwavering trust towards me is one of a kind. He inspires me to "*lean in*" and holds me when I stumble. He is the perfect soulmate an aspiring woman-in-science-and-engineering can dream of and I wholeheartedly thank his parents and his alma mater for raising him to be the person he is today. This journey is as much his as is mine.

November 12, 2013

ABSTRACT

EXPLORATORY MINING OF COLLABORATIVE SOCIAL CONTENT

MAHASHWETA DAS, Ph.D.

The University of Texas at Arlington, 2013

Supervising Professor: Dr. Gautam Das

The widespread use and growing popularity of online collaborative content sites (e.g., Yelp, Amazon, IMDB) has created rich resources for consumers to consult in order to make purchasing decisions on various items such as restaurants, movies, e-commerce products, movies, etc. It has also created new opportunities for producers of such items to improve business by designing better products, composing interesting advertisement snippets, building more effective personalized recommendation systems, etc. This motivates us to develop a framework for exploratory mining of user feedback on items in collaborative social content sites. Typically, the amount of user feedback (e.g., ratings, reviews, tags) associated with an item (or, a set of items) can easily reach hundreds or thousands resulting in an overwhelming amount of information (information explosion), which users may find difficult to cope with (information overload). For example, popular restaurants listed in the review site Yelp routinely receive several thousand ratings and reviews, thereby causing decision making cumbersome. Moreover, most online activities involve interactions between multiple items and different users and interpreting such complex user-item interactions becomes intractable too. Our research concerns developing novel data mining

and exploration algorithms to formally analyze how user and item attributes influence user-item interactions. In this dissertation, we choose to focus on short user feedback (i.e., ratings and tags) and reveal how it, in conjunction with structural attributes associated with items and users, open up exciting opportunities for performing aggregated analytics. The aggregate analysis goal is two-fold: (i) exploratory mining to benefit content consumers make more informed judgment (e.g., if a user will enjoy eating at a particular restaurant), as well as (ii) exploratory mining to benefit content producers conduct better business (e.g., a redesigned menu to attract more people of a certain demographic group, etc.). We identify a family of mining tasks and propose a suite of algorithms - exact, approximation with theoretical properties, and efficient heuristics - for solving the problems. Performance evaluation over synthetic data and real data crawled from the web validates the utility of our framework and effectiveness of our algorithms.

RELATED PUBLICATIONS

• **Mahashweta Das**, Saravanan T., Sihem Amer-Yahia, Gautam Das, Cong Yu. *An Expressive Framework and Efficient Algorithms for the Analysis of Collaborative Tagging*. VLDB Journal Special Issue on Best Papers of VLDB 2012 - **VLDBJ 2013**.

• **Mahashweta Das**, Gianmarco De F. Morales, Aristides Gionis, Ingmar Weber. *Learning to Question: Leveraging User Preferences for Shopping Advice*. 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining - **SIGKDD 2013**.

• **Mahashweta Das**, Habibur Rahman, Gautam Das, Vagelis Hristidis. *Generating Informative Snippet to Maximize Item Visibility*. 22nd ACM Conference of Information and Knowledge Management - **CIKM 2013**.

• **Mahashweta Das**, Saravanan T., Sihem Amer-Yahia, Gautam Das, Cong Yu. *Who Tags What? An Analysis Framework*. 38th International Conference on Very Large Data Bases - **VLDB 2012**.

• Saravanan T., **Mahashweta Das**, Shrikant Desai, Sihem Amer-Yahia, Gautam Das, Cong Yu. *MapRat: Meaningful Explanation, Interactive Exploration and Geo-Visualization of Collaborative Ratings*. 38th International Conference on Very Large Data Bases - **VLDB 2012**.

• **Mahashweta Das**, Sihem Amer-Yahia, Gautam Das, Cong Yu. *MRI: Meaningful Interpretations of Collaborative Ratings*.37th International Conference on Very Large Data Bases - **VLDB 2011**.

• **Mahashweta Das**, Gautam Das, Vagelis Hristidis. *Leveraging Collaborative Tagging for Web Item Design*. 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining - **SIGKDD 2011**.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

Introduction

1.1 Collaborative Social Content

Social media and communication is an integral part of most people's everyday lives and practices. Though Facebook, Twitter, LinkedIn, and Google Plus are readily the most popular social networking sites and have several years of intense academic research dedicated to interesting data-rich projects involving them, the potential of social sites like Amazon, CNET, Yelp, IMDB, etc. is immense. For example, Amazon enables people to connect with each other and share information, opinion, reviews, etc. as well as provides platform to expand business reach, promote content, maximize brand values, etc. The rich resource of user-generated content in these sites, in the form of reviews, ratings, question-and-answer forums, blog posts, audio files, digital images, etc., offers unprecedented opportunities for rapid innovations. For example, LinkedIn, the most popular social networking site for people in professional occupations, acquired ChoiceVendor, a startup that allows companies to rate and review business-to-business service providers, in 2010 in order to tap into the latter's collection of peer-to-peer reviews and ratings. Such user-generated content is increasingly created through the collaborative efforts of multiple individuals interacting with the social media. Hence, we refer to these sites as *Collaborative Social Content sites*.

A typical collaborative site consists of three components: (i) *users*, (ii) *items*, and (iii) *user feedback over items* (i.e., user-item interactions). Users are one of the most significant components in a collaborative site and they exist either as content consumers or as content producers. Users provide information about their demo-

graphics, interests, and other details in order to maximize the utility and benefits from the site. The second component of collaborative sites is items. An item is any object shared over the Web and can either be user generated (e.g., photos, blogs, videos, etc), or can be produced by a product designer (e.g., cameras, apparels, etc). The third component of collaborative content sites is user feedback. Most websites today encourage users to leave feedback for online resources with a purpose to promote their contents and allow users to share, discover and organize them. User feedback can be either short and cryptic (e.g. numeric ratings, keywords or tags associated to items, check-ins, etc.) or long and detailed (e.g., reviews in free-form text). Detailed reviews can be very informative and the problem of extracting semantic information from such reviews is a well-known problem in text mining and information extraction. Lately, user feedback is available in the form of images, audios, and videos too. Typically, the amount of user feedback associated with an item (or, a set of items) can easily reach hundreds or thousands. For example, the movie "The Social Network" has received 42,000+ ratings on IMDB after being released for just two months. Similarly, on the review site Yelp, a not-so-popular restaurant "Joe's Shanghai" received nearly a thousand ratings and more popular restaurants routinely exceed that number by many multipliers. In fact, over 100 million unique visitors came to Yelp and over 1 million online reviews were posted on Yelp in January, 2013 alone![1].

Examples of popular Collaborative Social Content sites: Amazon, CNET, eBay, Yahoo! Shopping, Walmarts, Epinions for e-commerce; Yelp, UrbanSpoon for restaurants; TripAdvisor, Expedia for vacation packages; IMDB, Netflix for movies; Last.fm for music; Goodreads for books; Flickr for photos; Youtube for videos; Newsvine for news articles; etc.

[1]http://www.reviewtrackers.com/100-million-visitors-yelp-records-all-time-high-traffic-one-month/

ID	Item			User			Feedback	
	Title	**Genre**	**Director**	**Name**	**Gender**	**Location**	**Rating**	**Review**
1		Drama	James Cameron	Amy	Female	California	7.5	It is a romantic adventure with visual grandeur and magnificence, a timeless tragic love story set against the backdrop of a major historical event. Powerful movie, it deserves all the Oscars.....
2		Drama	Steven Speilberg	John	Male	New York	9.0	If you want to see one of the greatest historical films ever made, then go see this. Spielberg rightly won his Oscar for this.....

Table 1.1. Example of collaborative social content expressed in a structured format

Our research impacts and benefits people (content producer or consumer) associated with all these sites. However, due to limited availability of collaborative social content data from these sites, we limit our experimental evaluation to few publicly available real data and large-scale crawls of few publicly available websites. Table 1.1 presents an example structured form of collaborative content from popular movie review site IMDB. The Rating is in a scale of 10. The phrases in blue in the Review column may be extracted as tags using state-of-art text mining and information retrieval techniques. The number of item features and user features are much more than the few (Genre, Director and Gender, Location) shown in the table.

1.2 Motivation and Challenges

The widespread use and growing popularity of these collaborative content sites has created rich resources for consumers to consult in order to make purchasing decisions on various items. For example, a cell phone that has been tagged `lightweight` by several users is likely to influence a prospective consumer decision in its favor. It has also created new opportunities for producers of such items to improve business by designing better products, composing eye-catching advertisement snippets, dealing with competition, providing more effective personalized recommendations, etc. For example, if an Europe backpacking trip package has received the tags `single`, `adventure`, and `budget` from several customers, the travel agency must highlight the relevant features youth-hostel, free city-attractions, and Eurail Youth Pass in its advertisement in order to draw the attention of interested future users. Not just in traditional marketplaces, a musical artist can leverage available user feedback for her past tracks to decide the features (e.g., genre, acoustic and audio features) of her new musical piece in order to increase its chances of becoming popular; or a blogger can select a topic based on the tags that other popular topics have received. A huge number of articles have invaded the Internet lately conveying how social media initiatives hold the key to fueling business. This motivates us to develop a framework for mining user feedback on items in collaborative social content sites in order to benefit experience and decision-making of both content producers and consumers. We formally analyze how user and item attributes influence user-item interactions.

As mentioned before, the amount of user feedback associated with an item (or, a set of items) can easily reach hundreds or thousands resulting in an overwhelming amount of information (*information explosion*), which users may find difficult to cope with (*information overload*). The number of *fraudulent users and user feedback* in the web is on the rise. Moreover, most online activities involve interactions between mul-

tiple items and different users and interpreting such *complex user-item interactions* becomes intractable too.

1.3 Dissertation Overview and Impact

Our research concerns developing novel data mining and exploration algorithms, that accounts for the above-mentioned challenges. We identify a family of *exploratory mining tasks* and develop a general framework for collaborative social content mining, the *first of its kind*. In this dissertation, we choose to focus on short user feedback because we believe that: (i) even though a single short feedback carries relatively less information content, the aggregated view over numerous such user-item interactions can be very useful for decision-making, and (ii) the majority of all user-item interactions today are short feedback since their simpler mechanisms (e.g., assigning star ratings) make user participation very easy, compared to detailed reviews which require significant amount of time and effort from the authors as well as the readers. For sites lacking the availability of user feedback in the form of ratings and tags, we employ state-of-art text mining techniques to extract phrases or keywords (i.e., tags) from reviews. Such user feedback over items, in conjunction with other structural attributes associated with items and users in online collaborative content sites (e.g., item attributes such as Cuisine, Ambience, or user attributes such as Age, Gender), open up exciting opportunities for performing aggregated analytics over the data in order to benefit experience and decision-making of content producers and consumers. The aggregate analysis goal is two-fold:

1 Exploratory mining to benefit content consumers make more informed judgment (e.g., if a user will enjoy eating at a particular restaurant)

2 Exploratory mining to benefit content producers conduct better business (e.g., a re-designed menu to attract more people of a certain demographic group)

The objective is to model the correlation between user feedback and item attributes and user attributes and leverage such attribute-feedback aggregations to drive a variety of applications. The first goal draws motivation from the vast literature in automatic review mining and summarization and aims to help potential buyers make more efficient and rational purchase decisions. The second goal is a more novel endeavor that aims to mine user feedback for serving the content producers (i.e., manufacturers, retailers, etc.). We provide a principled approach to automatically mine user generated content in order to boost business, as opposed to the many blogs, articles, magazines, etc. dedicated to connecting the dots between social media and e-commerce. Note that, the goal of exploratory mining to benefit content producers improve business is beneficial to content consumers; and vice versa.

1.3.1 Exploratory Mining Problems

The mining problems investigated in this dissertation under the two aggregate analysis goals and their *impacts* are as follows:

I. Exploratory mining of collaborative social content for content consumers:

i Collaborative Rating Behavior Analysis: In this problem, we aim to meaningfully explain ratings as a function of metadata associated with items and reviewers, in order to help future consumers quickly decide the desirability of an item. For example, given the movie "Toy Story", instead of simply showing the average rating of 8.3/10 from 350,000+ users, we return a set of meaningful factoids such as "young male students in California `love` this movie". Note that unlike unsupervised clustering approaches, groups returned by our problem are meaningful due to the common structural attribute value pairs shared by all reviewers in each group. We also build a system that visualizes the rating interpretations for popular movies on a map and facilitates exploration over different dimensions.

ii Collaborative Tagging Behavior Analysis: In this problem, we extend our previous task to handle tag-based feedback since analyzing how tags are assigned by certain users to certain items has greater implications in helping users search for desired information. We develop a general mining framework that encompasses many social tagging behavior analysis tasks (can handle greater than 100 problems in the same framework) and reveals interesting insights, in order to help consumers use that knowledge in subsequent actions. As in (i), we build a system for interactive mining and exploration of social tagging behavior, as well as systematic geo-visualization of the findings over a map.

II. Exploratory mining of collaborative social content for content producers:

i New Item Design: In this problem, we leverage available user feedback for designing new improved items that are likely to attract maximum positive response when published. The problem of item design has been studied by many disciplines including Economics and Industrial Engineering. Optimal item design or positioning is a well studied problem in Operations Research and Marketing. However, none of these works has studied the problem of item design in relation to social user behavior (i.e., feedback). Our work investigates this novel problem, i.e., how to decide the specifications of new items so that they draw the highest expected number of desirable tags.

ii Item Snippet Generation: In this problem, we leverage available user feedback (as in the previous task) for generating informative advertisement snippets that are likely to maximize an item's visibility among buyers. Succinct summarization of an item's descriptions that provides its first impression is referred to as snippet. We envision the utility of a dynamic snippet as opposed to the pre-defined and static item description summaries (e.g., in faceted navigation of Amazon and

eBay), in order to match a user's search query effectively. We investigate the problem of finding the top-k best snippets for an item that are likely to maximize the probability that the user preference (available in the form of search query) is satisfied. Since a search query returns multiple relevant items, we also study the problem of finding the best diverse set of snippets for the items in order to maximize the probability of a user liking at least one of the top items.

iii Technical Item Recommendation: In this problem, we consider available user feedback (as in the previous tasks) to understand user lifestyle preferences and help non-expert buyers shop for technical products (e.g., camera, laptop, etc.). Note that, buying such technical products online is not an easy task for an average shopper since she cannot rely on the guidance of a in-store shopping assistant and is exepected to understand the technical specifications associated with the product. In this task, we build a system that translates, for example, laptop details like "at least 4 GB of RAM" and "at least 500 GB of hard drive" to questions like "do you intend to use the laptop for playing games?" and "do you want to store a lot of movies in your laptop?", etc.

We formalize the problems and design a suite of algorithms - exact, approximation with theoretical properties, and efficient heuristics - for solving them. Performance evaluation over synthetic data and real data crawled from Amazon, IMDB, Yahoo! Autos, etc. validates the effectiveness of our algorithms and utility of our framework.

1.4 Dissertation Organization

The first chapter introduces the overview of mining collaborative social content. We introduce the set of exploratory mining problems and discuss its impact. The research in this dissertation has two primary objectives (i.e., parts), each of which

encompasses a set of challenging problems. We distribute the exploratory mining problems across the different chapters in the following way:

In Chapter 2, we introduce the problem of Collaborative Rating Behavior Analysis [1][2]. We define the notion of meaningful interpretation of collaborative ratings based on the idea of data cube and formalize two important sub-problems, meaningful description mining and meaningful difference problem. We prove that the problems are NP-Complete and design randomized hill exploration algorithms to solve them efficiently. Performance evaluation over real data shows that our algorithms perform much faster and generate equally good interpretations as brute-force algorithms. We also conduct user studies to validate that users overwhelmingly prefer meaningful rating explanations over simple average ratings.

In Chapter 3, we introduce the problem of Collaborative Tagging Behavior Analysis [3][4]. We develop a dual mining framework to explore tagging behavior. This framework is centered around two opposing measures: similarity and diversity, applied to one or more tagging components: users, items and tags, and therefore enables a wide range of analysis scenarios (such as characterizing similar users tagging diverse items with similar tags) and they are NP-Complete in general. We design four sets of efficient algorithms for solving many of those problems and demonstrate, through comprehensive experiments over real data, the superiority of our algorithms over their baseline counterparts.

In Chapter 4, we introduce the problem of New Item Design [5]. Given a training dataset of existing items with their user-submitted tags and a query set of desirable tags, the objective is to design the k best new items expected to attract the maximum number of desirable tags. We show that this problem is NP-Complete, even if simple Naive Bayes Classifiers are used for tag prediction. We present an exact two-tier

algorithm (based on top-k querying techniques) for moderate problem instances and a novel polynomial-time approximation algorithm with provable error bound for larger problem instances. We conduct detailed experiments on synthetic and real data crawled from the web to evaluate the efficiency and quality of our algorithms.

In Chapter 5, we introduce the problem of Item Snippet Generation [6]. We exploit the availability of user feedback in collaborative content sites in the form of tags to identify the salient item features that must be highlighted in the item's snippet, in order to help a user quickly discover the items she is interested in. Since a search query returns multiple items, we also study the problem of finding the best diverse set of snippets for the items in order to maximize the probability of a user liking at least one of the top items. We develop an exact top-k algorithm and an approximation algorithm for each of the problems and perform detailed experiments on synthetic and real data crawled from the web to to demonstrate the utility of our problems and effectiveness of our solutions.

In Chapter 6, we introduce the problem of Technical Item Recommendation [7]. We present ShoppingAdvisor, a novel recommender system that helps users in shopping for technical products by leveraging both user preferences and technical product attributes. The system elicits user preferences via a tree-shaped flowchart, where each node is a question to the user. At each node, ShoppingAdvisor suggests a ranking of products matching the preferences of the user and that gets progressively refined along the path from the tree's root to one of its leafs. First, we adapt the classical top-down strategy for building decision trees (by partitioning the user space rather than the product space) in order to find the best user attribute to ask at each node. Second, we show how to employ a learning-to-rank approach in order to learn, for each node of the tree, a ranking of products appropriate to the users who reach that node.

We experiment with both synthetic and real data to demonstrate that it produces good quality interpretable recommendations.

Chapter 7 discusses related work in the literature that focuses on (i) sub-areas reasonably aligned to our research of exploratory mining of collaborative content such as review mining, recommendation, etc., as well as (ii) techniques which we have adapted to solve some of our problems such as top-k algorithms, locality sensitive hashing, etc.

Chapter 8 summarizes the contributions made in this dissertation. It also outlines the research frontiers in achieving the bigger goal of a general collaborative content mining framework that that caters to any community featuring two-way communication between producers and consumers (e.g., manufacturer and buyer, healthcare service provider and patient, musical artist and listener, blogger and reader, etc.).

PART I

EXPLORATORY MINING OF COLLABORATIVE CONTENT FOR CONTENT CONSUMERS

"*A brand is no longer what we tell the consumer it is;*
it is what consumers tell each other it is."
- Scott D. Cook

CHAPTER 2

Collaborative Rating Behavior Analysis

2.1 Introduction

Collaborative rating sites drive a large number of decisions today. For example, online shoppers rely on ratings on Amazon to purchase a variety of goods such as books and electronics, and movie-goers use IMDB to find out about a movie before renting it. The number of ratings associated with an item (or a set of items) can easily reach hundreds or thousands, thus making reaching a decision cumbersome. To cope with the overwhelming amount of information, a user can either spend a lot of time examining ratings and reviews before making an informed decision (*maximalist option*), or simply go with overall rating aggregations, such as average, associated with an item (*minimalist option*). Not surprisingly, most users choose the latter due to lack of time and forgo the rich information embedded in ratings and in reviewers' profiles. Typically, average ratings are generated for a few pre-defined populations of reviewers (e.g., average among movie critics). In addition, aggregated ratings are only available for one item at a time, and therefore a user cannot obtain an understanding of a set of items of interest, such as all movies by a given director.

In this task, we aim to help users make better decisions by providing *meaningful interpretations* of ratings of items of interest, by leveraging metadata associated with items and reviewers in online collaborative rating sites. We call this problem *meaningful rating interpretation* (MRI), and define two sub-problems: *meaningful description mining* (DEM) and *meaningful difference mining* (DIM).

Given a set of items, the first problem, *meaningful description mining*, aims to identify groups of reviewers who share similar ratings on the items, with the added

constraint that each group consists of reviewers who are describable with a subset of their attributes (i.e., gender, age, etc.). The description thus returned to the user contains a small list of meaningfully labelled groups of reviewers and their ratings about the item, instead of a single monolithic average rating. This added information can help users judge items better by surfacing inherent reviewers' biases for the items. For example, the movie "Titanic" may have a very high overall average rating, but it is really the group of female reviewers under the age of 20 who give it very high ratings and raise the average. A user can then make informed decisions about items based on whether she tends to agree with that group.

The second problem, *meaningful difference mining*, aims to help users better understand controversial items by identifying groups of reviewers who consistently disagree on those items, again with the added constraint that each group is described with a meaningful label. For the movie "Titanic", we can see that two groups of reviewers, females under 20 and males between 30 and 45, are in consistent disagreement about it: the former group loves it while the latter does not.

We emphasize that while the examples above all involve a single item, both description mining and difference mining can be applied to a set of items with a common feature. For example, we can apply them to all movies directed by "Woody Allen" and help users learn some meaningful trends about Woody Allen as a director. The algorithms we describe in this section apply equally whether we are analyzing the ratings of a single item or a set of items.

2.2 Problem Framework

We model a collaborative rating site $\mathcal{D}$ as a triple $\langle \mathcal{U}, \mathcal{I}, \mathcal{R} \rangle$, representing the reviewers (i.e., users), sets of items (i.e., products), and feedback (i.e., ratings) respectively. Each rating, i.e., feedback $r \in \mathcal{R}$ is itself a triple $\langle u, p, s_{up} \rangle$, where $u \in \mathcal{U}$,

$p \in \mathcal{I}$, and $s_{up} \in [1,5]$ is the integer rating that reviewer u has assigned to item p[1]. Furthermore, $\mathcal{I}$ is associated with a set of attributes, denoted $\mathcal{I}_A = \{pa_1, pa_2, \ldots\}$, and each item $p \in \mathcal{I}$ is a tuple with $\mathcal{I}_A$ as its schema. In another word, $p = \langle pv_1, pv_2, \ldots\rangle$, where each pv_j is a value for attribute pa_j. Similarly, we have the schema $\mathcal{U}_A = \{ua_1, ua_2, \ldots\}$ for reviewers, i.e., $u = \langle uv_1, uv_2, \ldots\rangle \in \mathcal{U}$, where each uv_j is a value for attribute ua_j. As a result, each rating action, $r = \langle p, u, s_{up}\rangle$, is a tuple, $\langle uv_1, uv_2, \ldots, pv_1, pv_2, \ldots, s_{up}\rangle$, that concatenates the tuple for p, the tuple for u, and the numerical rating score s_{up}. The set of all attributes (including both item and reviewer attributes) is denoted as $A = \{a_1, a_2, \ldots\}$.

Item attributes are typically provided by the rating site. For example, restaurants on `Yelp` are described with attributes such as Cuisine (e.g., Thai, Sushi), Attire (e.g., Formal, Casual). Movies on IMDB are described with Title, Genre (e.g., Drama, Animation), Actors, Directors. An item attribute can be multi-valued (e.g., a movie can have many actors). Reviewer attributes include mostly demographics such as Age, Gender, Location and Occupation. Such attributes can either be provided to the site by the reviewer directly as in MovieLens, or obtained from social networking sites such as Facebook as their integration into content sites becomes increasingly common. In this section, we focus on item ratings describable by reviewer attributes. Our ideas can be easily extended to explain reviewer ratings by item attributes.

We model the notion of *group* based on data cube [8]. Intuitively, a group is a set of ratings described by a set of attribute value pairs shared among the reviewers and the items of those ratings. A group can also be interpreted as a selection query condition. More formally, a group description is defined as $c = \{\langle a_1, v_1\rangle, \langle a_2, v_2\rangle, \ldots\}$, where each $a_i \in A$ (where A is the set of all attributes as introduced earlier) and each v_i is a value for a_i. For example, $\{\langle \mathsf{genre}, \mathsf{war}\rangle, \langle \mathsf{location}, \mathsf{nyc}\rangle\}$ describes a group

[1]For simplicity, we convert ratings at different scales into the range $[1, 5]$.

representing all ratings of "war" movies by reviewers in "nyc." The total number of groups that can exist is given by $n = \prod_{i=1}^{|A|}(|\langle a_i, v_j\rangle| + 1)$, where $|A|$ is the cardinality of the set of attributes and $|\langle a_i, v_j\rangle|$ is the number of distinct v_j values each attribute a_i can take. When the ratings are viewed as tuples in a data warehouse, this notion of group coincides with the definition of cuboids in the data cube literature. Here, we take the view that, unlike unsupervised clustering of ratings, ratings grouped this way are much more meaningful to users, and form the foundation for meaningful rating interpretations. We now define three essential characteristics of the group.

Definition 2.2.1. *Coverage: Given a rating tuple* $r = \langle v_1, v_2, \ldots, v_k, s\rangle$*, where each* v_i *is a value for its corresponding attribute in the schema* A*, and a group* $c = \{\langle a_1, v_1\rangle, \langle a_2, v_2\rangle, \ldots, \langle a_n, v_n\rangle\}, n \leq k$*, we say* c *covers* r*, denoted as* $r \lessdot c$*, iff* $\forall i \in [1, n]$*,* $\exists r.v_j$ *such that* v_j *is a value for attribute* $c.a_i$ *and* $r.v_j = c.v_i$*. For example, the rating* $\langle$*female*, *nyc*, *cameron*, *winslet*, $4.0\rangle$ *is covered by the group* $\{\langle$*gender*, *female*$\rangle, \langle$ *location*, *nyc*$\rangle, \langle$*actor*, *winslet*$\rangle\}$*.*

Definition 2.2.2. *Relationship between groups: A group* c_1 *is considered an ancestor of another group* c_2*, denoted* $c_1 \supset c_2$*, iff* $\forall j$ *where* $\langle a_j, v_j\rangle \in c_2$*,* $\exists\langle a_j, v'_j\rangle \in c_1$*, such that* $v_j = v'_j$*, or* v'_j *semantically contains* v_j *according to the domain hierarchy. For example, the group of ratings* g_1 *by reviewers who live in Michigan is a parent of the group of ratings* g_2 *by reviewers who live in Detroit, since Detroit is located in Michigan according to the location hierarchy*[2]*.*

Definition 2.2.3. *Recursive coverage: Given a rating tuple* r *and a group* c*, we say* c *recursively covers* r *iff* $\exists c'$*, such that,* $c \supset c', r \lessdot c'$*. For example,* $\langle$*female, nyc*, *cameron*, *winslet*, $4.0\rangle$ *is recursively covered by* $\{\langle$*gender, female*$\rangle, \langle$*location, usa*$\rangle, \langle$*actor, winslet*$\rangle\}$*. For the rest of the section, we use the term coverage to mean recursive coverage for simplicity, unless otherwise noted.*

[2]Those domain hierarchies are essentially dimension tables and we assume they are given.

2.2.1 Meaningful Rating Interpretation

When the user is exploring an item (or a set of items) I, our goal is to meaningfully interpret the set of ratings for I, denoted R_I. Given a group c, the set of ratings in R_I that are covered by c are denoted as $c_{R_I} = \{r | r \in R_I \wedge r \ll c\}$. Similar to data cubes, the set of all possible groups form a lattice of n nodes, where the nodes correspond to groups and the edges correspond to parent/child relationships. Note that, for a given I, there are many groups not covering any rating from R_I. Let n' denote the total number of groups covering at least one rating. Solving the MRI problem is therefore to *quickly identify "good" groups that can help users understand ratings more effectively.*

Before introducing the problem formally, we first present a running example, shown in Figure 2.1, which will be used throughout the rest of the section.

Example 2.2.1. Consider the use case where we would like to explain all ratings of the movie (item) "Toy Story", by identifying describable groups of reviewers sharing common rating behaviors. As in data cube analysis, we adopt a lattice structure to group all ratings, where each node in the lattice corresponds to a group containing rating tuples sharing the set of common attribute value pairs, and each edge between two nodes corresponds to the parent/child relationship. Figure 2.1 illustrates a partial lattice for "Toy Story", where we have four reviewer attributes to analyze[3]: gender (G), age (A), location (L) and occupation (O). For simplicity, exactly one distinct value per attribute is shown in the example: ⟨gender, male⟩, ⟨age, young⟩, ⟨location, ca⟩ and ⟨occupation, student⟩. As a result, the total number of groups in the lattice is 16. Each group (i.e., node in the lattice) maps to a set of rating tuples that are satisfied by the selection condition corresponding to the group label, and the numeric values within each group denotes the total number of ratings and the average rating within the

[3]Since there is only one movie in this example, item attributes do not apply here.

group. For example, the base (bottom) group corresponds to all 452 ratings of "Toy Story", with an average rating of 3.88, while the double circled group in the center of the lattice corresponds to the 75 ratings provided by 'male & student' reviewers, who collectively gave it an average rating of 3.76. □

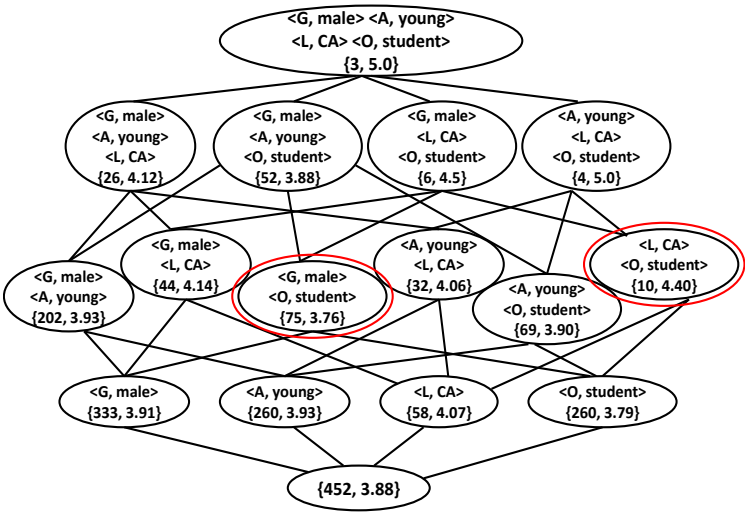


Figure 2.1. Partial rating lattice for movie "Toy Story" with one value for each attribute; the full lattice contains more nodes with multiple distinct values for each attribute.

Even when there is only a single item, the number of groups associated with its ratings can be too large for a user to browse. The challenge is therefore to identify *good* groups to be highlighted to the user. We define desiderata that such good groups should follow:

Desiderata 1: Each group should be easily understandable by the user. While this desiderata is often hard to satisfy through unsupervised clustering of ratings, it is easily enforced in our approach since each group is structurally meaningful and has an associated description that the user can understand.

Desiderata 2: Together, the groups should cover enough ratings in R_I. While ideally we would like all ratings in R_I to be covered, it is often infeasible given the constraint on the number of groups that a user can reasonably go through.

Desiderata 3: Ratings within each group should be as consistent as possible, i.e., should reflect users with similar opinions toward the input item(s). Note that we are referring to opinions within a group instead of opinions across groups. In fact, difference in opinion across groups is the key differentiator between the two sub-problems of MRI, which we will formally define in the next section.

We now formally define the two sub-problems of meaningful rating interpretation: *meaningful description mining* (DEM) and *meaningful difference mining* (DIM).

2.2.2 Meaningful Description Mining

Our first goal is to give a meaningful description of all the ratings over an item set I. We propose to present to the users a small set of meaningfully labelled rating groups (i.e., cuboids), each with their own average ratings. Specifically, we consider three main factors. First, the *number of cuboids*, k, to be presented to the user must be limited, so that users are not overwhelmed with too many cuboids. Second, all cuboids presented to the user must collectively *cover a large enough portion of ratings* for items in I. Third, the returned cuboids must collectively have the *minimum aggregate error*, which we will define next.

Consider a set of ratings R_I over input items in I. For each cuboid c, let $\texttt{avg}(c) = avg_{r_i \ll c}(r_i.s)$ (where $r_i.s$ is the score of the i^{th} tuple) be the average numerical score of ratings covered by c. Given a set of cuboids, C, to be returned to the user. We define two formal notions:

Definition 2.2.4. *Description coverage: Let* $C_{R_I} = \{r \mid r \in R_I, \exists c \in C, s.t.\ r \ll c\}$, $\texttt{coverage}(C, R_I) = \frac{|C_{R_I}|}{|R_I|}$.

Definition 2.2.5. *Description error: Let* $E_r = \texttt{avg}(|r.s - \texttt{avg}_{c \in C \wedge r \ll c}(c)|)$, $\texttt{error}(C, R_I) = \Sigma_{r \in R_I}(E_r)$.

Intuitively, *description coverage* measures the percentage of ratings covered by at least one of the returned cuboids, while *description error* measures how well the group average approximates the numerical score of each individual rating. (When a rating is covered by more than one returned cuboids, we average the errors over all the cuboids that cover the rating.)

2.2.3 Meaningful Difference Mining

Another important goal of rating interpretation is to identify meaningful groups of ratings where reviewers' opinions on the item(s) are divergent. To accomplish this goal, we start by dividing R_I into two sets $R_I^+ = \{r \mid r \in R_I \wedge r.s \geq \theta^+\}$ and $R_I^- = \{r \mid r \in R_I \wedge r.s \leq \theta^-\}$, where θ^+ and θ^- are thresholds that define whether a rating should be considered positive or negative respectively. Intuitively, θ^+ and θ^- can either be decided statically or dynamically according to the mean and variances of R_I. While setting the thresholds statically is easier computationally, it is not always clear what the thresholds should be. As a result, we follow the dynamic approach and set θ^+ and θ^- to be one standard deviation above and below the mean of R_I respectively.

Given R_I^+, R_I^- and a set of cuboid groups, C, we can now formalize the notion of *balance* as follows:

Definition 2.2.6. *Balance: Let indicator* $I_{(r_1,r_2)} = 1$ *if and only if* $\exists c \in C$ *s.t.* $r_1 \preccurlyeq c \wedge r_2 \preccurlyeq c$ *(i.e., there is at least one cuboid in* C *that covers both* r_1 *and* r_2*.). We then have* $\texttt{balance}(C, R_I^+, R_I^-) = m \times \Sigma_{r_1 \in R_I^+, r_2 \in R_I^-} I_{(r_1,r_2)}$*, where* $m = \frac{1}{|R_I^+| \times |R_I^-|}$ *is the normalization factor that normalizes all balance values into* $[0, 1]$.

Intuitively, the notion of *balance* captures whether the positive and negative ratings are "mingled together" (high balance) or "separated apart" (low balance).

Problem 2.2.1. *The problem of meaningful difference mining (DIM) for a given set of items I and their ratings R_I (split into R_I^+, R_I^-), identify a set of cuboids C, such that:*

- $\texttt{balance}(C, R_I^+, R_I^-)$ *is minimized, subject to:*
  - $|C| \leq k$;
  - $\texttt{coverage}(C, R_I^+) \geq \alpha \wedge \texttt{coverage}(C, R_I^-) \geq \alpha$.

2.3 Complexity Analysis

In this section, we analyze the computational complexity of the description mining and difference mining problem.

Problem 2.3.1. *The problem of meaningful description mining (DEM) for a given set of items I and their ratings R_I, identify a set of cuboids C, such that:*

- $\texttt{error}(C, R_I)$ *is minimized, subject to:*
  - $|C| \leq k$;
  - $\texttt{coverage}(C, R_I) \geq \alpha$.

Theorem 2.3.1. *The decision version of the problem of meaningful description mining (DEM) is NP-Complete even for boolean databases, where each attribute ia_j in $\mathcal{I}_A$ and each attribute ua_j in $\mathcal{U}_A$ takes either 0 or 1.*

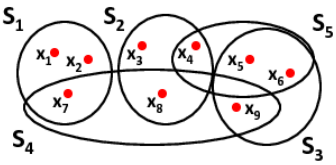
Proof : The decision version of the problem of meaningful description mining (DEM) is: For a given set of items and their ratings R_I, is there a set of cuboids C, such that $\texttt{error}(C, R_I) \leq \beta$, subject to $|C| \leq k$ and $\texttt{coverage}(C, R_I) \geq \alpha$. The membership of the decision version of the description mining problem in NP is obvious.

To verify NP-completeness, we reduce the Exact 3-Set Cover problem (EC3) to the decision version of our problem. EC3 is the problem of finding an exact cover for a finite set U, where each of the subsets available for use contain exactly 3 elements. The EC3 problem is proved to be NP-Complete by a reduction from the Three Dimensional Matching problem in computational complexity theory [9].

Let an instance of EC3 (U, S) consist of a finite set $U = \{x_1, x_2, \ldots x_n\}$ and a family $S = \{S_1, S_2, \ldots S_m\}$ of subsets of U, such that $|S_i| = 3$, $\forall i\ 1 \leq i \leq n$. We are required to construct an instance of DEM (R_I, k, α, β) having $k = (\frac{n}{3}+1)$, $\alpha = 100$ (so that, $\texttt{coverage}(C, R_I) = 100\%$) and $\beta = 0$ (so that, $\texttt{error}(C, R_I) = 0$); such that there exists a cover $C \subseteq S$ of $\frac{n}{3}$ pairwise disjoint sets, covering all elements in U, *if and only if*, a solution to our instance of DEM exists.

We define $(m+1)$ Boolean attributes $A = \{A_1, A_2, \ldots A_{m+1}\}$ and $(n+1)$ tuples $T = \{t_1, t_2, \ldots t_{n+1}\}$, where each entry t_i has a corresponding Boolean rating. For each $S_i = \{x_i, x_j, x_k\}$, A_i has Boolean 1 for tuples $\{t_i, t_j, t_k\}$; while the remaining tuples are set to Boolean 0. For attribute A_{m+1}, tuples $\{t_1, t_2, \ldots t_{n+1}\}$ are all set to 0. The ratings corresponding to tuples $\{t_1, t_2, \ldots t_n\}$ are all 0, while tuple $\{t_{n+1}\}$ has a rating of 1. Figure 2.2 illustrates example instances of the EC3 problem and our DEM problem.

EC3:



U = {x1, x2, x3, x4, x5, x6, x7, x8, x9}
S = {S1, S2, S3, S4, S5}
- S1 = {x1, x2, x7}
- S2 = {x3, x4, x8}
- S3 = {x5, x6, x9}
- S4 = {x7, x8, x9}
- S5 = {x4, x5, x6}

DEM:

R_I	A_1	A_2	A_3	A_4	A_5	A_6	Rating
t_1	1	0	0	0	0	0	0
t_2	1	0	0	0	0	0	0
t_3	0	1	0	0	0	0	0
t_4	0	1	0	0	1	0	0
t_5	0	0	1	0	1	0	0
t_6	0	0	1	0	1	0	0
t_7	1	0	0	1	0	0	0
t_8	0	1	0	1	0	0	0
t_9	0	0	1	1	0	0	0
t_{10}	0	0	0	0	0	0	1

Figure 2.2. Example instances of EC3 and DEM.

As defined in Section 2.2, cuboids (or, groups) are selection query conditions retrieving structurally meaningful groupings of the ratings. For Boolean attributes

$\{A_1, A_2, \ldots A_{m+1}\}$, a query condition $Q \in \{0, 1, *\}^{m+1}$, where attribute A_i in Q is set to 0, 1 or *, $\forall i\ 1 \leq i \leq (m+1)$. The space of all possible cuboids (or, query conditions) is 3^{m+1}.

Now, the DEM instance has a solution if $\texttt{error}(C, R_I) = 0$ and $\texttt{coverage}(C, R_I) = 100\%$. Note that, each cuboid in the set C of k cuboids in the solution for DEM should choose tuples either from $T_1 = \{t_1, t_2, \ldots t_n\}$ or from $T_2 = \{t_{n+1}\}$ to achieve $\texttt{error}(C, R_I) = 0$. We need one cuboid 0^{m+1} to cover the single tuple in T_2. Now, let us focus on how to cover tuples in T_1 with $\frac{n}{3}$ more cuboids.

Lemma 1: A selection query $Q \in \{0, *\}^m\{0, 1, *\}$ cannot retrieve non-empty set of tuples only from T_1.

For a query $Q \in \{0, *\}^m\{0, 1, *\}$: if $Q \in \{0, *\}^m\{1\}$, no tuple is selected; if $Q \in \{0, *\}^m\{0, *\}$, non-empty set of tuples from both T_1 and T_2 are selected. Thus queries of the form $\{0, *\}^m\{0, 1, *\}$ cannot yield a solution for the DEM instance.

Lemma 2: A query $Q \notin \{0, *\}^{i-1}\{1\}\{0, *\}^{m-i}0, 1, *$, $\forall i\ 1 \leq i \leq m$ cannot have a solution for the DEM instance.

If a cuboid (or selection query) has 2 or more attributes A_i set to 1, the set of covered tuples is strictly smaller than 3. Thus cuboids that select exactly 3 elements have to have exactly one attribute A_i is set to 1, $\forall i\ 1 \leq i \leq m$,

From Lemmas 1 and 2 we conclude that a set C of ($\frac{n}{3}$ pairwise disjoint cuboids in which one cuboid covers exactly one tuple (defined by query $\{0\}^{m+1}$), and the remaining $\frac{n}{3}$ cuboids each cover exactly 3 tuples (each defined by a query of the form $\{0, *\}^{i-1}\{1\}\{0, *\}^{m-i}\{0, 1, *\}$, and satisfying $\texttt{error}(C, R_I) = 0$ and $\texttt{coverage}(C, R_I) = 100\%$ corresponds to the solution to EC3. The meaningful description mining problem is NP-Complete for Boolean databases. □

Theorem 2.3.2. *The decision version of the problem of meaningful difference mining (DIM) is NP-Complete even for boolean databases.*

Proof: The decision version of the problem of meaningful difference mining (DIM) is as follows: For a given set of items and their ratings R_I, is there a set of cuboids C, such that $\texttt{balance}(C, R_I^+, R_I^-) \leq \beta$, subject to $|C| \leq k$ and $\texttt{coverage}(C, R_I^+) \geq \alpha \wedge \texttt{coverage}(C, R_I^-) \geq \alpha$. The membership of the decision version of the difference mining problem in NP is obvious. To verify its NP-completeness, we again reduce the Exact 3-Set Cover problem (EC3) to the decision version of DIM.

Similar to the proof in Theorem 2.3.1, we consider an instance of EC3 (U, S); we are required to construct an instance of DIM (R_I, k, α, β) having $k = (\frac{n}{3}+1)$, $\alpha = 100$ (so that, $\texttt{coverage}(C, R_I^+) = 100 \wedge \texttt{coverage}(C, R_I^-) = 100\%$) and $\beta = 0$ (so that, $\texttt{balance}(C, R_I^+, R_I^-) = 0$); such that there exists a cover $C \subseteq S$ of size $\frac{n}{3}$, covering all elements in U, *if and only if*, a solution to our instance of DIM exists. The reduction follows the same steps as that in Theorem 2.3.1, except that the ratings corresponding to tuples $\{t_1, t_2, \ldots t_n\}$ are all 0 (indicating negative rating), while tuple $\{t_{n+1}\}$ has a rating of 1 (indicating positive rating). □

Now, we propose efficient algorithms for both description mining and difference mining tasks. The brute-force algorithms for description mining and difference mining enumerate all possible combinations of cuboids, which can be prohibitively expensive. We propose alternative heuristic algorithms to solve the problems of meaningful description mining and meaningful difference mining efficiently.

2.4 Algorithms

2.4.1 Description Mining Algorithms

Given a set I of items and the set R_I of all ratings over I, the description mining task aims to identify a set C of cuboids over R_I, such that the overall aggregate error, $\texttt{error}(C, R_I)$, is minimized, and the size and coverage constraints are satisfied. The straightforward approach is to enumerate all possible combinations of cuboids over R_I.

We introduce this *Exact Algorithm* first, and later propose a more efficient heuristic algorithm based on *randomized hill exploration*.

2.4.1.1 Exact Algorithm (E-DEM)

This algorithm uses brute-force to enumerate all possible combinations of cuboids to return the *exact* (i.e., optimal) set of cuboids as the rating descriptions. Algorithm 10 illustrates its high level pseudo code. The algorithm consists of two stages. During the first stage, it maps the rating lattice to the ratings of the given item set I. In particular, lattice nodes that do not cover any rating in R_I are not materialized and the average ratings of the remaining lattice nodes are computed. In the second stage, the algorithm looks up all $\binom{n}{k}$ possible sets of cuboids C, where n is the number of lattice nodes remaining after the first stage. The set that minimizes $\texttt{error}(C, R_I)$ such that $|C| \leq k$ and $\texttt{coverage}(C, R_I) \geq \alpha$. Clearly, Algorithm E-DEM is exponential in the number of cuboids and can be prohibitively expensive.

Algorithm 1 – E-DEM Algorithm $(R_I,\, k,\, \alpha)\, :\, C$

- Build the rating lattice of n' cuboids (out of n), each of which covers at least one tuple from R_I.

1: **while** true **do**
2: build set $C \leftarrow$ combinatorics-getnext(n', k)
3: **if** $\texttt{coverage}(C, R_I) \geq \alpha$ **then**
4: build list $L \leftarrow$ (C, $\texttt{error}(C, R_I)$)
5: **end if**
6: **end while**
7: $C \leftarrow$ min $\texttt{error}(C, R_I)$ in L
8: **return** C

2.4.1.2 Randomized Hill Exploration Algorithm (RHE-DEM)

A common heuristic technique for solving optimization problems similar to our description mining problem is *random restart hill climbing* [10]. A straightforward

adoption of this technique involves the following. We first randomly select a set of k cuboids as the starting point. The process then continues by replacing one cuboid in the current set with one of its lattice neighbors[4] not in the set as long as the substitution reduces the aggregate error. The algorithm stops when no improvements can be made indicating a *local minima* has been reached. The process is repeated with multiple diverse sets of cuboids to increase the probability of finding the *global minima* that satisfies the constraints.

However, this simple application of hill climbing fails for our description mining task because of the critically important coverage constraint, $\texttt{coverage}(C, R_I) \geq \alpha$. For any given set of cuboids randomly chosen as the starting point, the probability of it satisfying the coverage constraint is fairly small since most cuboids in the lattice cover a small number of ratings. Since the simple hill climbing algorithm cannot optimize for both coverage and aggregate error at the same time, the results produced by the simple hill climbing algorithm often fail the coverage constraint. Hence, a large number of restarts is required before a solution can be found, negating the performance benefits.

To address this challenge, we propose the *Randomized Hill Exploration Algorithm* (RHE-DEM) which first initializes a randomly selected set of k cuboids as the starting point. However, instead of immediately starting to improve the aggregate error, it *explores* the hill to identify nearby cuboid sets that satisfy the coverage constraint. Specifically, RHE-DEM performs iterative improvements on the coverage that lead to a different set of cuboids where the coverage constraint is satisfied. This new cuboid set is then adopted as the starting point for the error optimization with the added condition that an improvement is valid only when the coverage constraint is satisfied. Furthermore, this exploration can advance in multiple directions, producing

[4]Two cuboids are neighbors if they are directly connected in the lattice.

multiple cuboid sets as new start points based on the single initial cuboid set. Since we found single direction exploration works well in practice, we have not pursued this technique.

The details of the algorithm are shown in Algorithm 2. Intuitively, we begin with the rating lattice constructed on R_I. The algorithm starts by picking k random cuboids to form the initial set C. For each cuboid c_i in C, we swap c_i with each of its neighbors c_j in the lattice, while the other cuboids in C remain fixed, to generate a new combination (i.e., cuboid set). The exploration phase computes $\texttt{coverage}(C, R_I)$ for each obtainable combination of k cuboids, until it finds one that satisfies $\texttt{coverage}(C, R_I) \geq \alpha$. The resulting set then acts as the initial condition for the second phase of the optimization to minimize the aggregate error $\texttt{error}(C, R_I)$. The configuration that satisfies $\texttt{coverage}(C, R_I) \geq \alpha$ and incurs minimum error $\texttt{error}(C, R_I)$ is the *best* rating explanation for item set I.

Example 2.4.1. Consider the example rating lattice introduced in Example 2.2.1 and suppose k=2, α=80%. The complete rating lattice will have many more cuboids than what is shown in Figure 2.1, since there are several other attribute-value pairs such as ⟨gender, female⟩, ⟨age, old⟩, ⟨location, NY⟩, etc. Precisely, the total number of cuboids in the rating lattice for "Toy Story" is $n = 17490$, of which $n' = 1846$ have $c_{\mathcal{R}_I} \neq 0$. However, we focus on the example rating lattice having 16 groups to illustrate our description mining algorithms. The exact algorithm will investigate all $\binom{16}{2}$ (or, $\binom{1846}{2}$ for complete rating lattice) possible combinations to retrieve the *best* rating descriptions. On the other hand, the randomized hill exploration algorithm begins by randomly selecting a set of k=2 cuboids, say $c_1 = \{\langle \texttt{G}, \text{male}\rangle, \langle \texttt{0}, \text{student}\rangle\}$ and $c_2 = \{\langle \texttt{L}, \text{ca}\rangle, \langle \texttt{0}, \text{student}\rangle\}$ (marked in double circle in Figure 2.1). Here C_{R_I} = 79, which does not satisfy the constraint $\texttt{coverage}(C, R_I) \geq 80\%$. Keeping c_2 fixed, the obtainable combinations by swapping c_1 with its parent/child are: $\{c_1', c_2\}$,

Algorithm 2 – RHE-DEM Algorithm $(R_I, k, \alpha) : C$

- Build the rating lattice of n' cuboids (out of n), each of which covers at least one tuple from R_I.

1: $C \leftarrow$ randomly select k of n' cuboids
2: **if** `coverage`$(C, R_I) \geq \alpha$ **then**
3: $C \leftarrow$ satisfy-coverage(C, R_I))
4: **end if**
5: $C \leftarrow$ minimize-error(C, R_I))
6: $C' \leftarrow$ best C so far
7: **return** C'

// method satisfy-coverage (C, R_I)*: C*

1: **while** true **do**
2: val $\leftarrow$ `coverage`(C, R_I)
3: **for** each cuboid c_i in C, each neighbor c_j of c_i **do**
4: $C' \leftarrow C - c_i + c_j$
5: $val' \leftarrow$ `coverage`(C', R_I)
6: **if** $val' \geq \alpha$ **then**
7: **return** C'
8: **end if**
9: **end for**
10: **end while**

// method minimize-error (C, R_I)*: C*

1: **while** true **do**
2: val $\leftarrow$ `error`(C, R_I)
3: $\mathcal{C} = \emptyset$
4: **for** each cuboid c_i in C, each neighbor c_j of c_i **do**
5: $C' \leftarrow C - c_i + c_j$
6: **if** `coverage`$(C', R_I) \geq \alpha$ **then**
7: add $(C',$ `error`$(C', R_I))$ to $\mathcal{C}$
8: **end if**
9: **end for**
10: let $(C'_m, val'_m) \in \mathcal{C}$ be the pair with minimum `error`
11: **if** $val'_m \geq val$ **then**
12: **return** C // we have found the local minima
13: **end if**
14: $C \leftarrow C'_m$
15: **end while**

$\{c_1'', c_2\}$, $\{c_1''', c_2\}$ and $\{c_1'''', c_2\}$, where $c_1' = \{\langle \texttt{G}, \textsf{male}\rangle\}$, $c_1'' = \{\langle \texttt{O}, \textsf{student}\rangle\}$, $c_1''' = \{\langle \texttt{G}, \textsf{male}\rangle, \langle \texttt{O}, \textsf{student}\rangle, \langle \texttt{A}, \textsf{young}\rangle\}$ and $c_1'''' = \{\langle \texttt{G}, \textsf{male}\rangle, \langle \texttt{O}, \textsf{student}\rangle, \langle \texttt{L}, \textsf{ca}\rangle\}$. We see $c_1' = \{\langle \texttt{G}, \textsf{male}\rangle\}$, $c_2 = \{\langle \texttt{L}, \textsf{ca}\rangle, \langle \texttt{O}, \textsf{student}\rangle\}$ satisfy the coverage constraint. The set $\{c_1', c_2\}$ is then used as the initial condition to explore the connected lattice and minimize the description error. RHE-DEM on this partial rating lattice eventually returns the cuboids $\{\langle \texttt{G}, \textsf{male}\rangle\}$ and $\{\langle \texttt{O}, \textsf{student}\rangle\}$ as the rating interpretations who share similar ratings on "Toy Story". □

2.4.2 Difference Mining Algorithms

Similar to the description mining task, the task of difference mining (Section 2.2.3) poses an optimization problem with the goal of, given an item set I, identifying a set C of cuboids with the most divergent opinions regarding the ratings R_I over I (i.e., minimizing the aggregate balance, $\texttt{balance}(C, R_I^+, R_I^-)$) and satisfying the size and coverage constraints. The difference mining task is even more challenging because computing the optimization objective, `balance`, is very expensive. We describe this challenge and propose a similar heuristic hill exploration algorithm that leverages the concept of *fundamental region*.

2.4.2.1 Exact Algorithm (E-DIM)

Similar to Algorithm E-DEM, this algorithm uses brute-force to enumerate all possible combinations of cuboids.

2.4.2.2 Randomized Hill Exploration Algorithm (RHE-DIM)

The difference mining problem shares many similar characteristics with the description mining problem. In particular, the measure, aggregate balance, needs to be minimized, while at the same time, a non-trivial constraint, coverage above

a threshold, must be satisfied. This makes the direct application of prior heuristic techniques such as hill climbing difficult. As a result, we leverage the same randomized hill exploration technique as introduced in Section 2.4.1 and propose RHE-DIM.

Similar to RHE-DEM, RHE-DIM first initializes a randomly selected set of k cuboids. It explores the search space, in the first phase, to find a new set of cuboids such that the coverage constraint is satisfied. During the second phase, the algorithm iteratively improves the aggregate balance while ensuring that the coverage constraint remains satisfied, until a *local minima* is identified.

Unlike the description mining problem, however, *computing the optimization measure* $\texttt{balance}(C, R_I^+, R_I^-)$ *for the difference mining problem can be very expensive.* When done naively, it involves a quadratic computation that scans all possible pairings of positive and negative ratings, for each set of k cuboids we encounter during the second phase. To address this computational challenge, we introduce the concept of *fundamental region* (FR), which defines core rating sets induced by a set of k cuboids, to aid the computation of $\texttt{balance}(C, R_I^+, R_I^-)$. The idea is inspired by the notion of finest partitioning [11], with the key difference here being the need for keeping track of both positive and negative ratings.

Definition 2.4.1. *Fundamental Region: Given R_I and the set C of k cuboids in the rating lattice, we can construct a k-bit vector signature for each tuple in R_I, where a bit is set to true if the tuple is covered by the corresponding cuboid. A* fundamental region (denoted by F) *is thus defined as the set of ratings that share the same signature. The number of fundamental regions is bounded by $2^k - 1$, but is often substantially smaller.*

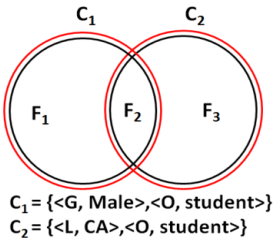
Given the set of fundamental regions, we can compute $\texttt{balance}(C, R_I^+, R_I^-)$ by iterating over all pairs of fundamental regions instead of all pairs of tuples, thus having a significant performance advantage. Specifically, for a self-pair involving a single region F_i, we have $\texttt{balance}(C, R_{I\,i}^+, R_{I\,i}^-) = F_i(R_I^+) \times F_i(R_I^-)$; for a pair of distinct

regions F_i and F_j sharing at least a common cuboid, we have $\texttt{balance}(C, R^+_{I\,ij}, R^-_{I\,ij})$ $= F_i(R^+_I) \times F_j(R^-_I) + F_j(R^+_I) \times F_i(R^-_I)$. Finally, we have:

$$\begin{aligned} \texttt{balance}(C, R^+_I, R^-_I) \;\; = \;\; & m \times (\sum\nolimits_i \texttt{balance}(C, R^+_{I\,i}, R^-_{I\,i}) + \\ & \sum\nolimits_{ij} \texttt{balance}(C, R^+_{I\,ij}, R^-_{I\,ij})) \end{aligned} \tag{2.1}$$

where m is the normalization factor described in Section 2.2.3.

Example 2.4.2. Consider a set $C = \{c_1, c_2\}$ of $k = 2$ cuboids, where $c_1 = \{\langle \texttt{G}, \textsf{male}\rangle$, $\langle \texttt{O}, \textsf{student}\rangle\}$ and $c_2 = \{\langle \texttt{L}, \textsf{ca}\rangle$, $\langle \texttt{O}, \textsf{student}\rangle\}$ (marked in double circle in Figure 2.1). The two cuboids partition the set of 79 ratings (covered by C) into 3 fundamental regions F_1, F_2 and F_3 each having a distinct signature, as shown in Figure 2.3. The positive and negative rating tuple counts, $F(R^+_I)$ and $F(R^-_I)$ respectively in each region are also presented in Figure 2.3. By Equation 2.1, $\texttt{balance}(C, R^+_I, R^-_I)$, can be computed as: $\frac{1}{46\times33}\times$ $(40\times29 + 4\times2 + 2\times2 + (40\times2 + 4\times29) + (4\times2 + 2\times2))$, based on counts in F_1, F_2, F_3, (F_1, F_2) and (F_2, F_3) respectively. □



F	C_1C_2	Count $F(R^+_I),F(R^-_I)$
F_1	1 0	40, 29
F_2	1 1	4, 2
F_3	0 1	2, 2

Figure 2.3. Computing $\texttt{balance}(C, R^+_I, R^-_I)$ using Fundamental Regions..

Theorem 2.4.1. *Given R_I and C, $\texttt{balance}(C, R^+_{I\,i}, R^-_{I\,i})$ computed using Equation 2.1 is equivalent to the one computed using the formula in Section 2.2.3.*

Proof : The standard computation of aggregate balance $\texttt{balance}(C, R^+_I, R^-_I)$ looks up all possible pairings of positive and negative ratings, for each set of k cuboids.

The pseudo code of the standard technique is presented in Algorithm 3. It scans each of the k cuboids in C to identify possible positive and negative rating pairings. The method maintains a $|R_I^+| \times |R_I^-|$ matrix for book-keeping, all of whose elements are first initialized to zero and then set to one, whenever a particular element position (corresponding to postive-negative rating pairing) is encountered. The total number of one-s in the $|R_I^+| \times |R_I^-|$ matrix determines the measure $\texttt{balance}(C, R_I^+, R_I^-)$. $\square$

This fundamental region based balance computation involves populating a $min(n_{fr}, |R_I^+|) \times min(n_{fr}, |R_I^-|))$ matrix, where n_{fr} is the number of fundamental regions induced by the set C of k cuboids ($n_{fr} \leq 2^k - 1$), each cell stores the balance between a pair of FRs (or a self pair), and summing over all cells to compute the overall balance. The details are presented in Algorithm 3. Finally, Algorithm RHE-DIM works the same way as RHE-DEM presented in Algorithm 2, with all $\texttt{error}(C, R_I)$ computation being replaced with compute-balance(C, R_I^+, R_I^-) of Algorithm 3.

Algorithm 3 – compute-balance(C, R_I^+, R_I^-): v

1: **for** i'=1 to n_{fr} **do**
2: $\quad F_{i'}(R_I^+, R_I^-) \leftarrow \{count(F_{i'}, R_I^+), count(F_{i'}, R_I^-)\}$
3: **end for**
4: **for** i=1 to n_{fr}, j=1 to n_{fr} **do**
5: $\quad$ pairing-matrix-fr(i, j) $\leftarrow 0$
6: **end for**
7: **for** i=1 to $2^k - 1$, j=1 to $2^k - 1$ **do**
8: $\quad$ **if** i = j **and** pairing-matrix-fr(i, j) = 0 **then**
9: $\quad\quad$ pairing-matrix-fr(i, j) $\leftarrow F_i(R_I^+) \times F_i(R_I^-)$
10: $\quad$ **else if** i $\neq$ j **and**
$\quad\quad$ pairing-matrix-fr(i, j) = 0 **and**
$\quad\quad$ F_i, F_j belongs to same cuboid in C **then**
11: $\quad\quad$ pairing-matrix-fr(i, j) $\leftarrow F_i(R_I^+) \times F_j(R_I^-)$
12: $\quad\quad$ pairing-matrix-fr(j, i) $\leftarrow F_j(R_I^+) \times F_i(R_I^-)$
13: $\quad$ **end if**
14: **end for**
15: $v \leftarrow$ sum of all non-zero products in pairing-matrix-fr
16: **return** v

Complexity Analysis: The computational complexity of the description mining and difference mining problems can be viewed as depending on the parameters: R_I (R_I^+, R_I^-), the set of ratings over item set I; n', the number of cuboids in rating lattice covering at least one rating from R_I; and k, the number of cuboids to be presented to user. The exact algorithms E-DEM and E-DIM are exponential in n'. The heuristic algorithms RHE-DEM and RHE-DIM work well in practice (as shown in Section 2.5); but of course they do not guarantee any sort of worst case behavior, either in running time or in result quality. We note that the performance of RHE-DIM for difference mining is dependent on the computation of the optimization measure, $\texttt{balance}(C, R_I^+, R_I^-)$.

The naive way of computing the aggregate balance involves a quadratic computation that scans all possible pairings of positive and negative ratings, for each set C of k cuboids, during the second phase. The runtime complexity for balance computation this way is $\mathrm{O}(k \times |R_I^+| \times |R_I^-|)$. The alternate way of using the concept of fundamental regions reduces the complexity since it concerns pairings of positive and negative rating regions, instead of pairings of positive and negative rating tuples. The number of fundamental regions for a set of k cuboids is 2^k-1. Therefore, the reduced running time is given by $\mathrm{O}(k \times min(2^k - 1, |R_I^+|) \times min(2^k - 1, |R_I^-|))$.

2.5 Experiments

We conduct a set of comprehensive experiments to demonstrate the quality and efficiency of our proposed MRI algorithms. First, we show that our randomized hill exploration algorithms are scalable and achieve much better response time than the exact algorithms while maintaining similar result quality (Section 2.5.1). Second, through a set of Amazon Mechanical Turk studies, we demonstrate that interpretations generated by our approaches are superior to the simple aggregate ratings returned by current systems (Section 2.5.2).

Real Movie Dataset: We use the MovieLens[5] 100K ratings dataset for our evaluation purposes because the two alternative MovieLens datasets with more ratings (1M and 10M ratings datasets) do not contain user details that are required for our study. The dataset has 100,000 ratings for 1682 movies by 943 users.

User Attributes: There are four user attributes that we consider in the MovieLens dataset that we adopt, including gender, age, occupation and zipcode. The attribute *gender* takes two distinct values: male or female. We convert the numeric *age* into four categorical attribute values, namely teen-aged (under 18), young (18 to 35), middle-aged (35 to 55) and old (over 55). There are 21 different *occupations* listed by MovieLens, such as student, artist, doctor, lawyer, etc. Finally, we convert zipcodes to states in the USA (or foreign, if not in USA) by using the USPS zip code lookup. This produces the user attribute, *location*, which takes 52 distinct values.

Table 2.1. Bin Statistics.

	lowest #rtg	highest #rtg	avg #rtgs
Bin 1	1	4	2
Bin 2	4	11	7
Bin 3	11	27	18
Bin 4	27	59	41
Bin 5	59	121	84
Bin 6	121	583	212

Binning the Movies: The number of ratings for each movie can vary significantly, which can have a significant impact on the performance of the algorithms. Intuitively, the more ratings we have to consider, the more costly the interpretation process is expected to be. Therefore, we order our set of 1682 movies according to the number of ratings each movie has, and then partition them into 6 bins of equal sizes, where Bin 1 contains movies with the fewest number of ratings (on average 2) and Bin 6 contains movies with highest number of ratings (on average 212). Table 2.1 shows the

[5]http://www.grouplens.org/datasets/movielens/

statistics of those bins. We randomly pick 100 movies from each bin and compare the execution time and the objective score (`error` for description mining and `balance` for difference mining) of both the exact algorithms and our heuristic algorithms.
System configuration: Our prototype system is implemented in Java with JDK 5.0. All experiments were conducted on an Windows XP machine with 3.0Ghz Intel Xeon processor and 2GB RAM. The JVM size is set to 512MB. All numbers are obtained as the average over three runs.

2.5.1 Quantitative Evaluation

We compare the execution time for computing interpretations using the exact and the randomized hill exploration algorithms. For all our experiments, we fix the number of groups to be returned at $k = 2$, since the brute-force algorithms are not scalable for larger k.

Figure 2.4 and 2.5 illustrates the average execution time and average description error respectively for E-DEM and RHE-DEM. As expected, while the execution time difference is small for movies with small number of ratings, the RHE algorithm computes the descriptions much faster than the exact algorithm for movies with a large number of ratings (i.e., Bins 5, 6). Moreover, it reduces the execution time from over 7 seconds to about 2 seconds on average for movies in *Bin 6*, which is significant because we can effectively adopt description mining in an interactive real-time setting with our RHE algorithm. Another observation is that for any movie randomly chosen from any bin, our heuristic algorithm identifies a local minima very close to the optimal solution without requiring multiple restarts. In fact, as Figure 2.5 illustrates, the average description error is only slightly larger for RHE, despite significant reduction in the execution time. Figures 2.6 and 2.7 report similar results comparing the average execution time and the average balance score respectively for E-DIM and

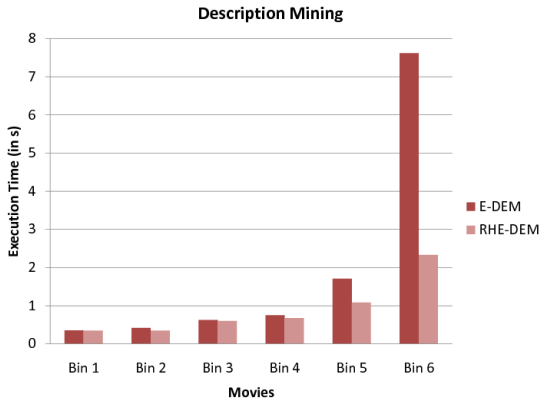


Figure 2.4. Execution time: E-DEM vs RHE-DEM.

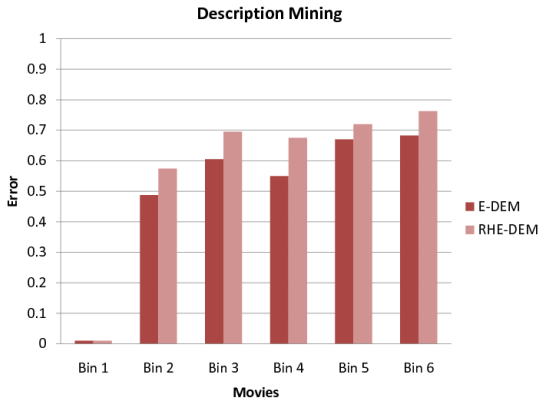


Figure 2.5. `error`(C, R_I): E-DEM vs RHE-DEM.

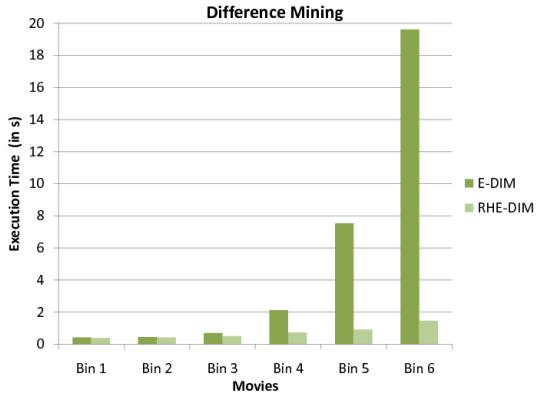


Figure 2.6. Execution time: E-DIM vs RHE-DIM.

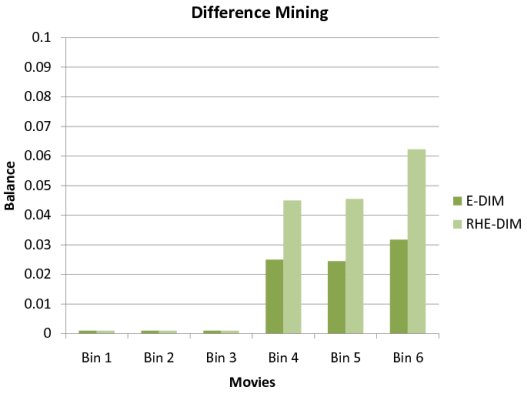


Figure 2.7. `balance`(C, R_I^+, R_I^-): E-DIM vs RHE-DIM.

RHE-DIM. Again, we see that our heuristic algorithm performs much faster (reducing the execution time from over 20 second to less than 2 seconds) without compromising much on quality.

2.5.1.1 Scalability Analysis

Figures 2.8 and 2.9 illustrate the execution time of our RHE algorithms for description mining and difference mining respectively, over increasing number of cuboids in the results. A randomly chosen movie, "Gone With The Wind", is

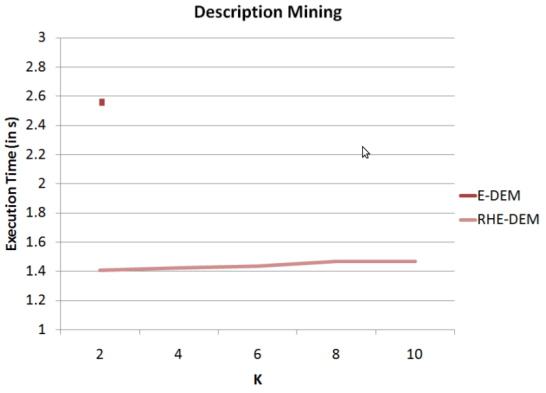


Figure 2.8. Execution time with increasing k: DEM.

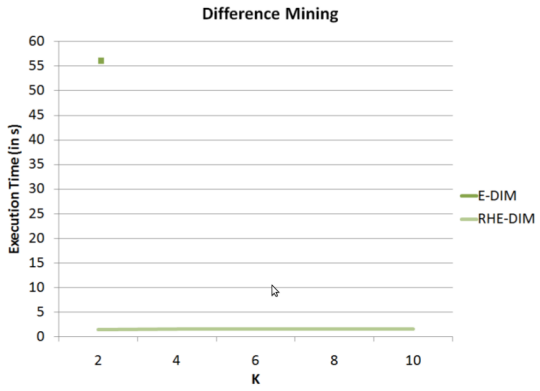


Figure 2.9. Execution time with increasing k: DIM.

used in this analysis. The results show that RHE algorithms are scalable where the execution time remains reasonably small through the range of k values up to 10, which we believe to be the upper limit of how many explanations a user can consume for a single item. Note that the execution time of brute-force algorithms could not be reported beyond $k = 2$ because they failed to finish within a reasonable amount of time. High coverage of item ratings by few general groups such as $\{\langle \mathsf{age}, \mathsf{young}\rangle, \langle \mathsf{occupation}, \mathsf{student}\rangle\}$, etc. who frequently participate in collaborative rating sites and very low coverage by majority of the groups in the rating lattice such as $\{\langle \mathsf{gender}, \mathsf{female}\rangle, \langle \mathsf{age}, \mathsf{old}\rangle, \langle \mathsf{occupation}, \mathsf{librarian}\rangle\}$, etc. supports the exploration phase to reach a local minima quickly, thus making our RHE algorithms scalable.

2.5.2 Qualitative Evaluation

We now evaluate the benefits of rating interpretations in an extensive user study conducted through Amazon Mechanical Turk (AMT). In particular, we aim to analyze whether users prefer the more sophisticated rating interpretations that we are proposing against the simple rating aggregation (i.e., average ratings) that all online rating sites currently adopt. We conduct two sets of user studies, one for description

mining and one for difference mining. Each set involves 4 randomly chosen popular movies[6] and 30 independent single-user tasks. For description mining, the four movies chosen are "Toy Story", "Titanic", "Mission Impossible", and "Forrest Gump". For difference mining, we bias toward more controversial movies and chose "Crash", "101 Dalmatians", "Space Jam", and "Ben Hur".

Each task is conducted in two phases: *User Knowledge Phase* and *User Judgment Phase*. During the first phase, we estimate the users' seriousness about the task and familiarity about the movies in the task by asking them to complete a survey. The survey contains a few very simple questions about the movies that we use to prune out malicious users who simply try to complete the task by answering questions randomly. We also draw some interesting observations from the user study based on the user's level of familiarities with the movie. In the second phase, for each movie in the task, we present to the user three alternative interpretations of the ratings about the movie for description mining and difference mining:

- Option (a) overall average rating (simple)
- Option (b) the interpretation produced by the exact algorithms (E-DEM, E-DIM)
- Option (c) the interpretation produced by our randomized hill exploration algorithms (RHE-DEM, RHE-DIM), where the number of explanations (i.e., the number of cuboid groups presented) for both exact and heuristic is limited at 3.

The user is then asked to judge which approach she prefers. The responses from all the users are then aggregated to provide an overall comparison between the three approaches.

[6]For movies that are not popular, uses can simply go over all the ratings one by one, therefore rating interpretation does not bring much benefit.

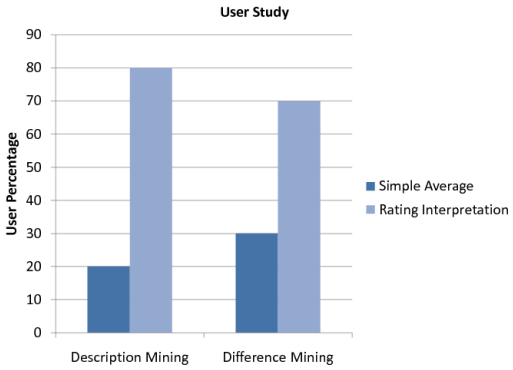


Figure 2.10. Users Prefer Rating Interpretations.

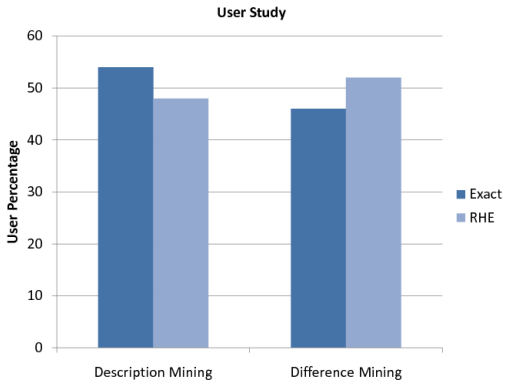


Figure 2.11. Exact and RHE Algorithms Produce Similar Results.

Figure 4.10 compares the simple overall average rating approach against our approach of returning movie rating interpretations to the user. The simple average represents the percentage of users choosing average ratings, whereas the latter is computed as an addition of percentage of users preferring rating interpretations produced by either exact or RHE algorithms. From the results, it is clear that users overwhelmingly prefer the more informative explanations to the overall average rating, thus confirming our motivation. We also observe that when an user is unfamiliar with a movie in the study, she is particularly inclined to meaningful rating explanations over average rating. To verify that the quality of results produced by our RHE algorithms are on par with the exact algorithms, we leverage the same user study facility to compare the interpretations produced by both. As shown in Figure 2.11, from the user's perspective and for both description mining and difference mining, results produced by exact and RHE algorithms are statistically similar. This validates that our heuristic algorithms are viable, cheaper, alternatives to the brute-force algorithms.

2.6 Demonstration

We build a system MapRat that allows a user to explore multiple carefully chosen aggregate analytic details over a set of user demographics that meaningfully

explain the ratings associated with item(s) of interest. MapRat allows a user to systematically explore, visualize and understand user rating patterns of input item(s) so as to make an informed decision quickly. We focus on two main tasks: Similarity Mining (SM) which identifies groups of reviewer sharing similar ratings on item(s) and Diversity Mining (DM) which identifies groups of reviewer sharing dissimilar ratings on item(s).

There exist prior systems such as OIC Weave[7] that can provide visualization of ratings along different demographic attributes. However, such systems do not provide any automatic and interactive exploration of the rating information. Though Weave allows exploration along a single dimension, it fails to identify more granular reviewer sub-populations that can potentially have interesting rating patterns.

2.6.1 Architecture

There are two major components in the MapRat system : Rating Mining and Visualization.

Rating Mining: This module accepts a set of items I from the front-end and collects all the corresponding rating tuples R_I. The set of groups that has at least one rating tuple in R_I are then constructed. The next step is to cast the problem as an optimization task corresponding to each of the two sub-problems : Similarity Mining and Diversity Mining. For each of the two sub-problems, the RHE algorithm is employed to retrieve the best set of reviewer groups that provide meaningful rating interpretations. Besides returning explanations, our system also provides visualization of the review groups. The location of the reviewer is a convenient and natural attribute to anchor the visualization. Such location based visualization allows for rapid scanning of the explanations, highlight geographical trends in rating patterns (if any) and also provides a mechanism to overlay explanations from different interpretations. Using

[7]http://oicweave.org/

a combination of aggressive data pre-processing, result pre-computation and caching techniques, the latency of MapRat is minimized.

Visualization: This module is responsible for displaying the rating interpretations over a map such that a user can get a fast overview of the rating trends over geographic regions. Each of the group always specify a geo-condition and hence it is always visualizable on the map. The set of groups that are generated from each of the sub-problems (SM and DM) is considered as rating interpretation *object*. Each set of such objects are then rendered as a Choropleth map [12] using the average group rating for shading. Dark red corresponds to lowest rating while dark green denotes the highest and the intermediate values are represented by the red-green gradient. Each group is also annotated with icons that identify the attribute value pairs used to define it. The set of these Choropleth maps form an *exploration*. Such an exploration is formed from the same set of input rating tuples R_I and constraints, but provide different perspective in terms of meaningful rating interpretations. Collectively, the two different visualizations provide a comprehensive insight into reviewer rating patterns. In addition, the system also allows a user to drill deeper and view lower level aggregate statistics. For example, if the original geo condition was over a state, the drill down provides city level statistics. Finally, navigation over time dimension allows a user to understand the evolution of the reviewer rating pattern over a period of time.

2.6.2 User Interface

The MapRat system can work on any collaborative rating site that provides data as described in Section 2.5. For the purpose of the demo, we use Million rating data set from MovieLens. We integrate the MovieLens data with information available from IMDB, in order to include additional item attributes such as actors and directors.

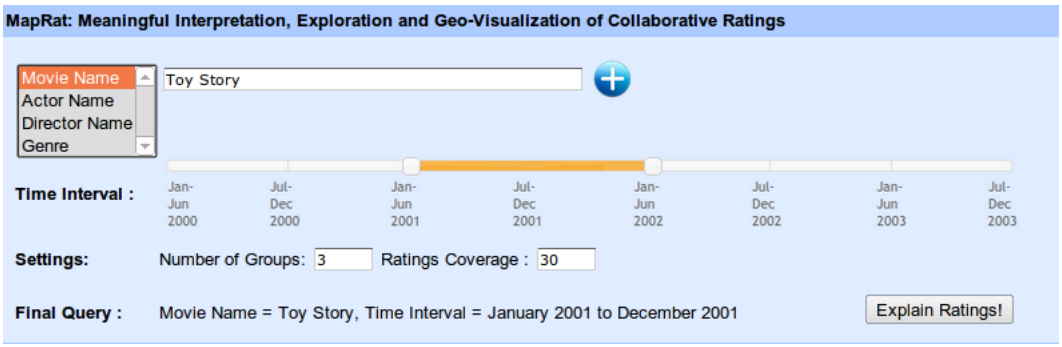


Figure 2.12. Primary User Interface of MapRat.

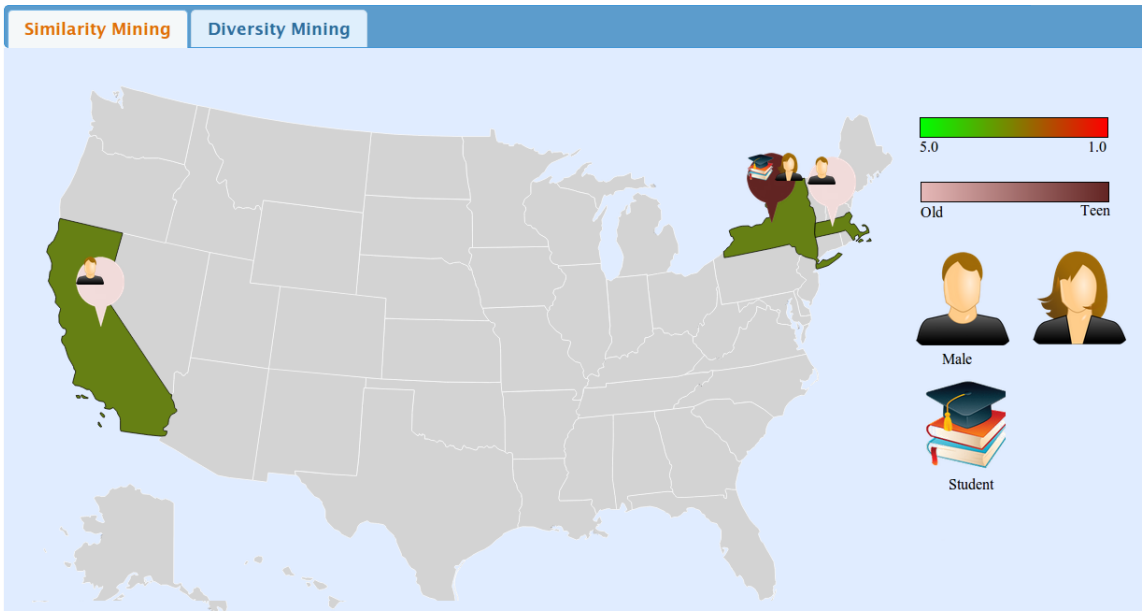


Figure 2.13. MapRat Explanation Result for Query in Figure 3.15.

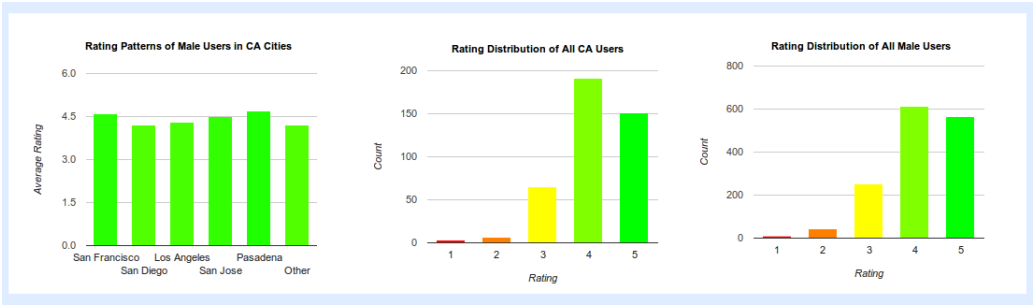


Figure 2.14. MapRat Exploration Result for Explanation Male reviewers from California.

The MapRat system consists of a web based front-end that allows a user to enter one or more items. The primary UI for entering is shown in Figure 3.15. A user can enter a conjunctive or disjunctive query by entering one or more attribute value pairs. Possible attributes include movie title, actor, director and genre. Furthermore, the user can restrict the mining over a specific time interval, so that the evolution of rating behaviors over a period can be observed. The user can enter additional search settings such as the maximum number of groups to be returned and its rating coverage. Suppose in Figure 3.15, a user wants to interpret how the reviewer ratings have evolved over the years for the movie "Toy Story". For this, the user enters the search query "Toy Story" and sets the type of the query to Movie Name. Once the additional search settings have been entered, the user clicks on Explain Ratings and fetches the results. Moving the time slider over the range of values allows the user to observe reviewer groups that provide best interpretations for the movie and how they change over time.

The result of such a query is shown in Figure 2.13. Groups from different sub-problems (Similarity Mining and Diversity Mining) are visualized in two different tabs. For this demo, each of the groups always specify the state as their geo condition in order to allow rendering of the explanation in the map. The average rating of the group is used for highlighting the state. We use a red (rating 1.0) to green (rating 5.0) Likert Scale for depicting the average rating. The other reviewer attributes associated with the group are highlighted through icons as a visual aid to the user. The color of the pin holding the icons depicts the age group of the sub-population.

For example, Figure 2.13 shows the best three groups for Similarity Mining: male reviewers from California, male reviewers from Massachusetts and female teen student reviewers from New York. In this particular instance, the displayed groups neatly correspond to the major market segments of animation movie box office - male and

young movie goers. However, it must be noted that MapRat strives to highlight representative groups that the user can self-identify with. Explaining why the chosen groups exhibited such rating behavior is significantly more complex and is not the primary aim of MapRat. All three groups have rated the movie positively as indicated by the color used for highlighting the respective states; the average rating by female teen student reviewers from New York is however lower than those by the remaining groups. These groups consists of reviewers whose individual rating is very closer to the group average and also cover a reasonable fraction of rating tuples. Clicking on any of the groups displays additional statistics about the group's rating. Figure 2.14 shows the statistical details that are shown to the user when she clicks on the result Male reviewers from California for further exploration. This provides a convenient way to compare the rating patterns of related groups. It is also possible to drill down and view the city level aggregate movie rating statistics for each of the groups during such interactive exploration. Finally, MapRat can exploit any user demographic information (gender, age, location or occupation) available to constrain the groups that are highlighted. This ensures that the resulting groups are the ones that user most self-identifies with and hence most relevant for decision making.

2.6.3 Demo Plan

Our demo allows the audience to use a web interface (as shown in Figure 3.15) and specify arbitrary search query involving one or more movie attributes. Example queries include "The Social Network", "Tom Hanks", "The Lord of the Rings film trilogy", "thriller movies directed by Steven Spielberg" and so on. The audience can specify other search settings and the time interval to restrict the mining.

Based on the query, our system will display the visualizations for the two meaningful rating interpretation problems. The audience can explore the results to have a

better understanding of the reviewer rating patterns for the query. They can observe how the rating patterns fluctuate over a period of time or drill down deeper to view the rating statistics at city level. Such exploration will give the audience a deeper appreciation of our system's utility to aid users make informed judgments about movies quickly. It will also clearly show the superiority of our system in describing rating explanations in terms of meaningfully labeled user groups, over existing collaborative rating sites.

Next, we extend the current ideas to analyze handle handle tag-based feedback since analyzing how tags are assigned by certain users to certain items has more powerful implications in helping users search for desired information.

CHAPTER 3

Collaborative Tagging Behavior Analysis

3.1 Introduction

Tagging is a core activity on the social web. It reflects a wide range of content interpretations and serves many purposes, ranging from bookmarking websites in del.icio.us, organizing personal videos in YouTube, and characterizing movies in MovieLens. While one can possibly examine tags used by a single user on a single item, it is easy to see that the task becomes quickly intractable for a collection of tagging actions involving multiple users and items. In this chapter, we aim to formalize the analysis of the tagging behavior of a set of users for a set of items and develop appropriate algorithms to complete that task.

A typical tagging action involves three components (i.e., dimensions), *user*, *item*, and *tag*. We propose to study a variety of analysis tasks that involve applying two alternative measures, *similarity* and *diversity*, to those components and producing groups of similar or diverse items, tagged by groups of similar or diverse users with similar or diverse tags. For example, one possible analysis outcome could be: "teenagers use diverse tags for action movies" or "males from New York and California use similar tags for movies directed by Quentin Tarantino". A general dual mining framework that encompasses many common analysis tasks is defined in Section 3.2.4, and an extension to the framework in Section 3.5. A core challenge in this dual mining framework is the design of similarity and diversity measures. For user or item components, defined by (attribute, value) pairs, several existing comparison techniques have been proposed that can leverage their structured nature or bipartite connections. Section 3.2.1 illustrates some of those techniques.

Figure 3.1. Tag Signature for All Users.

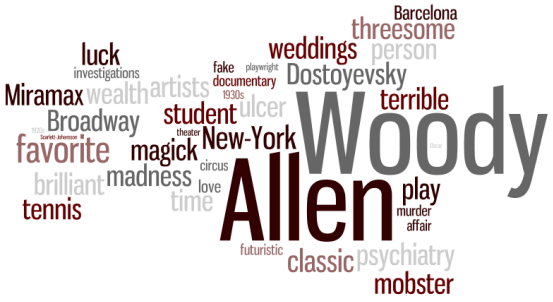


Figure 3.2. Tag Signature for CA Users.

Comparing similarity and diversity of tags used by various users on different items, however, presents a new challenge. First, tags are drawn from a much larger vocabulary than user or item attributes and exhibit a long tail characteristic. Second, it is often the case that different tags are used for the same set of items and, accounting for those tags separately would not capture their co-usage. Finally, tags may have linguistic connections such as synonymy. In order to capture tag similarity and diversity, we propose to *summarize* tags first to account for their co-usage and semantic relationships. Section 3.2.2 describes some techniques that we borrow from information retrieval and machine learning that can be used.

The tag component is also the most interesting among the three to be analyzed. Figure 3.1 shows a rendering of a tag summarization for "Woody Allen" movies in the form of a tag cloud. Similarly, Figure 3.2 shows a summarization of tags for the same movies from California users only. In both cases, summarization is defined as a simple frequency-based tag cloud where the size of a tag corresponds to how often it has been used on those movies. While `Woody` and `Allen` are not surprisingly common to both, the two clouds are different: all users highlight the `dramatic`, `tragic` and `disturbing` nature of those movies, and California users emphasize tags such as `classic` and `psychiatry`. Moreover, one of the director's popular movies, `Noiva Nervosa` is prominent in the tag cloud of all users, and yet is conspicuously absent

in that of California users. Our goal is to define analysis tasks that can help users easily spot those interesting patterns and use that knowledge in subsequent actions.

We emphasize that, in this study, it is *not* our goal to advocate one particular similarity or diversity measure over another. Rather, we focus on formalizing the Tagging Behavior Dual Mining (TagDM) framework and the problem definitions, and designing algorithms that will work well for most measures. The analysis problems formally defined in our proposed framework fall into the wider category of constrained optimization problems. We are looking for groups of tagging actions that achieve maximum similarity or diversity on one or more components while satisfying a set of conditions and constraints. We first discuss a set of mining tasks that our TagDM framework can handle, and then formalize the general TagDM framework.

Given that a typical tagging action involves three components (i.e., users, items, and tags), a large number of concrete problem instances can be defined, with their variations coming from two main aspects. The first category of variations depends on which measure (similarity or diversity) is applied to which tagging components (users, items, or tags). For example, a user can be interested in identifying *similar* user groups who use *similar* tags for *diverse* item groups, or in identifying *diverse* user groups who use *similar* tags for *similar* item groups. Since there are three components, each of which can adopt one of two measures, this variation alone can lead to $2^3 = 8$ different problem instances. Since we are looking for result groups that achieve maximum similarity or diversity on one or more components while satisfying a set of conditions and constraints, the second category of variations depends on which components the user is adding to the optimization goal and which components the user is adding to the constraints. For example, a user can be interested in finding tagging action groups that maximize a *tag diversity* measure while satisfying *user diversity and item similarity* constraints, or groups that maximize a combination of *tag diversity and user*

diversity measures while satisfying an *item similarity* constraint. Figures 3.1 and 3.2 depict an example of the former instance - diverse users (all users and California users) maximizing tag diversity for similar items (Woody Allen movies). Since each component can be part of the optimization goal, or part of the constraint, or neither, this variation can lead to $3^3 - 1 = 26$ different problem instances. Combining both categories of variations, there is a total of 112 concrete problem instances that our framework captures! We formalize such a TagDM framework in Section 3.2. Of course, we can extend the TagDM framework to add additional conditions to the optimization goal(s) of these 112 instances based on user needs, the details of which are described in Section 3.5.

Table 3.1. Concrete TagDM Problem Instantiations. Column C lists the constraint dimensions and column O lists the optimization dimensions.

ID	User	Item	Tag	C	O
1	similarity	similarity	similarity	U,I	T
2	similarity	diversity	similarity	U,I	T
3	diversity	similarity	similarity	U,I	T
4	diversity	similarity	diversity	U,I	T
5	similarity	diversity	diversity	U,I	T
6	similarity	similarity	diversity	U,I	T

Table 3.1 illustrates six of the problem instantiations that we have investigated in details through the rest of the chapter. In particular, we focus on problems with all three components with constraints on user and item and optimization on the tag component, since those are the most novel and intuitive mining problems. Under this setting, the user and item components are used primarily for enforcing constraints (similarity/diversity) and in providing an intuitive description of the tagging action groups. However, the optimization would be done over the tagging component. Two tagging action groups would be considered similar if their tagging components have small distance between them.

Not surprisingly, as our complexity analysis shows in Section 3.3, those problems are NP-Complete in general. We propose four sets of efficient algorithms for solving them, the first three of which consider pair-wise aggregation measures for capturing similarity and diversity while the fourth employs a more general mining measure. The first set incorporates Locality Sensitive Hashing (LSH) and can be used for problems maximizing tagging action component similarity. While traditional LSH is frequently used for performing nearest neighbor search in high-dimensional spaces, our algorithm finds the bucket containing the result set of our tagging behavior analysis. The second set of algorithms borrows ideas from techniques employed in Computational Geometry to handle the Facility Dispersion Problem (FDP) and is effective for problems maximizing diversity. Both sets of algorithms possess compelling theoretical characteristics for problem instances optimizing the dual mining goal without any constraints. For both sets, we also propose advanced techniques that return better quality results in comparable running time. The third set of algorithms use Hierarchical Agglomerative Clustering (HAC) techniques and can be applied to problems maximizing either similarity or diversity. It is particularly useful for handling complex mining problems, having additional conditions in optimization goals. All three sets of algorithms consider some form of pair-wise distance measure for computing similarity or diversity. Therefore, we propose a fourth set of more general algorithm that is based on the Hill Climbing (HC) technique and can handle any arbitrary mining function for similarity and diversity.

3.2 Problem Framework

Similar to Section 2, we model a collaborative tagging site $\mathcal{D}$ as a triple $\langle \mathcal{U}, \mathcal{I}, \mathcal{R} \rangle$, representing the set of reviewers (i.e., users), the set of items (i.e., products) and the feedback (i.e, tag vocabulary), respectively. Each feedback r, i.e., tagging action can

be considered as a triple itself, represented as $\langle u, p, \mathtt{T}\rangle$, where $u \in \mathcal{U}$, $i \in \mathcal{I}$, $\mathtt{T} \subset \mathcal{R}$, respectively. A group of tagging actions is denoted as $g = \{\langle u_1, p_1, \mathtt{T}_1\rangle, \langle u_2, p_2, \mathtt{T}_2\rangle, \ldots, \}$. We define a user schema, $\mathcal{U}_\mathcal{A} = \langle ua_1, ua_2, \ldots\rangle$, to represent each user as a set of attribute values conforming to the user schema: $u = \langle uv_1, uv_2, \ldots\rangle \in \mathcal{U}$, where each uv_x is a value for the attribute $ua_x \in \mathcal{U}_\mathcal{A}$. For example, let $\mathcal{U}_\mathcal{A} = \langle \mathsf{age}, \mathsf{gender}, \mathsf{state}, \mathsf{city}\rangle$, a user can be represented as $\langle 18, \mathsf{student}, \mathsf{new\ york}, \mathsf{nyc}\rangle$. Similarly, we define an item schema, $\mathcal{I}_\mathcal{A} = \langle pa_1, pa_2, \ldots\rangle$, to represent each item as a set of attribute values, $i = \langle pv_1, pv_2, \ldots\rangle$, where each pv_y is a value for the attribute $pa_y \in \mathcal{I}_\mathcal{A}$.

Each tagging action therefore can be represented as an expanded tuple that concatenates the user attributes, the item attributes and tags: $r = \langle uv_1, uv_2, \ldots, pv_1, pv_2, \ldots, \mathtt{T}\rangle$. G denotes the set of all such tagging action tuples. Many social sites have hundreds of millions of such tuples. Most, if not all, mining tasks involve analyzing sets of such tuples collectively. While there are a number of different ways tagging action tuples can be grouped, we adopt the view proposed and experimentally verified in [1], where groups of users (or items) that are *structurally describable* (i.e., sharing common attribute value pairs) are meaningful to end-users. Such groups correspond to conjunctive predicates on user or item attributes. An example of a *user describable tagging action group* is $\{\langle \mathsf{gender}, \mathsf{male}\rangle, \langle \mathsf{state}, \mathsf{new\ york}\rangle\}$, and of an *item describable group* is $\{\langle \mathsf{genre}, \mathsf{comedy}\rangle, \langle \mathsf{director}, \mathsf{woody\ allen}\rangle\}$. Next we define an essential characteristic of a set of tagging action groups.

Definition 3.2.1. *Describable User Group: A group of tagging actions, $g = \{\langle u_1, p_1, \mathtt{T}_1\rangle, \langle u_2, p_2, \mathtt{T}_2\rangle, \ldots, \}$, are considered a* meaningful user group *iff: $\exists A \subset \mathcal{U}_\mathcal{A}, |A| > 0$, such that, for each $a \in A$, $\exists v, \forall r \in g, r_u.ua = uv$. A set of attribute value pairs, $D^g_{user} = \{ua_1{=}uv_1, ua_2{=}uv_2, \ldots, \}$, where $ua_x \in A$ and uv_x is the value of a_x shared by all tuples in g, are called the* user group description. *Furthermore, we say that tuples in g* satisfy *D^g_{user}.*

Definition 3.2.2. *Describable Item Group: A group of tagging actions,* $g = \{\langle u_1, p_1, \mathtt{T}_1\rangle, \langle u_2, p_2, \mathtt{T}_2\rangle, \ldots, \}$*, are considered a* meaningful item group *iff:* $\exists A \subset \mathcal{I}_{\mathcal{A}}, |A| > 0$*, such that, for each* $a \in A$*,* $\exists v, \forall r \in g, r_i.pa = pv$*. We define* item group description *as* $D^g_{item} = \{pa_1{=}pv_1, pa_2{=}pv_2, \ldots, \}$*, where* $pa_x \in A$ *and* pv_x *is the value of* pa_x *shared by all tuples in* g*. We say that tuples in* g satisfy D^g_{item}*.*

Definition 3.2.3. *Group Support: Given the input set of tagging action tuples* G*, the* support *of a set of tagging action groups* $\mathcal{G} = \{g_1, g_2, \ldots\}$ *over* G*, is defined as* $Support^{\mathcal{G}}_G = |\{r \in G \mid \exists g_x \in \mathcal{G}, r \in g_x\}|$*. Intuitively, group support measures the number of input tagging action tuples that belongs to at least one of the groups in* $\mathcal{G}$*.*

Before we formalize the mining problems, we introduce the core concept of Mining Function that computes a similarity or diversity score using arbitrary evaluations over the tagging action groups. Similarity and diversity are usually estimated as functions of distance. In particular, aggregation of pair-wise distances between the different objects (i.e., tagging action groups) offers powerful means for solving many real mining scenarios:

Definition 3.2.4. *Pair-Wise Aggregation Dual Mining Function. A Pair-Wise Aggregation (PA) Dual Mining Function,* $F_{pa} : \mathcal{G} \times b \times m \rightarrow \mathtt{float}$*, is a dual mining function with two component function* $F_p : g_i \times g_j \times b \times m \rightarrow \mathtt{float}$ *and* $F_a : \{s_1, s_2, \ldots\} \rightarrow \mathtt{float}$*, where* (g_i, g_j) *is a pair of distinct tagging action groups and each* s_i *is an intermediate score produced by* F_p*, such that:* $F_{pa}(\mathcal{G}, b, m) = F_a(\{F_p(g_i, g_j, b, m)\}$*,* $\forall g_i, g_j \in \mathcal{G}, i \neq j$*.*

We now present a few examples of the pair-wise dual mining function. The key to a pair-wise dual mining function is the pair-wise comparison function, $F_p(g_1, g_2, b, m)$, where g_1 and g_2 are tagging action groups, and $b \in \{\mathtt{users}, \mathtt{items}, \mathtt{tags}\}$, is a tagging behavior dimension, and $m \in \{\mathtt{similarity}, \mathtt{diversity}\}$, is a dual mining criterion.

3.2.1 User and Item Dimensions Dual Mining

Given a user describable tagging action group[1], its user dimension is effectively its user group description, i.e., a set of (attribute, value) pairs that describes the group. Therefore, given two user groups, g_1 and g_2, their similarity or diversity can be captured mainly in two ways: 1) structural distance between the user group descriptions and 2) set distance based on the items they have rated.

Let A be the set of user attributes shared between two user describable tagging action groups g_1 and g_2, an example of the pair-wise comparison function leveraging structural distance is the following:

$F_p(g_1, g_2, \texttt{users}, \texttt{similarity}) = \sum_{a \in A} sim(v_1, v_2)$

where $a.v_1$ and $a.v_2$ belong to the set of user attribute value pairs and sim can be a string similarity function that simply computes the edit distance between two values or a more sophisticated similarity function that takes domain knowledge into consideration. For example, a domain-aware similarity function can determine "New York City" to be more similar to "Boston" than to "Dallas". $F_p(g_1, g_2, \texttt{users}, \texttt{diversity})$ can be similarly defined using the inverse function.

Let $g_1.I$ and $g_2.I$ be the sets of items tagged by tuples in g_1 and g_2, respectively, an example of the pair-wise comparison function leveraging set distance is:

$F'_p(g_1, g_2, \texttt{users}, \texttt{similarity}) = \frac{|\{r | r \in g_1.I \wedge r \in g_2.I\}|}{|\{r | r \in g_1.I \vee r \in g_2.I\}|}$

which simply computes the percentages of items tagged by both groups (akin to Jaccard distance.) If numerical ratings are available for each tagging tuple, a more sophisticated set distance similarity function can further impose an additional constraint that an item is common to both groups if its average ratings in both are close. $F'_p(g_1, g_2, \texttt{users}, \texttt{diversity})$ can be similarly defined using the inverse function.

[1]Since the user and item dimensions share the same characteristics in the dual mining framework, we present here only the user dimension for simplicity.

3.2.2 Tag Dimension Dual Mining

The tag dimension is fundamentally different from the user and item dimensions. First, there is no fixed set of attributes associated with the tag dimension, therefore the structural distance does not apply. Second, tags are chosen freely by users using diverse vocabularies. As a result, a single tagging action group can contain a large number of tags. Both characteristics make comparing two sets of tags very difficult.

We propose a two-step approach for handling the tag dimension. First, we propose an initial step to summarize the set of all tags of a tagging action group into a smaller representative set of tags, called *group tag signature*. Second, we apply comparison functions to compute distance between signatures. Once again, we are not advocating any particular way of producing signatures and/or comparing them. Rather, we simply argue for the need for tag signatures and their comparisons.

Definition 3.2.5. *Group tag signature: Given a group of tagging actions* $g = \{\langle u_1, p_1, \mathtt{T}_1\rangle, \langle u_2, p_2, \mathtt{T}_2\rangle, \ldots\}$*, we aim to summarize the tags in* $T_1 \cup T_2 \cup \ldots$ *into a tag signature denoted as* $T_{rep}(g)$*. The general form of* $T_{rep}(g)$ *is* $\{(tc_1, w_1), (tc_2, w_2), \ldots\}$ *where* tc_i *is topic category (which can be a tag itself) and* w_i *is a weight representing the relevance of* g *for* c_i*.*

One can define several methods to compute tag signatures. For example, when tags are hand-picked by editors and hence the number of unique tags is small, a simple definition can be $T_{rep}(g) = \{(t, \mathtt{freq}(t)) \mid t \in \mathtt{T}_1 \cup \mathtt{T}_2 \cup \ldots\}$, where $\mathtt{freq}(t)$ computes how many times t is used in g.

Most collaborative tagging sites such as del.icio.us and YouTube, encourage users to create their own tags, thereby creating a long tail of tags. This raises challenges such as sparsity (many tags are used once or twice only) and the choice of different tags to express similar meanings. Techniques from information retrieval and machine learning such as tf*idf and Latent Dirichlet Allocation (LDA) [13, 14] can be

used for tag summarization. Also, a web service such as Open Calais[2] can be used to match a set of tags to a set of pre-defined categories through sophisticated language analysis and information extraction..

Comparing group tag signatures: When tagging action groups are represented as tag signatures over the same set of topics, we can leverage many existing vector comparison functions to compute the distance between any two group tag signature vectors pair-wisely. An example is simply cosine similarity as follows:

$F''_P(g_1, g_2, \texttt{tags}, \texttt{similarity}) = \texttt{cos}(\theta(T_{rep}(g_1), T_{rep}(g_2)))$,

where θ is the angle between the two vectors. $F''_P(g_1, g_2, \texttt{tags}, \texttt{diversity})$ can be defined similarly. The comparison can also be enhanced by using an ontology such as WordNet to compare entries of similar topics.

3.2.3 Concrete Problem Instances

We are now ready to formally define two of the concrete dual mining problems listed in Table 3.1 in the introduction. The first one (Problem 2 in Table 3.1) aims to find similar user sub-populations who agree most on their tagging behavior for a diverse set of items. The second one (Problem 4 in Table 3.1) finds diverse user sub-populations who disagree most on their tagging behavior for a similar set of items.

Problem 3.2.2. *Identify a set of tagging action groups,* $G^{opt} = \{g_1, g_2, \ldots\}$*, such that:*

- $\forall g_x \in G^{opt}$*,* g_x *is user- and/or item-describable;*
- $1 \leq |G^{opt}| \leq k$*;*
- $Support_G^{G^{opt}} \geq \alpha$*;*
- $F_1(G^{opt}, \texttt{users}, \texttt{similarity}) \geq q$*;*
- $F_2(G^{opt}, \texttt{items}, \texttt{diversity}) \geq r$*;*
- $F_3(G^{opt}, \texttt{tags}, \texttt{similarity})$ *is maximized.*

[2]https://www.opencalais.com/

where F_1 and F_2 are structural similarity based dual mining functions as defined in Definition described in Section 3.2.1, and F_3 is the LDA based tag dual mining function as described in Section 3.2.2.

For $k = 2$, $\alpha = 100$, $q = 0.5$, and $r = 0.5$, solving the problem on the full set of tagging action tuples in MovieLens [15] can give us the following G^{opt}:

$g_1 = \{\langle$gender, male$\rangle, \langle$age, young$\rangle, \langle$actor, j. aniston$\rangle,$

$(\texttt{comedy}, \texttt{drama}, \texttt{friendship})\}$

$g_2 = \{\langle$gender,male$\rangle, \langle$age, young$\rangle, \langle$actor, j. timberlake$\rangle,$

$(\texttt{drama}, \texttt{friendship})\}$

which illustrates the interesting pattern that male young users assign similar tags, `drama` and `friendship`, to movies with Jennifer Aniston and Justin Timberlake, the former for her involvement in the popular TV show "Friends" and the latter for his movie "The Social Network".

A closely related problem to Problem 3.2.2 is to inverse the similarity and diversity constraints for the user and item components, i.e., finding diverse user subpopulations who agree most on their tagging behavior for a similar set of items (Problem 3 in Table 3.1 in the introduction). Both problems focus on optimizing the tag similarity and therefore can be solved using similar techniques. Next, we define a problem that aims to identify groups that disagree on their tagging behavior.

Problem 3.2.5. *Identify a set of tagging action groups, $G^{opt} = \{g_1, g_2, \ldots\}$, such that:*

- $\forall g_x \in G^{opt}$, *g_x is user- and/or item-describable;*
- $1 \leq |G^{opt}| \leq k$;
- $Support_G^{G^{opt}} \geq \alpha$;
- $F_1(G^{opt}, \texttt{users}, \texttt{diversity}) \geq q$;
- $F_2(G^{opt}, \texttt{items}, \texttt{similarity}) \geq r$;

- $F_3(G^{opt}, \texttt{tags}, \texttt{diversity})$ *is maximized.*

where F_1, F_2, and F_3 are similarly defined as in Problem 3.2.2.

For $k = 2$, $\alpha = 100$, $q = 0.5$, and $r = 0.5$, solving the problem on the full set of tagging action tuples in MovieLens can give us the following G^{opt}:

$g_1 = \{\langle$gender,male$\rangle, \langle$age, teen$\rangle, \langle$genre,action$\rangle,$
$(\texttt{gun}, \texttt{special effects})\}$

$g_2 = \{\langle$gender,female$\rangle, \langle$age,teen$\rangle, \langle$genre,action$\rangle,$
$($violence, gory$)\}$

which illustrates teenaged male users and female users have entirely different perspectives on action movies. This gives a user a new insight that there is something about action movies that is causing the different reactions among different groups of users.

3.2.4 The TagDM Framework

The formal definitions for Problems 2 and 4 share a number of similarities. The objective is to identify a set of tagging action groups that maximizes similarity/diversity over a specific dimension. In addition, it also has constraints on the number of results, their coverage and whether similarity/diversity measures over individual dimensions exceed certain threshold.

In this chapter, we describe a constrained optimization based framework for tagging behavior mining. The framework is very general such that each of the concrete problem instances previously described can naturally be defined. Additionally, it is also easily extensible to explore complex objective functions and constraints depending on user needs.

We formally define the TagDM framework in the following definition.

Definition 3.2.6. *Tagging Behavior Dual Mining (TagDM) Problem. Given a triple* $\langle G, C, O\rangle$ *in the TagDM framework where* G *is the input set of tagging actions and* C, O *are the sets of constraints and optimization criteria respectively, the Tagging*

Behavior Dual Mining problem is to identify a set of tagging action groups, $G^{opt} = \{g_1, g_2, \ldots\}$ *for* $b \in \{\texttt{users}, \texttt{items}, \texttt{tags}\}$ *and* $m \in \{\texttt{similarity}, \texttt{diversity}\}$*, such that:*

- $\forall g_x \in G^{opt}$*,* g_x *is user- and/or item-describable;*
- $k_{lo} \leq |G^{opt}| \leq k_{hi}$*;*
- $Support_G^{G^{opt}} \geq \alpha$*;*
- $\forall c_i \in C, c_i.F(G^{opt}, b, m) \geq$ *threshold;*
- $\Sigma_{o_j \in O}, o_j.F(G^{opt}, b, m)$ *is maximized.*

Intuitively, TagDM aims to identify a set of user- and/or item-describable subgroups from input tagging actions, such that the dual mining constraints are satisfied and a dual mining goal is optimized. We now clearly see how this framework generalizes the common problem instances given in Section 3.2.3. The notation $c_i.F$ refers to a function (associated with constraint c_i) that measures similarity/diversity of the corresponding dimension. Similarly, the notation $o_j.F$ represents the dual mining function that operates over the specific dimension whose value must be optimized.

Notice that, the definition of TagDM problem is not limited to pair-wise aggregation dual mining functions, described in Definition 3.2.4. Pair-wise dual mining functions has a number of appealing properties - they are very common, can be computed efficiently, and offers a vast literature of techniques to exploit for developing fast solutions. However, restricting to pair-wise functions severely limits the expressiveness and generality of our TagDM framework.

General dual mining functions takes as input a set of tagging action groups and performs a holistic analysis over the entire set. They are not restricted to identifying and then accumulating the local properties at the tagging action group level. A number of optimization functions cannot be easily be defined in terms of pair-wise aggregation functions. Here are some examples:

- Identify a set of tagging action groups such that the variance of individual tagging actions is minimized. This helps us find user-describable and/or item-describable groups who share similar tagging behavior.
- Identify a set of tagging action groups such that the number of distinct tags they use is minimized. This helps us discover user-describable and/or item-describable groups where there is an universal agreement on the tag usage.
- Identify a set of tagging action groups such that their tag vocabulary (obtaining by performing union of all tagging actions) form a coherent ontology - either they all are synonyms, antonyms or occur within a specific distance of each other in an Ontology service such as Wordnet.

The generalized dual mining function can be formally defined as :

Definition 3.2.7. *Dual Mining Function. A Dual Mining Function,* $F : \mathcal{G} \times b \times m \rightarrow$ `float`*, takes as inputs:* $\mathcal{G}$*, a set of tagging action groups;* $b \in \{\texttt{users}, \texttt{items}, \texttt{tags}\}$*, a tagging behavior dimension;* $m \in \{\texttt{similarity}, \texttt{diversity}\}$*, a dual mining criterion; and produces a float score,* s*, that quantifies the mining criterion over the particular dimension for the set of tagging action groups.*

3.3 Complexity Analysis

In this section, we prove that the Tagging Behavior Dual Mining problem is NP-Complete. The decision version of the TagDM problem is defined as follows:

Given a triple $\langle G, C, O \rangle$, is there a set of tagging action groups $G^{opt} = \{g_1, g_2, \ldots\}$ such that $\sum_{o_j \in O}(o_j.Wt \times oj.F(G_{opt}, o_j.D, o_j.M) \geq \alpha$ subject to:

- $\forall g_x \in G^{opt}, g_x$ is user- and/or item-describable.
- $k_{lo} \leq |G^{opt}| \leq k_{hi}$
- $Support_G^{G^{opt}} \geq \alpha$
- $\forall c_i \in C, c_i.F(G^{opt}, c_i.D, c_i.M) \geq c_i.Th$

Theorem 3.3.1. *The decision version of the TagDM problem is NP-Complete.*

Proof. The membership of decision version of TagDM problem in NP is obvious. To verify NP-Completeness, we reduce Complete Bipartite Subgraph problem (CBS) to our problem and argue that a solution to CBS exists, *if and only if*, a solution our instance of TagDM exists. First, we show that the problem CBS is NP-Complete.

Lemma 3.3.1. *Complete bipartite subgraph problem (CBS) is NP-Complete.*

Proof. The decision version of CBS is defined as follows:

Given a bipartite graph $G' = (V_1, V_2, E)$ and two positive integers $n_1 \leq |V_1|, n_2 \leq |V_2|$, are there two disjoint subsets $V_1^{'} \subseteq V_1, V_2^{'} \subseteq V_2$ such that $|V_1^{'}| = n_1, |V_2^{'}| = n_2$ and $u \in V_1^{'}, v \in V_2^{'}$ implies that $\{u, v\} \in E$.

The membership of CBS in NP is obvious. We verify the NP-Completeness of the problem by reducing it to Balanced Complete Bipartite Subgraph (BCBS) problem which is defined as : Given a bipartite graph $G'' = (V_1^{''}, V_2^{''}, E^{'})$ and a positive integer $n^{'}$, find two disjoint subsets $V_1^{'''} \subseteq V_1^{''}, V_2^{'''} \subseteq V_2^{''}$ such that $|V_1^{'''}| = |V_2^{'''}| = n^{'}$ and $u \in V_1^{'''}, v \in V_2^{'''}$ implies that $\{u, v\} \in E^{'}$. This problem was proved to be NP-Complete by reduction from Clique in [16]. We can reduce BCBS to CBS by passing the input graph $G''(V_1^{''}, V_2^{''}, E^{'})$ of BCBS to CBS and setting n_1 and n_2 to $n^{'}$. If a solution exists for the CBS instances, then the disjoint subsets $V_1^{'''}, V_2^{'''}$ form a balanced complete bipartite subgraph in G''. □

We have already established that TagDM problem is in NP. To verify its NP-Completeness, we reduce CBS to the decision version of our problem. Given an instance of the problem CBS with $G^{'} = (V_1, V_2, E)$ and positive integers n_1, n_2, we construct an instance of TagDM problem such that there exists a complete bipartite subgraph induced by disjoint vertex subsets $V_1^{'} \subseteq V_1, V_2^{'} \subseteq V_2$ and $|V_1^{'}| = n_1, |V_2^{'}| = n_2$, *if and only if*, a solution to our instance of TagDM exists.

First, we create an user schema $S_U = \langle a_1, a_2, \ldots, a_{|V_2|} \rangle$ such that for each vertex $v_j \in V_2$, there exists a corresponding user attribute $a_j \in S_U$. Next, we define a set of users $U = \{u_1, u_2, \ldots, u_{|V_1|}\}$. Again, for each vertex $v_i \in V_1$ there exists a corresponding user $u_i \in U$.

For all pairs of vertices $(v_i, v_j), v_i \in V_1, v_j \in V_2$, we set $u_i.a_j$ to 1 if $\{v_i, v_j\} \in E$; else, we set it to a unique value such that $u_{x1}.a_{y1} \neq u_{x2}.a_{y2}$ unless $x_1 = x_2, y_1 = y_2$. Intuitively, we set the j-th attribute of i-th user to 1 if an edge exists between vertex pairs (v_i, v_j); else, we set it to a unique value that is not shared with any attribute of any user. One possible way to assign the unique attribute values is to pick a previously unassigned value from the set $[2, |V_1| \times |V_2| + 1]$. Since the number of possible edges is at most $|V_1| \times |V_2|$, this set suffices to generate unique attribute values.

We construct an instance of the TagDM problem where $I = \{i\}$ and $T = \{t\}$. This results in a set of tagging actions, $G = \{\langle u_1, i, t \rangle, \ldots, \langle u_{|V_1|}, i, t \rangle\}$ where only the user dimension plays a non-trivial role in determining the problem solution. Given a pair of users, the pairwise similarity function F_1 on user dimension measures their structural similarity by counting the number of attribute values that are shared between them. Intuitively, the problem collapses to that of finding a subset of users who share a subset of attributes. We then define our TagDM problem instance as : For a given a triple $\langle G, C, O \rangle$, identify a set G^{opt} of tagging action groups such that $F_3(G^{opt}, tags, m) \geq 0$ subject to:

- $1 \leq |G^{opt}| \leq n_1$
- $Support_G^{G^{opt}} \geq n_1$
- $F_1(G^{opt}, \texttt{users}, \texttt{similarity}) \geq n_2 \times \binom{n_1}{2}$

If there exists a solution to this TagDM problem instance, then there are n_1 users who have identical values for at least n_2 of their attributes. If two users u_x and u_y have same values for a set of attributes A, then for all attributes $a \in A$,

$u_x.a = u_y.a = 1$. In other words, whenever the attributes of two users overlap, the shared attributes can only take a value of 1. Any other symbol that was assigned is unique and cannot overlap by construction. If there exists a subset of attributes $A' \subseteq S_U$ and a subset of users $U' \subseteq U$, then the corresponding vertices in V_1 and V_2 form a complete bipartite subgraph solving the input instance of BCS. Thus TagDM problem is NP-Complete. □ □

A brute-force exhaustive approach (henceforth, referred to as Exact) to solve the TagDM problem requires us to enumerate all possible combinations of tagging action groups in order to return the optimal set of groups maximizing the mining criterion and satisfying the constraints. The number of possible candidate sets is exponential in the number of groups. Evaluating the constraints on each of the candidate sets and selecting the optimal result can thus be prohibitively expensive. Each tagging action group is associated with a group tag signature vector (the size of which is determined by the cardinality of the global set of tag topics), which may introduce additional challenges in the form of higher dimensionality. Therefore, we develop practical efficient solutions.

We develop two sets of algorithms that are capable of solving all 112 concrete problem instances that TagDM framework captures. The first set comprises of locality sensitive hashing based algorithms for handling TagDM problem instances maximizing similarity of tagging action components. The algorithms are efficient in practice, but cannot handle TagDM problem instances maximizing diversity. The second set is based on techniques employed in computational geometry for the facility dispersion problem and is our solution for diversity mining problem instances.

Next, we discuss ways to extend the general TagDM framework, i.e., Definition 3.2.6, and further develop two additional algorithms which are capable of han-

dling complex mining tasks. The first of these algorithms draws inspiration from hierarchical clustering methods and handles problem instances maximizing similarity as well as those maximizing diversity. The technique is particularly useful for handling mining tasks which involve additional conditions and criteria in the optimization goal of the general TagDM framework. The second of these algorithms consider the more general mining function in Definition 3.2.7 as opposed to the pairwise measures used by all the remaining algorithms.

3.4 Algorithms

3.4.1 Similarity Maximization: LSH Based Algorithms

The first of our solutions is based on locality sensitive hashing (LSH) which is a popular technique to solve nearest neighbor search problems in higher dimensions [17]. LSH is preferred over several seemingly promising techniques (such as constructing efficient indices, which suffers from the curse of dimensionality) because it scales gracefully to higher dimensional data, is efficient and provides theoretical guarantees.

LSH performs a probabilistic dimensionality reduction of high dimensional data by projecting input items in higher dimension to a lower dimension such that items that were in close proximity in the higher dimension retain their proximity in lower dimensional space with high probability. LSH *hashes* an input item such that similar input items fall into the same bucket (i.e., uniquely definable hash signature) with high probability. In other words, all the input items in the same bucket are highly likely to be similar.

A classical application of LSH is to identify nearest neighbors efficiently. Typically, the input items in LSH are high dimensional vectors such as multimedia content. There are two major parameters for LSH : d', the number of hash functions (which

also determines the lower dimension to which items are projected to) and l, the number of hash tables. A hash table is associated with d' hash functions randomly chosen from a hash family that is problem specific. Each hash function accepts a vector in high dimension and returns a scalar. By concatenating the result of d' hash functions, we get a d' dimensional vector that also forms a distinct hash signature for the item. All the input items that have identical hash signatures are said to fall in the same bucket. This process is repeated for all the l hash tables. Intuitively, each hash table can be considered as a partition of the input items such that similar items fall into same partition with high probability. Typically, each hash table results in different partition of the input data based on the hash functions associated with it. Once the input data has been projected to lower dimensions, it can then be used for applications such as identifying nearest neighbors of a given item. For each of the l hash tables, we project the input query item and identify which bucket it falls into. All the items that co-occur in any of the l buckets the input query items fall into are considered as candidate nearest neighbors. However, the set of candidates is usually much smaller than the size of input items and allows nearest neighbor(s) to be computed efficiently.

LSH guarantees a lower bound on the probability that two similar input items fall into the same bucket in the projected space and also the upper bound on the probability that two dissimilar vectors fall into the same bucket. For any pair of points in a high-dimensional space and a given hash function h, P_1 is the probability of two close points receiving the same value after hashing. P_2 is the probability of two far-apart points falling receiving the same value after hashing. We want $P_2 < P_1$ and typically $P_1 > \frac{1}{2}$. Formally, given a pair of nearby points p_1, q_1 (defined as points within distance R_1) and far-apart points p_2, q_2 (those with distance at least $R_2 = cR_1$) we have:

$$P[h(p_1) = h(q_1)] \geq P_1 \text{ for } dist(p_1, q_1) \leq R_1$$
$$P[h(p_2) = h(q_2)] \leq P_2 \text{ for } dist(p_2, q_2) \geq R_2 \quad (3.1)$$

Two items fall into the same bucket if they receive identical values for each of the d' independently chosen hash functions. After projecting the input items from higher dimension d to a lower dimension d', we can see that the following probabilistic bounds hold for similar/dissimilar items falling into same bucket :

$$P(\text{similar items falling in same bucket}) \geq P_1^{d'}$$
$$P(\text{dissimilar items falling in same bucket}) \leq P_2^{d'} \quad (3.2)$$

Given the background on LSH, we now adapt it to select a set of tagging action groups that are similar in their tagging behavior based on locality sensitive hashing. Recall that our input is the set G of n tagging action groups (i.e., n d-dimensional tag signature vectors, where d is the cardinality of the global set of tag topics mentioned in Section 3.2.2) using a pair-wise comparison function $F_p''(g_1, g_2, \texttt{tags}, \texttt{similarity})$ that operates on group tag signature vectors in order to optimize tag similarity. Our expected result is a set of k tagging action groups G^{opt} such that they have the least average pair-wise distance between them

Note that, our LSH based algorithms works for Problems 1, 2 and 3 in Table 3.1 maximizing tag similarity. We first introduce an algorithm that returns the set of tagging action groups G^{opt}, $1 \leq |G^{opt}| \leq k$ having maximum similarity in tagging behavior (Column O in Table 3.1) and then discuss additional techniques to include the multiple hard constraints into the solution (Column C in Table 3.1).

3.4.1.1 Maximizing Similarity based on LSH: SM-LSH

Our LSH based algorithm SM-LSH deals with TagDM problem instances optimizing tag SiMilarity. The classical usage of LSH is to find the nearest neighbor(s) for a given query item. However, in TagDM problems, there is no specific query item -

instead, we want to identify a set of k tagging action groups that have high pair-wise similarity over their tagging behavior. We reconcile these two seemingly different usages by exploiting the fact that given a bucket, all points in it are highly likely to be similar. We employ LSH to partition the tagging action groups such that similar groups fall into the same bucket. The next step is to choose the best set of tagging groups for a hash table. We identify the best k-subset for each bucket and pick the best set over all buckets. Different sets are comparable using the dual mining scoring function. We repeat the process for each hash table and finally choose the best set among all hash tables.

One of the key requirement for good performance of LSH is the careful selection of the family of hashing functions. In SM-LSH, we use the LSH scheme proposed by Charikar [18] which employs a family of hashing functions based on cosine similarity. It is possible to utilize other distance measures such as Euclidean, Jaccard, Earth-Movers etc. The only change involves how the hash functions are chosen and how the probability values P_1 and P_2 are computed. As discussed in Section 3.2.2, the cosine similarity between two group tag signature vectors is defined as the cosine of the angle between them and can be defined as:

$$cos(\theta(T_{rep}(g_x), T_{rep}(g_y))) = \frac{|T_{rep}(g_x).T_{rep}(g_y)|}{\sqrt{|T_{rep}(g_x)|.|T_{rep}(g_x)|}}$$

The algorithm computes a succinct hash signature of the input set of n tagging action groups by computing d' independent dot products of each d-dimensional group tag signature vector $T_{rep}(g_x)$, where $g_x \subseteq G$ with a random unit vector $\vec{r}$ and retaining the sign of the d' resulting products. This maps a higher d-dimensional vector to a lower d'-dimensional vector ($d' \ll d$). Each entry of $\vec{r}$ is drawn from a 1-dimensional Normal distribution N(0,1) with zero mean and unit variance. Alternatively, we can generate a spherically symmetric random vector $\vec{r}$ of unit length from the d-

dimensional space. The LSH function for cosine similarity for our problem is given by the following Theorem 3.4.1 adapted from [18]:

Theorem 3.4.1. *Given a collection of n d-dimensional vectors where each vector $T_{rep}(g_x)$ corresponds to a $g_x \subseteq G$, and a random unit vector $\vec{r}$ drawn from a 1-dimensional Normal distribution N(0,1), define the hash function h_r as:*

$$h_r(T_{rep}(g_x)) = \begin{cases} 1 & if\ \vec{r}.T_{rep}(g_x) \geq 0 \\ 0 & if\ \vec{r}.T_{rep}(g_x) < 0 \end{cases}$$

Then for two arbitrary vectors $T_{rep}(g_x)$ and $T_{rep}(g_y)$:

$$P[h_r(T_{rep}(g_x)) = h_r(T_{rep}(g_y))] = 1 - \frac{\theta(T_{rep}(g_x), T_{rep}(g_y))}{\pi}$$

where $\theta(T_{rep}(g_x), T_{rep}(g_y))$ is angle between two vectors.

The proof of the above Theorem 3.4.1 establishing that the probability of a random hyperplane (defined by $\vec{r}$ to hash input vectors) separating two vectors is directly proportional to the angle between the two vectors follows from Goemans et. al's theorem [19]. Any pair-wise dual mining function for comparing tag signatures must satisfy such properties. We represent the d'-dimensional-bit LSH function as:

$$g(T_{rep}(g_x)) \quad = \quad [h_{r1}(T_{rep}(g_x)), \ldots, h_{rd'}(T_{rep}(g_x))]^T$$

For d' LSH functions and from (2), the probability of similar tag signature vectors g_x and g_y falling into the same bucket for all d' hash functions is at least :

$$P(g(T_{rep}(g_x)) = g(T_{rep}(g_y))) \geq \left(1 - \frac{\theta(T_{rep}(g_x), T_{rep}(g_y))}{\pi}\right)^{d'}$$

Now, each input vector is entered into the l hash tables indexed by independently constructed hash functions $g_1(T_{rep}(g_x))$, ..., $g_l(T_{rep}(g_x))$. Using this LSH scheme, we hash the tag vectors to l different d'-dimensional hash signatures(or, buckets). The total number of possible hash signatures in each of the l lower dimen-

sional space is $2^{d'}$. However, the maximum bound on the number of buckets in each of the lower dimensional space is n.

While LSH is generally used to find the nearest neighbors for new items, we take the novel approach of finding the right bucket to output as result of our problem based on checking for the number of tagging action groups in result set and ranking by scoring function. For each of the l hash tables, we first check for satisfiability of $1 \leq |G^{opt}| \leq k$ in each bucket and then rank the buckets based on the scoring function in order to determine the result set of tagging action groups G^{opt} with maximum similarity.

Theorem 3.4.2. *Given a collection of n d-dimensional tag signature vectors where each pair of vectors $T_{rep}(g_x)$ and $T_{rep}(g_y)$ corresponds to a $g_x, g_y \subseteq G$, the probability of finding result set G^{opt} of k most similar vectors by SM-LSH is bounded by:*

$$P(G^{opt}) \geq max\left(0, 1 - \sum_{g_x, g_y \in G^{opt}} \left[1 - \left(1 - \frac{\theta(T_{rep}(g_x), T_{rep}(g_y))}{\pi}\right)^{d'}\right]\right)$$

Proof. The probability of finding the set of tagging action groups G^{opt}, $1 \leq |G^{opt}| \leq k$ having maximum similarity in tagging behavior, $P(G^{opt})$:

$= 1$ - P(at least one of ${}^{k}C_2$ vector pairs belongs to different buckets)

≥ 1 - $\sum_{g_x, g_y \in G^{opt}}$ P($T_{rep}(g_x)$,$T_{rep}(g_y)$ in different buckets)

≥ 1 - $\sum_{g_x, g_y \in G^{opt}}$ [1 - P($T_{rep}(g_x)$,$T_{rep}(g_y)$ in same buckets)]

≥ 1 - $\sum_{g_x, g_y \in G^{opt}} \left[1 - \left(1 - \frac{\theta(T_{rep}(g_x), T_{rep}(g_y))}{\pi}\right)^{d'}\right]$ □

Notice that the lower bound provided by Theorem 3.4.2 is conservative. We start with the probability that at least one of the ${}^{k}C_2$ vector pair belongs to different buckets. If $E_{x,y}$ denotes the event that vectors g_x and g_y fall in different buckets, our objective is to estimate $P(\cup_{x,y \in G^{opt}} E_{x,y})$. For this purpose, we use union bound which states that $P(\cup_{x,y \in G^{opt}} E_{x,y}) \leq \sum_{x,y \in G^{opt}} P(E_{x,y})$. While this a weak bound,

it still expresses that our method has a probabilistic guarantee, as opposed to other heuristics. We experimentally validate in Section 3.6.1 that SM-LSH works extremely well. We leave the derivation of tighter bounds for Theorem 3.4.2 for future work.

Algorithm 4 is the pseudo code of our SM-LSH algorithm. This algorithm may return null result if post-processing of all l hash tables yields no bucket satisfying $1 \leq |G^{opt}| \leq k$. In other words, there are no set of tagging action groups in any bucket such that they satisfy the constraints (size of set, coverage, user/item overlap) of our problem instance. This motivates us to tune SM-LSH by *iterative relaxation* that varies the input parameter d' in each iteration. Decreasing the parameter d' increases the expected number of tagging action groups hashing into a bucket, thereby increasing the chances of our algorithm finding the result set. We start with an initial value of d' and reduce the parameter systematically (in a manner similar to performing a binary search) such that we find a value of d' such that it has some buckets that contain more than k tagging action groups satisfying the constraints. However, the expected size of bucket must not be too high as that will make the problem of finding the best k-subset of tagging action groups in a bucket very expensive. (please see Subsection 3.4.1.3 for additional discussion).

Example 3.4.1. *Let us consider one of the Problems 1, 2, or 3 in Table 3.1 where the objective is to optimize tag similarity. Consider a dataset where the input G of tagging action groups consists of $n = 5$ 3-dimensional tag signature vectors. Let the group tag vectors be $T_{rep}(g_1) = [0.6, 0.2, 0.2]$, $T_{rep}(g_2) = [0.1, 0.7, 0.1]$, $T_{rep}(g_3) = [0.1, 0.1, 0.8]$, $T_{rep}(g_4) = [0.6, 0.4, 0.0]$ and $T_{rep}(g_5) = [0.4, 0.2, 0.4]$. The tagging vectors can be obtained via multiple methods including tf-idf or LDA. The dimensionality $d = 3$ of the vectors correspond to the tag topics under consideration, and can be words like* `love`*,* `oscar winning`*,* `gory`*, etc. The tagging action groups*

Algorithm 4 SM-LSH (G, O, k, d', l)**:** G^{opt}

//Main Algorithm

$min = 1$
$max = d'$
$T^U_{rep} \leftarrow \{\}$; $T^I_{rep} \leftarrow \{\}$
if $C_1.m =$ `similarity` **then**
  $T^U_{rep} \leftarrow$ Unarize user vector
end if
if $C_2.m =$ `similarity` **then**
  $T^I_{rep} \leftarrow$ Unarize item vector
end if
for $x = 1$ to n **do**
  $T_{rep}(g_x) \leftarrow T^U_{rep}(g_x) + T^I_{rep}(g_x) + T_{rep}(g_x)$
end for
repeat
  $Buckets \leftarrow$ LSH(G, d', l)
  $G^{opt} \leftarrow$ MAX(Rank($Buckets$, k))
  if $G^{opt} = null$ **then**
    $max = d' - 1$
  else
    $min = d' + 1$
  end if
  $d' = (min + max)/2$
until $(min > max)$ **or** $(G^{opt} \neq null)$
return G^{opt}

//LSH(G, d', l): Buckets

for $z = 1$ to l **do**
  for $x = 1$ to n **do**
    for $y = 1$ to d' **do**
      Randomly choose $\vec{r}$ from d-dimensional Normal distribution N(0, 1)
      if $\vec{r}.T_{rep}(g_x) \geq 0$ **then**
        $h_{ry}(T_{rep}(g_x)) \leftarrow 1$
      else
        $h_{ry}(T_{rep}(g_x)) \leftarrow 0$
      end if
      $g_z(T_{rep}(g_x)) = [h_{r1}(T_{rep}(g_x)), .., h_{rd'}(T_{rep}(g_x))]^T$
    end for
  end for
end for
$Buckets \leftarrow g_1(T_{rep}(g_x)) \cup \cdots \cup g_l(T_{rep}(g_x))$
return $Buckets$

are user-describable. Specifically, let the descriptions of g_1 *be* $\langle$gender, female$\rangle$, g_2 *be* $\langle$state, texas$\rangle$, g_3 *be* $\langle$state, california$\rangle$, g_4 *be* $\{\langle$gender, female$\rangle, \langle$state, texas$\rangle\}$ *and* g_5 *be* $\{\langle$gender, female$\rangle, \langle$state, california$\rangle\}$ *respectively. The objective is to identify the result set* G^{opt} *of* $k = 2$ *groups having maximum similarity in tagging behavior for the dataset under consideration. The naive way (Exact Algorithm) would perform* $\binom{n}{k} = \binom{5}{2} = 10$ *comparisons to find the best pair, which is not always a feasible solution. Our SM-LSH algorithm helps us retrieve* G^{opt} *in the following manner.*

Let the LSH parameters be $d' = 2$ *and* $l = 1$*; let the randomly chosen vectors be* $r_1 = [+1, -1, 0]$, $r2 = [-1, -1, +1]$. *We reduce the dimensionality of each vector from* $d = 3$ *to* $d' = 2$. *For a vector* $T_{rep}(g_x)$, *the first component of its corresponding lower dimensional representation is* $r_1.T_{rep}(g_x)$, *while its second component is* $r_2.T_{rep}(g_x)$. *If a component is non negative, we set it to 1 else to 0. As an example, given vector* $T_{rep}(g_1)$, *its lower dimensional representation is* $[T_{rep}(g_1) \cdot r_1, T_{rep}(g_1) \cdot r_2]$ $= [0.426, -0.52]$. *This is then transformed to* $[1, 0]$. *Repeating the same procedure for the other vectors, we get their lower dimensional representations as:* $g_2 = [0, 0]$, $g_3 = [1, 1]$, $g_4 = [1, 0]$ *and* $g_5 = [1, 0]$. *Out of the* $2^{d'} = 2^2 = 4$ *possible buckets, we have 3 non empty buckets. SM-LSH finds out that the only bucket with at least* $k = 2$ *elements is* $[1, 0]$. *This bucket, incidentally, also contains the optimal solution. This is identified by finding the best* $k = 2$ *tagging action groups in the set* $\{g_1, g_4, g_5\}$ *by enumerating all possible pairs. The result* $G^{opt} = \{g_4, g_5\}$ *can be interpreted as:* female users *in the dataset under consideration have similar tagging behavior.*

Note that, we have considered one of the Problems 1, 2, or 3 in Table 3.1 in our running example. Each of the problems could have multiple hard constraints. Techniques to refine the SM-LSH results for satisfiability of the hard constraints are discussed later in Sections 3.4.1.2 and 3.4.1.3. □

Complexity Analysis: For each round, the pre-processing of locality sensitive hashing time is bounded by $O(nld)$. The number of rounds is logarithmic in d in the worst case. The time complexity for post-processing phase where the buckets are ordered for ranking by scoring function depends on the maximum number of non empty buckets B for any hash table and the size of the largest bucket n_b resulting in $O\left(Bl\binom{n_b}{k}\right)$. Notice that $\binom{n_b}{k}$ gives the complexity of finding the best k-subset in a bucket. The space complexity of the algorithm is $O(nl)$ since there are l hash tables and each table has at most n buckets.

SM-LSH is a fast algorithm with interesting probabilistic guarantees and is advantageous, especially for high-dimensional input vectors. However, the hard constraints along user and item dimensions are not leveraged in the optimization solution so far. Next, we discuss approaches for accommodating the multiple hard constraints.

3.4.1.2 Dealing with Constraints (Filtering): SM-LSH-Fi

A straightforward method of refining the result set of SM-LSH for satisfiability of all the hard constraints in TagDM problem instances is post-processing or Filtering. We refer to this algorithm as SM-LSH-Fi. For each of the l hash tables, we first check for satisfiability of the hard constraints in each bucket and then rank the buckets (satisfying hard constraints) according to the scoring function in order to determine the result set of tagging action groups G^{app} (We represent G^{opt} as G^{app} since LSH based technique now perform approximate nearest neighbor search) with maximum similarity. Such post-processing of buckets for satisfiability of hard constraints may also return null results, if post-processing of hash tables yields no bucket satisfying all the hard constraints. Therefore, we propose a smarter method that folds the hard constraints concerning similarity as part of vectors in high-dimensional space, thereby increasing the chances of similar groups hashing into the same bucket.

3.4.1.3 Dealing with Constraints (Folding): SM-LSH-Fo

Problems 2 and 3 in Table 3.1 has two out of the three tagging action components to be mined for similarity. In order to explore the main idea of LSH, we Fold the hard constraints maximizing similarity as soft constraints into our SM-LSH algorithm in order to hash similar input tagging action groups (similar with respect to group tag signature vector and user and/or item attributes) into the same bucket with high probability. We refer to this algorithm as SM-LSH-Fo. We fold the user or item similarity hard constraints in Problems 2 and 3 respectively into the optimization goal and apply our algorithm, so that tagging action groups with similar user attributes or similar item attributes, and similar group tag signature vectors hash to the same bucket. For each tagging action group $g_x \subseteq G$, we represent the categorical user attributes or item attributes as a boolean vector and concatenate it with $T_{rep}(g_x)$. We map n vectors from a higher $(d + \sum_{i=1}^{|S_U|} \sum_{j=1}^{|a_i|} |a_i = v_j|)$ dimensional space for users (replace $|S_U|$ with $|S_I|$ for items) to a lower d' dimensional space. Similar to Algorithm 4, we consider l LSH hash functions and then post-process the buckets for satisfiability of the remaining constraints in order to retrieve the final result set of tagging action groups G^{app} with maximum optimization score. Problem 1 in Table 3.1 has all three tagging action components set to similarity. In this case, we build one long vector for each tagging action group $g_x \subseteq G$ by concatenating boolean vector corresponding to categorical user attributes, boolean vector corresponding to categorical item attributes and numeric tag topic signature vector $T_{rep}(g_x)$. The dimensionality of the high-dimensional space for Problem 1 is $d + \sum_{i=1}^{S_U} \sum_{j=1}^{|a_i|} |a_i = v_j| + \sum_{i=1}^{S_I} \sum_{j=1}^{|a_i|} |a_i = v_j|$.

Practical Considerations: While Theorem 3.4.2 establishes the theoretical probabilistic bound of finding the optimal result set, there are a number of practical issues.

The first issue is how to set the initial values for parameters d' and l. Ideally, the parameters must result in buckets such that their expected size be k. This will ensure that all the tagging action groups in the bucket becomes a candidate output set. We can use Theorem 1 in [20] to set the initial values to be

$$d' = \log_{1/P_2} \frac{n}{k}$$
$$l = \left(\frac{n}{k}\right)^{\rho}$$

where $\rho = \frac{\ln 1/P_1}{\ln 1/P_2}$. However, there are a number of work including [21, 22, 23] on LSH-parameter tuning that can be used as well.

An important issue to notice is that LSH is a Monte Carlo randomized algorithm. In other words, while the probability of finding the optimal solution is reasonable, it is possible that we did not find in our first attempt. There are two possible ways to boost the success probability. First, we can increase the number of trials of our algorithm but keeping the same values for parameters d' and l. An alternate approach, which we have used in our algorithm, is to reduce the number of hash functions d'. In both cases, each round of our algorithms are *independent*. In other words, the hash functions are chosen from scratch. We then identify the best set of tagging action groups for each round and choose the best over all rounds.

Consider the approach where we change the parameter d'. Intuitively, as the value of d' decreases, the expected number of input items that fall into any bucket increases. In the extreme case, for $d' = 1$, we expect half the points to fall in the bucket $h_r(.) = 0$ and half in $h_r(.) = 1$. This particular design choice is appealing as the expected size of buckets increases, the chances of the optimal set of tagging action groups to belong to the same bucket also increases.

The next design choice involves how to choose the dimensionality d'' for the next iteration. For example, we can decrement the current number of hash functions

to project to the next lower dimension. i.e. $d'' = d' - 1$. Alternatively, we can be more aggressive and reduce the dimensionality by some constance factor (such as $d'' = \frac{d'}{2}$). There is a cost-benefit tradeoff here. On one hand, when the dimensionality is cut by half, the expected number of items falling into a bucket dramatically increases. However, there is a corresponding increase in the probability of finding the optimal set of points also. We choose an approach that balances runtime vs success probability. If we have not identified enough candidates for a given value of d', the value for next invocation is chosen as $d'' = \frac{d'}{2}$. However, if the resulting bucket sizes are too large for d'' (thereby finding the best subset of k points in a bucket would be very expensive), we then choose a new dimension half-way between d'' and d'. In the worst case, the value of d' goes all the way to 1 and our algorithm degenerates to Exact. We note that there exist a number of theoretical and empirical work on tuning LSH parameters. The most common techniques to handle failure (in our case, it is the lack of buckets with atleast k tagging action groups in it) are reducing the dimension [21, 22, 23, 24], choosing the best parameters based on their performance over different samples over dataset[25] or based on their distance profiles [21] and finally multiprobing [26]. We chose the approach of reducing the dimension as it is the most intuitive and requires the least amount of additional information.

Both SM-LSH-Fi and SM-LSH-Fo are efficient algorithms for solving TagDM similarity maximization problem instances and readily out-performs the baseline Exact, as shown in Section 3.6. However, there are other instantiations namely, Problems 4, 5 and 6 in Table 3.1 which concern tag diversity maximization. Since it is non-obvious how the hash function may be inversed to account for dissimilarity while preserving the properties of LSH, we develop another set of algorithms (less efficient than LSH based, as per complexity analysis) in Section 3.4.2 for diversity problems. They are based on the popular Facility Dispersion problem, as described next.

3.4.2 Diversity Maximization: FDP Based Algorithms

The second of our algorithmic solutions borrows ideas from techniques employed in computational geometry, which model data objects as points in high dimensional space and determine a subset of points optimizing some objective function. Such geometric problem examples include clustering a set of points in euclidean space so as to minimize the maximum intercluster distance, computing the k^{th} smallest or largest inter-point distance for a finite set of points in euclidean space, etc. Since we consider tagging action groups as tag signature vectors, and since the cardinality of the global set of topics (that, in turn, determines the size of each vector) is often high, computational geometry based approach is an intuitive choice to pursue.

We focus on a specific geometric problem, namely the facility dispersion problem (FDP), which is analogous to our TagDM problem instances, finding the tagging action groups maximizing the mining criterion. The facility dispersion problem deals with the location of facilities on a network in order to maximize distances between facilities, minimize transportation costs, avoid placing hazardous materials near housing, outperform competitors' facilities, etc. We consider the problem variant in Ravi et al.'s paper [27] that maximizes some function of the distances between facilities. The optimality criteria considered in the section are MAX-MIN (i.e., maximize the minimum distance between a pair of facilities) and MAX-AVG (i.e., maximize the average distance between a pair of facilities). Under either criterion, the problem is known to be NP-hard by reduction from the Set Cover problem, even when the distances satisfy the triangle inequality [28]. The authors present an approximation algorithm for the MAX-AVG dispersion problem, that provides a performance guarantee of 4. The algorithm initializes a pair of nodes (i.e., facilities) which are joined by an edge of maximum weight and adds a node in each subsequent iteration which has the maximum distance to the nodes already selected.

The facility dispersion problem solution provides an approach to determine a set of tagging actions groups that have maximum average pair-wise distance, i.e., that are dissimilar in their tagging behavior. In fact, this approach may also be extended to determine a set of tagging action groups that are similar in their behavior, unlike the LSH based algorithm in Section 3.4.1 (which works only for similarity, not diversity). We consider each of the input n tagging action groups as d-dimensional tag signature vector in a unit hypercube and intend to identify k vectors with maximum average pairwise distance between them. We compare the input set G of n tagging action groups using a pairwise comparison function $F''_p(g_1, g_2, \texttt{tags}, diversity)$ that operates on tagging action group signature vectors; and return the set of tagging groups $\leq k$ having maximum diversity in tagging behavior.

Our FDP based algorithms work for Problems 4, 5 and 6 in Table 3.1 maximizing tag diversity. We first introduce an algorithm that returns the set of tagging action groups having maximum diversity in tagging behavior (Column O in Table 3.1) and then discuss additional techniques to handle the multiple hard constraints in the solution (Column C in Table 3.1).

3.4.2.1 Maximizing Diversity based on FDP: DV-FDP

Our FDP based algorithm DV-FDP handles TagDM problem instances optimizing tag DiVersity. Given an input set G of n tagging action groups, each having a numeric tag signature vector $T_{rep}(g_x)$, where $g_x \subseteq G$, we build the result set G^{app} (we represent the result set as G^{app} since the technique returns approximate solution) by adding a tagging action group in each iteration which has the maximum distance to the groups already included in the result set. Again, we use cosine similarity measure between two tag signature vectors for determining the distance since the distance metric hold triangular inequality property. Thus, our DV-FDP attempts to find one

tight set of k groups with maximum average pairwise distance between them. The approximation bounds for this algorithm follows from [27]:

Theorem 3.4.3. *Let I be an instance of the TagDM problem maximizing the mining criterion with $k \geq 2$ and no other hard constraints, where the collection of n d-dimensional vectors are in a unit hypercube satisfying the triangle inequality. Let G^{opt} and G^{app} denote respectively the optimal set of k tagging action groups returned by Exact and DV-FDP algorithms. Then $G^{opt}/G^{app} \leq 4$.*

Algorithm 5 is the pseudo-code of our DV-FDP algorithm. Once the $n \times n$ distance matrix S^G is built using the cosine distance function, the implementation exhaustively scans S for determining the best add operation in each of the subsequent iterations. If A represents the result set, the objective is to find an entry from $G - A$ to add to A, such that its total sum of weight to a node in A is maximum.

Algorithm 5 DV-FDP (G, O, k): G^{app}

//Main Algorithm
1: $S^G \leftarrow$ Compute $n \times n$ Distance Matrix(G)
2: $\{g_x, I_x, g_y, I_y\} \leftarrow \text{MAX}(S^G)$
3: $A \leftarrow [g_x, g_y]$
4: **while** $A \neq k$ **do**
5: $\quad g_z \leftarrow \Sigma_{\{z' \in [A], z \in [G-A]\}} \text{MAX}(S^{G-A})$
6: $\quad A \leftarrow [A, g_z]$
7: **end while**
8: $G^{app} \leftarrow A$
9: return G^{app}

Example 3.4.2. *Let us consider one of the Problems 4, 5, or 6 in Table 3.1 where the objective is to optimize tag diversity. Let us consider the same dataset and input set G of tagging action groups, as in Example 3.4.1. The objective is to identify the result set G^{app} of $k = 3$ groups having maximum diversity in tagging behavior. Using the cosine distance (computed as 1.0 - cosine similarity score for capturing diversity)*

Table 3.2. Distance matrix S^G

	g_1	g_2	g_3	g_4	g_5
g_1	0.00	0.54	0.55	0.08	0.10
g_2	0.54	0.00	0.72	0.34	0.49
g_3	0.55	0.72	0.00	0.83	0.22
g_4	0.08	0.34	0.83	0.00	0.26
g_5	0.10	0.49	0.22	0.26	0.00

as our distance measure, DV-FDP functions as follows. The first step is to compute the matrix S^G, which is shown in Table 3.2.The pair of tagging action groups with highest distance between them are g_3 and g_4. These form our initial approximate group $G^{app} = \{g_3, g_4\}$. Next, we find the tagging action group that has the largest aggregate distance from g_3 and g_4. The tagging action group has an aggregate distance of $0.55 + 0.08 = 0.63$ from the set g_3, g_4. The corresponding values for g_2 and g_5 are 1.06 and 0.48 respectively. This means that g_2 becomes the next member of the output group and the algorithm returns $G^{app} = \{g_2, g_3, g_4\}$. Observe that in this example, the approximate and optimal answers happen to be identical, which may not be the case in general. The result $G^{app} = \{g_2, g_3, g_4\}$ can be interpreted as: users in California *and* Texas *have diverse tagging behavior for the dataset under consideration. Note that, we have considered one of the Problems 4, 5, or 6 in Table 3.1 in our running example. Each of the problems have multiple hard constraints. Techniques to refine the DV-FDP results for satisfiability of the hard constraints are discussed later in Sections 3.4.2.2 and 3.4.2.3.* □

Complexity Analysis: The complexity of the implementation of the DV-FDP algorithm is $O(n^2 + nk)$, i.e., $O(n^2)$ due to operations around the $n \times n$ distance matrix S^G. The space complexity of the algorithm is $O(n^2)$. Thus the complexity of DV-FDP is polynomial in the number of groups. Recall that the number of groups n

is given by $\prod_{a \in S_U \cup S_I}(|a| + 1)$, where $|a|$ returns the number of distinct values taken by the categorical attribute a. For example, if there are two attributes, say gender and states with cardinality 2 and 50 respectively, the maximum number of groups induced by these attributes is $(2 + 1) \times (50 + 1)$. The additional 1 is added to account for the scenario when attribute takes no values (e.g., age = '*'). Note that, the maximum number of groups is dependent only on the cardinality of user and item attributes. However, in practice, the *actual* number of groups is substantially smaller due to the sparsity of the datasets. Note that, our LSH based algorithms have better space and time complexity than FDP based algorithms. However, experiments in Section 3.6.1 show comparable execution time for LSH and FDP based algorithms in a practical setting. Like SM-LSH, this algorithm does not leverage the hard constraints along user and item dimensions into the optimization solution too. We now illustrate approaches for including the multiple hard constraints into the solution.

3.4.2.2 Dealing with Constraints (Filtering): DV-FDP-Fi

Similar to SM-LSH-Fi, a straightforward method of refining the result set of groups for satisfiability of all the hard constraints in TagDM problem instances is post-processing or Filtering. We refer to this algorithm as DV-FDP-Fi. Once the result set G^{app} of k groups is identified, we post-process it to retrieve the relevant answer set of tagging action groups, satisfying all the hard constraints. Now, such post-processing of the result set for satisfiability of hard constraints may return null results frequently and hence we propose a smarter algorithm that folds some of the hard constraints into DV-FDP, thereby decreasing the chances of hitting a null result.

3.4.2.3 Dealing with Constraints (Folding): DV-FDP-Fo

In contrast to general DV-FDP algorithm whose objective is to add groups to the result set greedily so that average pair-wise distance is maximized, we want to re-

trieve the set in each iteration whose members, besides being dissimilar, satisfy many other constraints. In DV-FDP, the greedy add operation in Line 5 of Algorithm 5 maximizes tag diversity. If the algorithm includes a bad tagging action group to the result set in an iteration, the algorithm may return null result or an inferior approximate result, after final filtering of the result set for hard constraint satisfiability. Therefore, we propose our second approach in which hard constraints maximizing diversity are Folded into the add operation. We refer to this algorithm as DV-FDP-Fo. During each new group addition to the result set, we not only check for the pair with maximum distance, but also check for the satisfiability of the diversity maximization hard constraints on user and item dimension, if any. The algorithm terminates when the number of groups in result set equals k. Once the result set of k groups is identified, we post-process the set for satisfiability of the hard constraint(s) and support constraint, in order to retrieve the final result of tagging action groups $G^{app'}$.

3.5 Extensions to TagDM Framework

The TagDM framework in Definition 3.2.6 consists of tagging behavior dimensions (i.e., users, items and tags), a set of constraints and optimization goal, and the mining function (i.e., similarity and diversity). In this section, we discuss natural extensions to TagDM framework that allow it to be more expressive and practical, and design additional algorithms for solving them. Our extensions are two fold: first, we allow the inclusion of conditions over the dimension(s) in the optimization goal; second, we generalize the mining function from pair-wise aggregation to arbitrary dual mining functions.

3.5.1 Conditions in optimization goal: HAC Based Algorithm

Recall that TagDM framework handles a set of social tagging behavior analysis tasks that optimizes one or more of the tagging action components (i.e., users, items,

tags), and adds to constraints components which are not part of the optimization goal. However, the type of optimization goal allowed in the TagDM framework is highly specific: it operates on one or more dimensions and the objective function maximizes a similarity or diversity score over the tagging action groups. However in reality, a user may be interested in adding few conditions over dimensions that are not easily expressed in terms of similarity or diversity. For example, a user may be interested in finding tagging action groups that maximize a *tag diversity* measure and satisfy *user and item similarity* constraints, such that the tagging groups have *frequent taggers*. Or, a user may be interested in finding groups that maximize a combination of *tag diversity and user diversity* measures and satisfy an *item similarity* constraint such that the groups have *median user age of 30*. In other words, the analysis task may require us to optimize a dimension along with the condition that the dimension satisfy some property over its distribution.

We observe that it is not easy to extend the previously described algorithms to handle such conditions. First, the hash functions used in LSH do not accommodate any technique for additional conditions. The algorithm SM-LSH-Fo was possible because the constraints F_1, F_2 and F_3 were based on similarity. Had they been based on other properties such as frequency, the folding technique would have been inadequate. While the FDP based algorithms can fold the constraints, it is tied to maximizing the average pair-wise distance between the facilities (i.e., the groups). Hence, we need a *general* algorithm that can seamlessly handle arbitrary conditions over dimensions in the optimization goal, for both similarity and diversity mining problems.

We propose an algorithm based on hierarchical agglomerative clustering (HAC) that has the following advantages:

- It can handle both similarity and diversity maximization problems, unlike LSH and FDP based which can handle similarity and diversity respectively.

- It is capable of similarity/diversity maximization along with some property of the distribution associated with the target optimization function, while LSH and FDP based methods cannot. Moreover, by changing how the clusters are merged, it can handle any objective function.

Hierarchical agglomerative clustering is a popular *bottom-up clustering* technique in which each data observation is treated as a singleton cluster at the outset, and then successively merged pair-wise until all clusters have been merged into a single cluster containing all data observations [29]. This unsupervised technique outputs an informative tree-like structure (known as *dendogram*) efficiently. It takes a symmetric matrix of distances between data observations as input, thereby helping us accommodate both tag similarity and tag diversity maximization problems. The merge operation in each iteration is determined greedily by looking up the distance matrix for a pair separated by smallest distance (in case of similarity maximization) and by largest distance (in case of diversity maximization.)

We specifically consider the average linkage HAC variant which merges pair of clusters with the minimum (or maximum) average distance from any member of one cluster to any member of the other cluster, since this function is equivalent to our Pair-Wise Aggregation Dual Mining Function F_{pa}. For the problem instantiations concerning tag similarity or diversity maximization, we need to compare the input set G of n tagging action groups using $F''_p(g_1, g_2, \texttt{tags}, m)$, $m \in \{\texttt{similarity}, \texttt{diversity}\}$ that operates on $T_{rep}(g_x)$, where $g_x \subseteq G$. The result set of groups with maximum tag similarity (or diversity) can be retrieved by determining the k vectors (from n d-dimensional tag signature vectors) with minimum (or maximum) average pair-wise distance between them. Our HAC based algorithm is often less efficient than LSH and FDP based algorithms as we see in Section 3.6.1, but are capable of handling a wide variety of complex problems which LSH and FDP based techniques cannot.

We adopt the HAC technique in the following way to handle our problem instances: given an input set G of n tagging action groups, each having a numeric tag signature vector $T_{rep}(g_x)$, where $g_x \subseteq G$, we employ average linkage HAC to merge clusters in each iteration. Once the dendogram optimizing tag similarity (or diversity) is generated, we post-process the dendogram in a top-down manner to retrieve the result set of tagging action groups G^{app} (We represent G^{opt} as G^{app} since HAC return approximate solutions) satisfying the hard constraints. Note that, the handling of the additional property, i.e., satisfiability of condition in the optimization goal is conducted during the merging of clusters in each iteration. In other words, we construct clusters such that members of the cluster, besides being similar (or dissimilar), satisfy additional properties and constraints. Thus, while traditional HAC algorithms are used for clustering a dataset into different partitions, our algorithm attempts to find one small group satisfying the constraints and maximizing the conditional criterion.

In our HAC based algorithm optimizing tag similarity (or diversity), the optimization function conditions, denoted by $F_C(G^{app}, \texttt{tags})$, and multiple hard constraints are *folded* into the merge operation. During each merge operation, we not only check for the pair with maximum (or minimum) average pair-wise cosine similarity score, but also check for the satisfiability of the condition along the dimension, as well as the hard constraints $1 \leq |G^{app}| \leq k$, $F'_P(\mathcal{G}_x, \mathcal{G}_y, \texttt{users}, \text{m}) \geq q$ and $F'_P(\mathcal{G}_x, \mathcal{G}_y, \texttt{items}, m) \geq \text{r}$, where $\mathcal{G}_x$, $\mathcal{G}_y$ are intermediate clusters.The algorithm terminates when $Support_G^{G^{app}} \geq p$, often without having to build the complete dendogram.

Algorithm 6 is the pseudo-code of the HAC based algorithm with average-link based agglomerative method. The naive implementation of HAC algorithm is $O(n^3)$ since it will exhaustively scan the $n \times n$ distance matrix S for determining the best merge in each of the $(n-1)$ iterations. However, if the function to compute the pairwise distance $F''_p(g_1, g_2, \texttt{tags}, m)$ is *cosine similarity*, then in conjunction with *heap*

Algorithm 6 HAC based (G, O, C, k, p): G^{app}

//Main Algorithm

S, I, P ←Compute $n \times n$ Distance Matrix(G)
$A \leftarrow []$; $i \leftarrow 1$; $G^{app} \leftarrow \{\}$
while $Support_G^{G^{app}} \geq \alpha$ **do**
  $\mathcal{G}_x \leftarrow argmax_{\{i:I[i]=1\}}$ P[i].m().distance //Using *cosine*
  $//m() \in \{\texttt{MAX}(), \texttt{MIN}()\}$ for $m \in \{\texttt{similarity}, \texttt{diversity}\}$
  $\mathcal{G}_y \leftarrow function($ $P[i].m().index$, $F_C(G^{app}, \texttt{tags})$, $|\{\mathcal{G}_x, \mathcal{G}_y\}| \leq k$, $F'_P(\{\mathcal{G}_x, \mathcal{G}_y\}, \texttt{users}, m) \geq q$, $F'_P(\{\mathcal{G}_x, \mathcal{G}_y\}, \texttt{items}, m) \geq r)$
  A.append $(\langle \mathcal{G}_x, \mathcal{G}_y \rangle)$
  S, I, $P \leftarrow$ Update Priority Queue(C, S, P, x, y)
  $i \leftarrow i + 1$
end while
$G^{app} \leftarrow$ Post-process(A, k, p, C)
return G^{app}

//Compute $n \times n$ Distance Matrix(G): S, I, P

for $x = 1$ to n **do**
  for $y = 1$ to n **do**
    $S[x][y].distance \leftarrow T_{rep}(g_x).T_{rep}(g_y)$
    $S[x][y].index \leftarrow i$
  end for
  $I[x] \leftarrow 1$
  $P[x] \leftarrow$ priority queue for $S[x]$ sorted on cosine similarity
  $P[x].delete(S[x][x])$
end for
return S, I, P

//Update Priority Queue(S, I, P, x, y): S, I, P

$I[y] \leftarrow 0$
$P[x] \leftarrow []$
for z **do**
  if $I[z] = 1 \wedge z \neq x$ **then**
    $P[z].delete(C[z][x])$
    $P[z].delete(C[z][y])$
    $S[z][x].distance \leftarrow F(\{\mathcal{G}^x, g_y\}, tags, m)$
    $P[z].insert(C[z][x])$ $//\mathcal{G}^x$ is intermediate cluster of g_z and g_x
    $S[x][z].distance \leftarrow F(\{\mathcal{G}^x, g_y\}, tags, m)$
    $P[x].insert(S[x][z])$
  end if
end for
return S, I, P

based priority queues which have $O(\log n)$ time for inserts and deletes, the algorithm will have a complexity of $O(n^2 \log n)$. In Algorithm 6, the rows of the $n \times n$ distance matrix S are sorted based on distance in the priority queues P. $P[x].MAX()$ or $P[x].MIN()$ returns the cluster in $P[x]$ which currently has the highest similarity (or dissimilarity) with $\mathcal{G}_x$, where $\mathcal{G}_x$ is the intermediate cluster formed in the x^{th} iteration. $\mathcal{G}_x$ is chosen as the representative of the cluster obtained by merging $\mathcal{G}_x$ and $\mathcal{G}_y$. After each merge operation, the priority queues maintained for each cluster are updated. The cluster similarity (or dissimilarity) computation takes constant time if vector sums $\sum_{g_x \in \mathcal{G}_x} T_{rep}(g_x)$ and $\sum_{g_y \in \mathcal{G}_y} T_{rep}(g_y)$ are available, where $\mathcal{G}_x$ and $\mathcal{G}_y$ are intermediate clusters selected for merging. This follows from Theorem 3.5.1 [30]:

Theorem 3.5.1. *The group average of the merged clusters for cosine similarity in average linkage hierarchical agglomerative clustering is given by :*

$$
\begin{aligned}
& F''_P(\{\mathcal{G}_x, \mathcal{G}_y\}, \texttt{tags}, m) \\
&= \frac{1}{(N)(N-1)} \sum_{g_x \in \mathcal{G}_x} \sum_{g_y \in \mathcal{G}_y; g_x \neq g_y} T_{rep}(g_x).T_{rep}(g_y) \\
&= \frac{1}{(N)(N-1)} [(\sum_{g_x \in \mathcal{G}_x} T_{rep}(g_x) + \sum_{g_y \in \mathcal{G}_x} T_{rep}(g_y))^2 - (N)]
\end{aligned}
$$

where $N = n_i + n_j$, $\mathcal{G}_x$ and $\mathcal{G}_y$ are intermediate clusters being selected for merging, $T_{rep}(g_x)$ and $T_{rep}(g_y)$ are length normalized tag signature vectors of corresponding to tagging groups g_x and g_y respectively, . denotes the dot product, n_x and n_y are the number of groups in $\mathcal{G}_x$ and $\mathcal{G}_y$ respectively. Therefore, the distributivity of the dot product with respect to vector addition aids constant time cluster merge condition computation. Note that, when the TagDM problem instance optimizes similarity in tagging behavior, the algorithm merges two most similar clusters having maximum average pair-wise cosine similarity score; when the problem instance optimizes diversity in tagging behavior, it merges two most dissimilar clusters having minimum average pair-wise cosine similarity score.

Example 3.5.1. *Let us re-examine the problem of identifying the most similar set of tagging action groups, as in Example 3.4.1. We consider the same dataset and input set G of tagging action groups. Let us re-define the optimization goal as: identify the result set G^{app} of $k = 3$ groups having maximum similarity in tagging behavior such that majority (i.e., more than 50%) of the users in the tagging groups are frequent taggers (i.e., have tagged at least 25 items, say). Using cosine similarity as the distance measure, we first build the distance matrix S which is shown in Table 3.3.*

Table 3.3. Distance matrix S^G

	g_1	g_2	g_3	g_4	g_5
g_1	1.00	0.46	0.45	0.92	0.90
g_2	0.46	1.00	0.28	0.66	0.51
g_3	0.45	0.28	1.00	0.17	0.78
g_4	0.92	0.66	0.17	1.00	0.74
g_5	0.90	0.51	0.78	0.74	1.00

The pair of tagging action groups with highest similarity between them are g_1 and g_4. If the multiple hard constraints in the problem (such as constraints along user and item dimensions, etc.) as well as the additional condition in the optimization goal is satisfied, we merge these two groups to a single cluster $\mathcal{G}_1$. If the constraints and conditions are not taken care of, we proceed with the second most similar pair of tagging action groups, namely g_1 and g_5. Assume, we merge g_1 and g_4 to $\mathcal{G}_1$. Next, we update the similarity between the remaining groups g_2, g_3 and g_5 with $\mathcal{G}_1$ using the average link similarity. In other words, similarity between $\{g_1, g_4\}$ and say g_2 is computed as the average of similarities between pairs (g_1, g_2) and (g_4, g_2). Using the updated matrix, we observe that the cluster $\mathcal{G}_1 = \{g_1, g_4\}$ and tagging action group g_5 have the highest similarity. We check for the satisfiability of the conditions and constraints, and find out that $\{g_1, g_4, g_5\}$ do not satisfy the frequent tagger condition in

the optimization goal. That is, less than 50% of users belonging to the set $\{g_1, g_4, g_5\}$ *are frequent taggers. Hence, we do not proceed with this merge operation and move to the second best merge option that satisfies the frequent tagger condition and all the hard constraints - we merge the cluster* $\mathcal{G}_1 = \{g_1, g_4\}$ *and tagging action group* g_2 *(say). The merged cluster* $\mathcal{G}_2$ *has* $k = 3$ *tagging action groups, and also satisfies the support condition (say). Hence, we terminate our algorithm and return the set* $G^{app} = \{g_1, g_2, g_4\}$ *as the solution. The result* G^{app} *can be interpreted as:* female users in Texas *are frequent taggers and have similar tagging behavior for the dataset under consideration.* □

Complexity Analysis: The complexity of the efficient implementation of the HAC based algorithm is $O(n^2 \log n)$. A priority queue requires $O(\log n)$ time for inserts and deletes, resulting in $O(n \log n)$ time for n priority queues as opposed to $O(n^2)$ distance matrix update time in naive implementation.

3.5.2 General dual mining functions: HC Based Algorithm

One factor responsible for bringing in variations in TagDM problem instances is which measure (similarity or diversity) a user is interested in applying to which tagging components (users, items, or tags). All the algorithmic solutions proposed consider pair-wise aggregation dual mining function in Function 3.2.4. However, a user may be interested in more general mining measures which cannot be computed over pairs of tagging action groups, as discussed in Section 3.2.4. Transitioning from pair-wise to general dual mining functions allows one to characterize holistic properties that occur at the global level in the optimal set of groups instead of simply aggregating local (and pair-wise) properties. Every dual mining function that a user may want to explore can be placed in the spectrum between local and global properties. While holistic properties increases the expressive power of the TagDM frame-

work, they also necessitate the development of generic algorithms that can handle arbitrary dual mining functions. Without additional knowledge of dual mining function properties (such as monotonicity, sub-modularity or even metric property), it is often difficult to describe deterministic algorithms that have any meaningful approximation guarantees. The Exact algorithm needs to check an exponential number of combinations and is simply not scalable.

We extend ideas from our earlier work [1] and propose a hill-climbing (HC) based technique that is capable of handling all the problem instances we have discussed so far, with general dual mining defined in Definition 3.2.7, instead of pair-wise aggregation mining function with limited scope defined in Definition 3.2.4. We build a lattice of all possible tagging action groups (which are structurally describable by user and/or item attributes), where the nodes correspond to user describable and item describable groups and the edges correspond to parent/child relationships. Note that the number of nodes in the lattice of TagDM framework is usually higher than that in the lattice of MRI in [1], since the latter has either user-describable lattice (for item-based query) or item-describable lattice (for user-based query). Also, the scalar numeric rating values have been replaced with numeric vectors in this framework. Our HC based algorithm is advantageous over all the previously discussed algorithms in the following ways:

- It can handle general dual mining measures for similarity and diversity mining which LSH, FDP and HAC based algorithms cannot. Therefore, this algorithm is capable of solving a wide range of analysis tasks that none of the other three algorithms can.
- It can handle both similarity and diversity maximization problems like HAC based, unlike LSH and FDP based which can handle similarity and diversity respectively.

- It is also capable of similarity/diversity maximization along with some condition in the optimization goal like HAC based, while LSH and FDP based cannot.

A straightforward adoption of the *random restart hill climbing* [10] technique involves the following steps: we first randomly select a set of k tag signature vectors (corresponding to k tagging action groups) as the starting seed; the process then continues by replacing one group in the current set with one of its neighbors[3] not in the set as long as the substitution maximizes or minimizes the general dual mining function; the algorithm stops when no improvements can be made indicating a *local optima* has been reached. The process is repeated with multiple diverse seed sets to increase the probability of finding the *global optima* that satisfies the conditions and constraints. However, this simple application of hill climbing does not suffice because of the inclusion of constraints and/or conditions in the tasks. For any given set of groups randomly chosen as the starting set, the probability of it satisfying all the constraints is fairly small, thereby necessitating a large number of restarts.

Therefore, we consider the *Randomized Hill Exploration Algorithm* [1](RHE) which first initializes a randomly selected set of k vectors as the starting set. However, instead of immediately starting to improve the target function, it *explores* the hill to detect sets of k tagging vectors in the neighborhood that satisfy the conditions and constraints. This new set of k group tag signature vectors is then adopted as the starting point for the tag similarity or diversity maximization, with the added condition that an improvement is valid only when the constraints and conditions hold.

The details of the algorithm are shown in Algorithm 8. Intuitively, we begin with the construction of the tagging vector lattice L_T. The algorithm starts by picking k random groups in the lattice to form the initial seed set G_C. For each group g_i in G_C, we swap g_i with each of its neighbors g_j in the lattice, while the

[3]Two tagging action groups are neighbors if they are directly connected in the lattice.

other groups in G_C remain fixed, to generate a new combination. The exploration phase continues till it finds the *best* set of k tagging action groups that satisfies all the constraints and conditions. The resulting set then acts as the initial condition for the second phase of the optimization to maximize (or minimize) the general dual mining function F measuring tag similarity (or diversity). The configuration that satisfies the constraints and conditions, and incurs the optimum value of F is the *best* tagging behavior explanation for the query.

Example 3.5.2. *Once again, we re-examine the problem of identifying the most similar set of tagging action groups, as in Example 3.4.1 and 3.5.1. We consider the same dataset and input set G of tagging action groups. The objective is to identify the result set G^{app} of $k = 3$ groups having maximum similarity in tagging behavior using the HC-based algorithm. Recall that, here we consider a general dual mining function, defined in Definition 3.2.7, instead of the pair-wise aggregation cosine similarity measure used in Examples 3.4.1, 3.4.2, and 3.5.1. The tagging action groups are concisely represented as a lattice L_T, as shown in Figure 3.3. We can see that there are 3 tagging action groups that can be described using a single attribute and 2 that are described using two user attributes.*

When the algorithm starts, it randomly picks 3 groups from L_T. Suppose, it picks the groups containing all female users and the users from Texas and California, i.e., $G_C = \{g_1, g_2, g_3\}$. Our algorithm explores the neighboring groups for each of the candidates in G_C. One of the candidates is g_4, the set of female users from Texas. If $G'_C = \{g_1, g_3, g_4\}$ improves the aggregate score (measured by general dual mining function) as compared to $G_C = \{g_1, g_2, g_3\}$ while satisfying the multiple constraints and conditions, we update G_C to G'_C. We explore the remaining neighbors (i.e., g_5) and update G_C in a similar fashion. Suppose, $G_C = \{g_1, g_3, g_5\}$ has the highest aggregate score. When there are no more neighbors that can increase the aggregate

Algorithm 7 HC based (G, O, C, k, p): G^{app}

- Build lattice L_T of all tagging action groups, as in [1].

//Main Algorithm

1: $G_C \leftarrow$ randomly select k groups/nodes from L_T
2: **if** $Support_G^{G_C} \geq \alpha$, $F_C(G_C, \texttt{tags})$, $F'_p(g_1, g_2, \texttt{users}, m) \geq q$ and $F'_p(g_1, g_2, \texttt{items}, m) \geq r$ **then**
3: $C \leftarrow$ satisfy-constraints(G_C, L_T)
4: **end if**
5: $C \leftarrow$ optimize-dual-mining-function(C, L_T)
6: $G^{app} \leftarrow$ best C so far
7: **return** G^{app}

// satisfy-constraints(G_C, L_T): C

1: **while** true **do**
2: val $\leftarrow \texttt{coverage}(G_C, L_T)$
3: **for** each group g_i in G_C, each neighbor g_j of g_i **do**
4: $C' \leftarrow G_C - g_i + g_j$
5: $val' \leftarrow Support_G^{C'} \geq \alpha$
6: **if** $val' \geq \alpha$, $F_C(C', \texttt{tags})$, $F'_p(g_1, g_2, \texttt{users}, m) \geq q$ and $F'_p(g_1, g_2, \texttt{items}, m) \geq r$ **then**
7: $C \leftarrow C'$
8: **return** C'
9: **end if**
10: **end for**
11: **end while**

// optimize-dual-mining-function(C, L_T): C

1: **while** true **do**
2: val $\leftarrow F(C, L_T)$ // F is general dual mining function
3: $\mathcal{C} = \emptyset$
4: **for** each group g_i in C, each neighbor g_j of g_i **do**
5: $C' \leftarrow C - g_i + g_j$
6: **if** $Support_G^{C'} \geq \alpha$, $F_C(C', \texttt{tags})$, $F'_p(g_1, g_2, \texttt{users}, m) \geq q$ and $F'_p(g_1, g_2, \texttt{items}, m) \geq r$ **then**
7: add $(C', F(C, L_T))$ to $\mathcal{C}$
8: **end if**
9: **end for**
10: let $(C'_m, val'_m) \in \mathcal{C}$ be the pair with minimum F
11: **if** $val'_m \geq val$ **then**
12: **return** C // we have found the local minima
13: **end if**
14: $C \leftarrow C'_m$
15: **end while**

score, the algorithm terminates since it has reached the local maxima. The result $G^{app} = \{g_1, g_3, g_5\}$ *is returned, which can be interpreted as:* female users from California *have similar tagging behavior.*

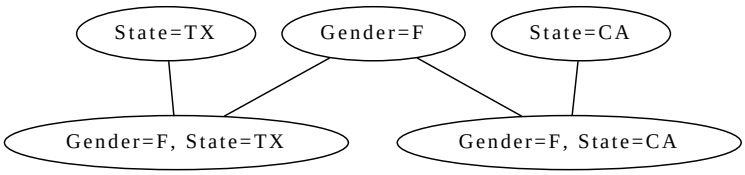


Figure 3.3. Lattice representation of tagging action groups for example.

Complexity Analysis: The worst case complexity of the hill climbing based algorithm is exponential in the size of the search space. The space complexity of the search space, i.e., the number of groups/nodes in the lattice, is a function of the number of attribute values users and items take.

Discussion: Table 3.4 broadly summarizes our algorithmic contributions for solving the TagDM problem instances in the general TagDM framework and those in the extensions of the TagDM framework.

Table 3.4. Summary of our Algorithmic Solutions. Column $O.m$ lists the optimization O mining criterion ($m \in \{\texttt{similarity}, \texttt{diversity}\}$), column C_O lists if Algorithm can handle condition(s) in optimization goal, column $O.F$ lists the mining function (F $\in \{F_p(pairwise), F(general)\}$) that Algorithm can handle, and the final column discusses how Algorithm handles hard constraints.

Algorithm	O.m	C_o	O.F	Additional Techniques
LSH based	similarity	no	F_p	fold similarity and filter diversity constraints
FDP based	diversity	no	F_p	fold diversity and filter similarity constraints
HAC based	similarity, diversity	yes	F_p	fold both similarity and diversity constraints during merge
HC based	similarity, diversity	yes	F_p, F	fold both similarity and diversity constraints during exploration

3.6 Experiments

We conduct a comprehensive set of experiments for quantitative (Section 3.6.1) and qualitative (Section 3.6.2) analysis of our proposed algorithms for TagDM problem instances. Our quantitative performance indicators are efficiency of the algorithms and analysis quality of the results produced. The efficiency of our algorithms is measured by the overall response time, whereas the result quality is measured by the average pair-wise distance between the k tagging action group vectors returned by our algorithms. In order to demonstrate that the tagging behavior analysis generated by our approaches are interesting to end users, we conduct a user study through Amazon Mechanical Turk. We also present interesting case studies to show how results generated by our algorithms for TagDM problem instances varies.

Real Movie Dataset: We require dataset(s) that contains information about a set of users tagging a set of items, where attributes associated with users and attributes associated with items are known. We use the MovieLens[4] 1M and 10M ratings dataset for our evaluation purposes. The MovieLens 1M dataset consists of 1 million ratings from 6000 users on 4000 movies while the 10M version has 10 million ratings and 100,000 tagging actions applied to 10,000 movies by 72,000 users. The titles of movies in MovieLens are matched with those in the IMDB[5] to obtain movie attributes.

User Attributes: The 1M dataset has well-defined user attributes but no tagging information, whereas the 10M dataset has tagging information but no user attributes. Therefore, for each user in the 1M dataset with a complete set of attributes, we build her rating vector and compare it to the rating vectors (if available) of all 72,000 users in the 10M dataset. For every user in 10M dataset, we find the user in 1M dataset such that the cosine similarity of their movie rating vector is the highest (i.e., user rating

[4]http://www.grouplens.org/datasets/movielens/
[5]http://www.imdb.com/interfaces

behaviors are most identical). The attributes of user in 10M dataset are obtained from the closest user in 1M dataset. In this way, we build a dataset consisting of 33,322 tagging and rating actions applied to 6,258 movies by 2,320 users. The tag vocabulary size is 64,663. The user attributes are gender, age, occupation and zip-code. The attribute *gender* takes 2 distinct values: male or female. The attribute *age* is chosen from one of the eight age-ranges: under 18, 18-24, ..., 56+. There are 21 different *occupations* listed by MovieLens such as student, artist, doctor, lawyer, etc. Finally, we convert zipcodes to states in the USA (or foreign, if not in USA) by using the USPS zip code lookup[6]. This produces the user attribute *location*, which takes 52 distinct values.

Movie Attributes: Movie attributes are genre, actor and director. There are 19 movie *genres* such as action, animation, comedy, drama, etc. The pool of actor values and director values, corresponding to movies which have been rated by at least one user in the MovieLens dataset, is huge. We pick only those actors and directors who belong to at least one movie that has received greater than 5 tagging actions. In our experiments, the number of distinct *actor* attribute values is 697 while that of distinct *director* is 210.

Mining Functions: The set of tagging action groups is built by performing a cartesian product of user attribute values with item attribute values. An example tagging action group is {gender=male, age=under 18, occupation=student, location=new york, genre=action, actor=tom hanks, director=steven spielberg}. The total number of possible tagging action groups is more than 40 billion, while the number of tagging action groups containing at least one tuple is over 300K. For our experiments, we consider 4535 groups that contain at least 5 tagging action tuples. The user and item similarity (or diversity) is measured by determining the structural distance between user and

[6]http://www.grouplens.org/datasets/movielens/http://zip4.usps.com

item descriptions of groups respectively. For topic discovery, we apply LDA [13] as discussed in Section 3.2.2. We generate a set of 25 global topic categories for the entire dataset, i.e., $d = 25$. For each tagging action group, we perform LDA inference on its tag set to determine its topic distribution and then generate its tag signature vector of length 25. Finally, we use cosine similarity (or, some general mining measure) for computing pair-wise similarity between tag signature vectors.

System Configuration: Our prototype system is implemented in Python. All experiments were conducted on an Ubuntu 11.10 machine with 4 GB RAM, AMD Phenom II N930 Quad-Core Processor.

3.6.1 Quantitative Evaluation

First, we compare the execution time and result quality of all 6 TagDM problem instantiations in Table 3.1 for the entire dataset (consisting of 33K tuples and 4K tagging action groups) using Exact, SM-LSH-Fi, SM-LSH-Fo, DV-FDP-Fi and DV-FDP-Fo algorithms. We use the name Exact for the brute-force approach on both tag similarity and diversity maximization instances. We set the number of tagging action groups to be returned at $k = 3$, since the Exact algorithm is not scalable for larger k. Figure 3.4 and 3.5 compare the execution time and quality respectively of Exact and LSH based algorithms for Problems 1, 2 and 3 (Tag Similarity). Figure 3.6 and 3.7 compare the execution time and quality respectively of Exact and FDP based algorithms for Problems 4, 5 and 6 (Tag Diversity). The execution time is the time taken to retrieve the result set. The quality of the result set is measured by computing the average pair-wise cosine similarity between the tag signature vectors of the $k = 3$ tagging action groups returned. The group support is set at $p = 350$ (i.e., 1%); the user attribute similarity (or, diversity) constraint as well as the item attribute similarity (or, diversity) constraint is set to $q = 50\%$, $r = 50\%$ respectively. For LSH based algorithms, the number of hash tables is $l = 1$ while the initial value of d' is 10.

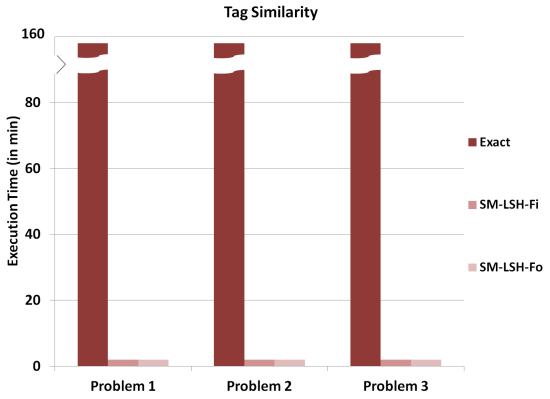


Figure 3.4. Execution Time:Problems 1, 2, 3 in Table 3.1.

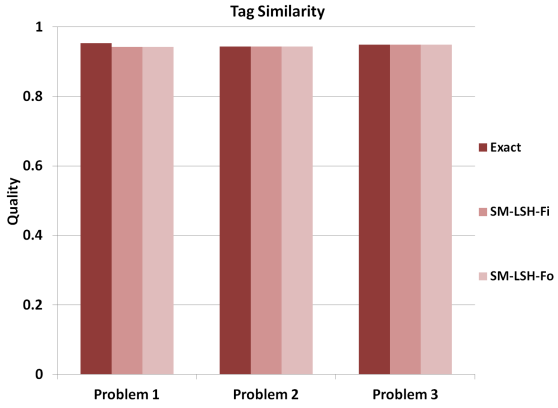


Figure 3.5. Quality:Problems 1, 2, 3 in Table 3.1.

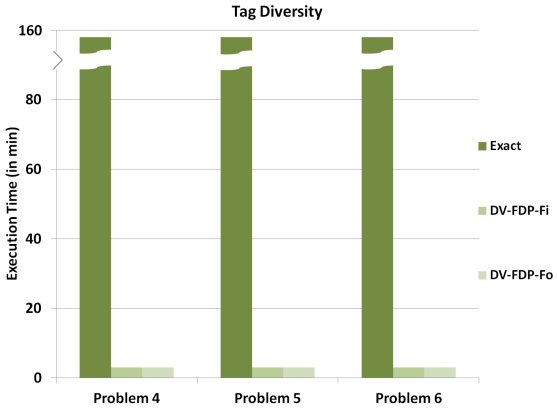


Figure 3.6. Execution Time:Problems 4, 5, 6 in Table 3.1.

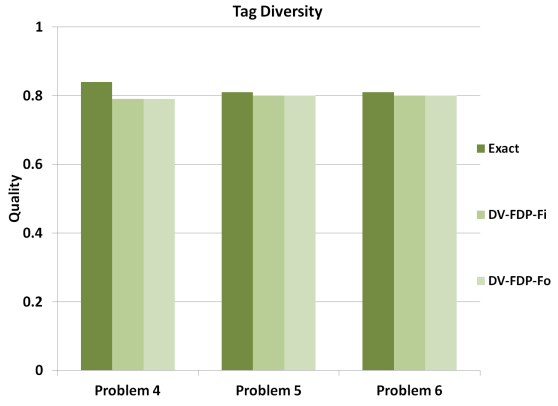


Figure 3.7. Quality:Problems 4, 5, 6 in Table 3.1.

We observe that the execution time of our LSH based for similarity and FDP based algorithm for tag diversity problem instances are much faster than the Exact equivalent. In Figure 3.4, the execution times of SM-LSH-Fi and SM-LSH-Fo for Problems 1, 2 and 3 are comparable to each other and is around 2 minutes. In Figure 3.6, the execution times of DV-FDP-Fi and DV-FDP-Fo for Problems 4, 5 and 6 are slightly more than 3 minutes. Despite significant reduction in time, our algorithms do not compromise much in terms of quality, as evident from Figure 3.5 and 3.7.

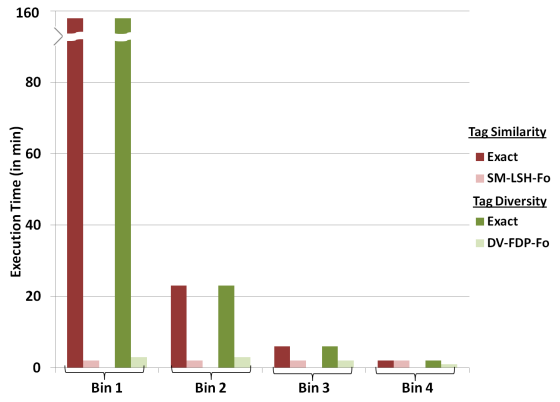


Figure 3.8. Execution Time:Varying Tagging Tuples.

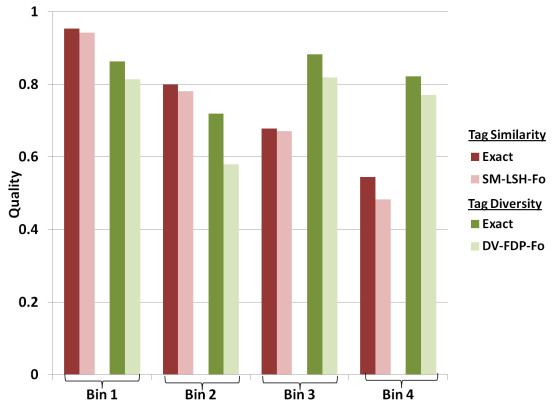


Figure 3.9. Quality:Varying Tagging Tuples.

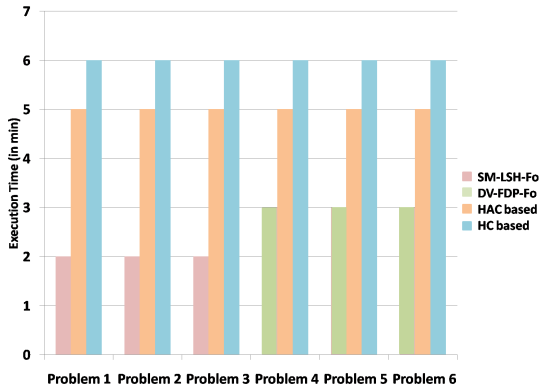


Figure 3.10. Execution Time: Different Algorithms.

Figure 3.11. Quality: Different Algorithms.

The number of input tagging action tuples available for tagging behavior analysis is dependent on the *query* under consideration. For the entire dataset, there are 33K such tuples. However, if we want to perform tagging behavior analysis of all movies tagged by {`gender`= `male`} or {`location`= `ca`}, the number of available tuples is 26,229 and 6,256 respectively. Or, if want to perform tagging behavior analysis of all users who have tagged movies having {`genre`= `drama`}, the number of tuples is 17,368. Needless to say, the number of tagging action tuples can have a significant impact on the performance of the algorithms since it affects the number of non-empty

Figure 3.12. Execution Time: Exact vs HAC, Exact vs HC.

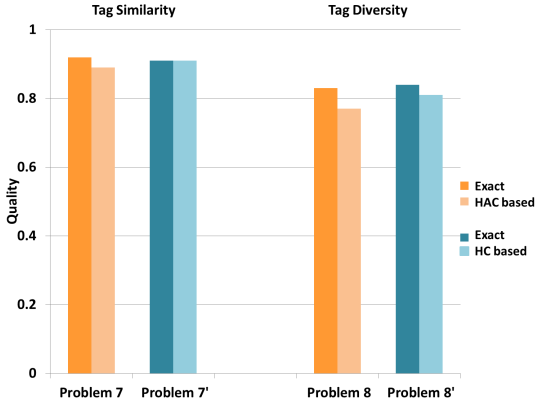


Figure 3.13. Quality: Exact vs HAC, Exact vs HC.

tagging action groups on which our algorithms operate. As a result, we build 4 bins having 30K, 20K, 10K and 5K tagging action tuples respectively (assume, each bin is a result of some query on the entire dataset) and compare our algorithm performances for one of the tag similarity maximization problems and one of the tag diversity maximization problems, say Problem 1 and Problem 6 from Table 3.1 respectively. Both Problems 1 and 6 have user and item dimension constraints set to similarity. Figures 3.8 and 3.9 compare the execution time and quality respectively of the Exact algorithm with our smart algorithms: SM-LSH-Fo for Problem 1 and DV-FDP-Fo for Problem 6. The group support is set at $p = 350$ (i.e., 1%); user similarity (or, diversity) constraint and item similarity (or, diversity) constraint are set to $q = 50\%$, $r = 50\%$ respectively, and $k = 3$. For each bin along the X axis, the first two vertical bars stand for Problem 1 (tag similarity) and the last two stand for Problem 6 (tag diversity). As expected, the difference in execution time between our algorithms and the Exact is small for bins with lesser number of tagging tuples for both tag similarity and diversity. However, our algorithms return results much faster than Exact for bins with larger number of tagging tuples. The quality scores continue to be comparable to the optimal answer, as shown in Figures 3.9.

Next, we analyze the performance behavior of the HAC based and HC based algorithms introduced in Section 3.5. Recall that, both these techniques were developed to solve a wide variety of complex mining problems which LSH based and FDP based cannot handle. Before showcasing the power of HAC and HC based algorithms, we first compare all the four sets of algorithms - LSH based, FDP based, HAC based, and HC based, under the same settings as above for the 6 TagDM problem instantiations in Table 3.1. Then, we investigate the potential of HAC based and HC based algorithms to handle analysis tasks which LSH and FDP based cannot.

Figure 3.10 and Figure 3.11 compare the execution time and quality respectively of the four algorithms for the 6 TagDM problem instantiations in Table 3.1 - the first three dealing with Tag Similarity and the last three dealing with Tag Diversity. We employ LSH based technique SM-LSH-Fo for Problems 1, 2, 3 and FDP based technique SM-FDP-Fo for Problems 4, 5, 6. HAC and HC based techniques are capable of handling both similarity and diversity mining problems. In order to compare the algorithms under the same settings, we consider cosine measure as the dual mining function for HC based algorithm (though it can handle general measures), since the LSH based, FDP based and HAC based methods can only handle pair-wise aggregation mining function. Figure 3.10 reveals that the time taken by the different algorithms are comparable to each other. For Problems 1, 2, 3, SM-LSH-Fo takes 2 seconds while HAC based and HC based algorithms take around 5 and 6 seconds respectively. For Problems 4, 5, 6, DV-FDP-Fo takes 3 seconds while HAC based and HC based algorithms take around 5 and 6 seconds respectively. From Figure 3.11, we see that the quality of results returned by the different algorithms for the 6 problems are very close to each other. Though the time taken by HAC and HC algorithms are slightly higher than LSH and FDP based techniques for the same set of problems, HAC and HC algorithms returns good quality results and are capable of

handling additional complex mining objectives which LSH and FDP based methods cannot handle. Therefore, we recommend LSH and FDP based algorithms for the simple tasks of TagDM framework, and HAC and HC based algorithms for the more advanced problems belonging to the extended TagDM framework.

In order to evaluate the performance behavior of HAC based algorithm and HC based algorithm for complex mining tasks, we compare the execution time and result quality of two complex TagDM problems - one for similarity, one for diversity - using Exact and HAC based, and Exact and HC based. We use the name Exact for the brute-force approach on both similarity and diversity maximization instances. We set the number of tagging action groups to be returned at k = 3, since the Exact algorithm is not scalable for larger k; the other settings (p, q, etc.) remain same too. The complex mining tasks that we investigate are as follows:

PROBLEM 7: We extend Problem 3.2.2 to handle additional condition in the optimization goal. The objective is to find similar user sub-populations who agree most on their tagging behavior for a diverse set of items such that the selected user groups contain at least one tagger from the list of top 50 frequent taggers in the data. When the mining measure is general instead of pair-wise, we refer to it as PROBLEM 7'.

PROBLEM 8: We extend Problem 3.2.5 to handle additional condition in the optimization goal. The objective is to find diverse user sub-populations who disagree most on their tagging for a similar set of items such that the selected user groups contain at least one tagger from the list of top 50 frequent taggers in the data. When the mining measure is general instead of pair-wise, we refer to it as PROBLEM 8'.

Figures 3.12 and 3.13 compare the execution time and quality respectively of Exact and HAC based, as well as Exact and HC based algorithms for Problems 7, 7' (Tag Similarity) and Problems 8, 8' (Tag Diversity). We observe that the execution time of our HAC based and HC based techniques are much faster than the Exact

equivalent. Despite significant reduction in execution time, our algorithms do not compromise much in terms of quality, as evident from Figure 3.13.

3.6.2 Qualitative Evaluation

We now validate how social tagging behavior analysis can help users spot interesting patterns and draw conclusions about the desirability of an item, by presenting several anecdotal results on real data. We also compare the utility and popularity of the 6 novel mining problems in Table 3.1 in an extensive user study conducted on Amazon Mechanical Turk (AMT)[7].

3.6.2.1 Case Study

First, we present a set of anecdotal results returned by our algorithms for the same query for different TagDM problem instances. Specifically, we focus on Problems 2, 3, and 4 in Table 3.1 and observe the results returned by our algorithms:

$\Diamond$ *Analyze tagging behavior of* $\langle$occupation= student$\rangle$ *users for movies.*

- Problem 2 finds similar user sub-populations who agree most on their tagging behavior for a diverse set of items. We retrieve that male students use similar tags `dystopia`, `sci-fi`, `post-apocalyptic`, etc. for diverse movies "Serenity" and "The Matrix" - the former is a space western movie while the latter is a science fiction action film.
- Problem 3 finds diverse user sub-populations who agree most on their tagging behavior for a similar set of items. We identify that male and female students use similar tags `classic`, `hope`, `friendship`, `based on a book`, etc. for the movie "The Shawshank Redemption".
- Problem 4 finds diverse user sub-populations who disagree most on their tagging behavior for a similar set of items. Our algorithm returns that male and female students use diverse tags for movies directed by "Quentin Tarantino" -

[7]https://www.mturk.com

male reviewers use tags `crime, cult film, dark comedy,` etc. while female reviewers use `insane`, `visceral`, `ultra-violence`, etc.

Second, we show how TagDM results change due to addition of conditions in optimization goal (Section 3.5.1) and consideration of general dual mining function (Section 3.5.2), other settings remaining the same. Let us consider the problem of finding diverse user sub-populations who agree most on their tagging behavior for a similar set of items, and then show how the results change due to inclusion of a condition in the optimization goal. We observe the result returned for the query:
$\Diamond$ *Analyze tagging behavior of* for $\langle$genre= romance$\rangle$ movies.

- Young female reviewers and middle-aged female reviewers use similar tags like `classic`, `sweet`, `love`, etc. for romance movies.

If the task is to find out diverse user sub-populations who agree most on their tagging for {`genre` = `romance`} movies such that majority of the users in the result set have used the tag `love` at least once, the result is:

- Young female reviewers and middle-aged female reviewers use similar tags `Oscar`, `Meg Ryan`, `Nora Ephron`, `Julia Roberts`, etc. for romance movies.

Thus, we can infer that young and middle-aged female reviewers agree in their feedback towards romance movies; when these reviewers use the tag `love`, their agreement is specifically expressed for movies starring Meg Ryan, Julia Roberts, etc.

Let us consider the problem of finding diverse user sub-populations who agree most on their tagging behavior for a similar set of items, where similarity is either measured as pair-wise aggregation (cosine) or is computed using a general mining function. We compare the results returned for the query:
$\Diamond$ *Analyze tagging behavior for* $\langle textsfdirector=$ steven spielberg$\rangle$ movies.

- For pair-wise cosine similarity measure: Male and female use similar tags `Oscar`, `true story`, `violence`, etc. for war movies "Saving Private Ryan" and "Schindler's List" directed by Steven Spielberg.

- For general similarity mining measure (say, tagging behavior of result sub-population is closest to tagging behavior of global population): Old male and young male use similar tags `Oscar`, `true story`, `based on a book`, etc. for action movies and fantasy movies directed by Steven Spielberg.

Thus, we can infer that the different mining measures yield different analysis of tagging behavior. Due to the nature of the general mining measure, it returns groups that covers a broader spectrum of movies belonging to the query than that covered by pair-wise cosine similarity aggregation.

Third, we show how results returned by our algorithms for TagDM problem instances varies with increase in k, i.e., the number of user sub-populations being returned. We consider the problem of finding similar user groups who have similar tagging behavior for diverse set of items for the query:

$\Diamond$ *Analyze tagging behavior of* $\langle$gender= male$\rangle$ *users for movies.*

- For $k = 2$: Young male students use similar tags `sci-fi`, `dystopia`, `post apocalyptic`, etc. for diverse movies "Serenity" and "The Matrix". The groups returned are:

  $g_1 = \{\langle$gender, male$\rangle, \langle$age, young$\rangle, \langle$occupation, student$\rangle, \langle$title, Serenity$\rangle, \langle$genre, space western$\rangle, (\texttt{sci}-\texttt{fi}, \texttt{dystopia}, \texttt{apocalyptic})\}$

  $g_2 = \{\langle$gender, male$\rangle, \langle$age, young$\rangle, \langle$occupation, student$\rangle, \langle$title, The Matrix$\rangle, \langle$genre, action$\rangle, (\texttt{sci}-\texttt{fi}, \texttt{dystopia}, \texttt{apocalyptic}, \texttt{martialarts})\}$

- For $k = 4$: Young male students use similar tags `sci-fi`, `dystopia`, `cult`, `philosophical`, etc. for diverse movies "Serenity", "The Matrix", "Blade Runner", and movies directed by "Peter Jackson". Movies directed by Peter Jackson include "The Lord of the Rings (film series)".
- For $k = 6$: Young male students use similar tags `sci-fi`, `dystopia`, `based on a book`, `fantasy`, etc. for diverse movies "Serenity", "The Matrix", "Blade

Runner", movies directed by "Peter Jackson", movies directed by "Joss Whedon", and "Eternal Sunshine of the Spotless Mind". Movies directed by Joss Whedon include "Serenity" and "The Avengers".

Thus, we can infer that young male student reviewers agree in general in their feedback towards diverse set of movies - a space western movie, a science fiction action film, a noir detective fiction, a romantic dramedy science fiction film, etc. We can also infer that diversity of items returned as part of the result broadens with increase in the value of k.

TagDM analysis tasks can also throw in surprising results, as we see for the query:

$\Diamond$ *Analyze tagging behavior of* $\langle$gender= male, location= california$\rangle$ *users for movies.*

- Old male and young male living in California use similar tags for "Lord of the Rings" film trilogy of fantasy genre. However, they differ in their tagging towards "Star Wars" movies having similar genre. This is because, the genre of the latter series borders between fantasy and science fiction. Surprisingly, old male likes it while young male does not.

3.6.2.2 User Study

We conduct a user study through Amazon Mechanical Turk to elicit user responses towards the different TagDM problem instances we have focused on in the chapter, and investigate if the problems are interesting. We generate analysis corresponding to all 6 problem instantiations for the following randomly selected queries:

- $\diamond$ *Analyze tagging behavior of* $\langle$gender= male$\rangle$ *users for movies.*
- $\diamond$ *Analyze tagging behavior of* $\langle$occupation= student$\rangle$ *users for movies.*
- $\diamond$ *Analyze user tagging behavior for* $\langle$genre= drama$\rangle$ *movies.*

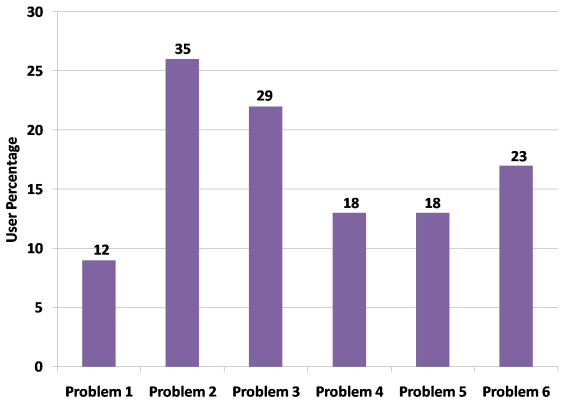


Figure 3.14. User Study.

We have 50 independent single-user tasks. Each task is conducted in two phases: User Knowledge Phase and User Judgment Phase. During the first phase, we estimate the user's familiarity about movies in the task using a survey, besides her demographics. In the second phase, we ask users to select the most preferred analysis, out of the 6 presented to them, for each query. Since there are 3 queries and 50 single-user tasks, we obtain a total of $3 \times 50 = 150$ responses. The first phase eliminates 5 of the 50 single-users. Therefore, our user study is based on a total of $3 \times 35 = 135$ responses, which are aggregated to provide an overall comparison between all problem instances in Figure 3.14. The height of the vertical bars represent the percentage of users, preferring a problem instance. We also place the numerical number of user responses against each of the vertical bars. It is evident that users prefer TagDM Problems 2 (*find similar user sub-populations who agree most on their tagging behavior for a diverse set of items*), 3 (*find diverse user sub-populations who agree most on their tagging behavior for a similar set of items*) and 6 (*find similar user sub-populations who disagree most on their tagging behavior for a similar set of items*), having diversity as the measure for exactly one of the tagging component: item, user and tag respectively.

3.7 Demonstration

We propose a system VISTAR for interactive mining and exploration, as well as geo-visualization of social tagging and rating behavior. We choose to focus on short user feedback (*ratings* and *tags*) since its aggregated view over numerous user-item interactions is informative, unlike long textual reviews that require semantic analysis. Our system carefully aggregates the rapidly growing data in collaborative content sites and allows a user to analyze how ratings and tags are assigned by certain users to certain items. In contrast to MAPRAT in Section 2.6, our system performs *simultaneous mining* on both rating and tags. Our proposed solution extends the dual mining framework in this chapter to additionally include rating as the fourth dimension. By applying the notion of similarity and diversity on the different dimensions namely users, items, ratings and tags, our system can identify multiple interesting patterns that cannot be extracted by traditional state-of-art analysis.

3.7.1 Architecture

There are two major components in VISTAR: Dual Mining and Visualization. Dual Mining: This module accepts a query q (consisting of user attributes u or item attributes i or a combination of both) from the front-end and collects all the corresponding feedback tuples R_q. The set of user attribute based groups and item attribute based groups that has at least one feedback tuple in R_q are then constructed. Depending on the user requirements, we formulate it as a constrained optimization problem as defined by the TagDM framework. Based on whether the optimization target is similarity or diversity, we invoke variants of LSH or FDP based algorithms discussed in this chapter. The output of this component is a set of *interesting* groups that maximizes the user criterion. Using a combination of aggressive data pre-processing, result pre-computation and caching techniques, the latency of our system is minimized.

Visualization: This module takes a set of groups returned by the dual mining module as input and visualizes the results, so that any user of VISTAR can quickly examine the trends for making decisions. We identify that location of a reviewer is a natural attribute over which social feedback behavior can be effectively visualized. Such geo-visualization allows quick cognition of the behaviors, highlights geographical trends in user feedback patterns (if any) and also provides a mechanism to overlay explanations from different result sets. The dual mining module ensures that all groups that are selected have the location attribute associated with them.

This module has to visualize results generated for two forms of feedbacks, ratings and tags. The set of groups that are generated by maximizing the objective functions based on the user query can be considered as feedback interpretation object. Each set of such objects are then rendered as a Choropleth map [12] using the average group rating for shading. Dark red corresponds to lowest rating while dark green denotes the highest and the intermediate values are represented by the red-green gradient. For each object, the tags associated with are visualized using tag clouds. Tag clouds are a particularly appealing mechanism to visualize user feedback in the form of tags. Popular tags are shown at a higher font size. Additionally each tag in the tag cloud is also proportionally colored based on the *topic* it is associated with. The tag to topic assignment is identified by performing LDA on R_q. Each group is also annotated with icons that identify the attribute value pairs used to define it. The set of these Choropleth maps form an *exploration*. Such an exploration is formed from the same set of input feedback tuples R_q and constraints, but provide different perspective in terms of meaningful rating interpretations. As an example, one exploration might identify set of reviewer groups that provide similar tags for diverse items while the next would identify the reviewer groups that provide diverse tags for the same. Collectively, the different visualizations provide a comprehensive

insight into reviewer feedback patterns. The system also allows a user to drill deeper and view lower level aggregate statistics. For example, if the original geo-condition was over a state, the drill down provides city level rating and tagging statistics.

3.7.2 User Interface

The ViSTaR system can work on any collaborative content site that provides data as described in Section 3.6. For the purpose of the demo, we use the 1M rating data from MovieLens. We integrate the MovieLens data with information available from IMDB, in order to include additional item attributes such as actors and directors.

ViSTaR is a web based system that allows a user to enter queries about reviewers, items or both. The primary UI for entering is shown in Figure 3.15. A user can enter a conjunctive query by entering one or more attribute values for reviewers and/or items. The item based queries include attribute values for movie title, actor, director and genre. The user based queries involve reviewer gender, age, occupation and location. The user can enter additional search settings such as the maximum number of groups to be returned and its feedback coverage. The user can also enter additional soft constraints on the reviewer and item dimensions that influences the chosen groups. Finally, the user chooses the optimization goal on one or both of ratings and tags. It is also possible *not* to specify any attribute value pairs on the input interface - the system then performs mining on the entire dataset.

Suppose a user wants to identify a diverse set of items for which a similar group of users, say young male reviewers, share similar rating and tagging behavior. Figure 3.15 is the corresponding input query interface. The user enters Young as the reviewer age and Male as the reviewer gender for query; the item dimension is left blank. In the Mining Criterion section, the user selects diversity for item dimension, similarity for tag dimension and similarity for rating dimension. If the user wants to

identify diverse items for which young male reviewers have used similar tags, she needs to enter similarity for tag dimension and keep the rating dimension blank. Again, if the user wants to identify similar items for which young male reviewers have used diverse ratings and tags, she needs to select similarity for item dimension, diversity for tag dimension and diversity for rating dimension. In this way, the user can choose from a set of options to mine results for a series of different analysis tasks. Note that, a user is also capable of exploring the feedback behavior of different sub-populations by submitting different combinations of the reviewer attributes as the query.

VisTaR identifies that diverse item groups ⟨actor, Jennifer Aniston⟩ and ⟨actor, Justin Timberlake⟩ receives similar feedback from young male reviewers, as seen in Figure 3.16. Each distinct item group is displayed in a carousel; the user can flip through the arrows to view the other groups. Each item group is associated with a map which highlights the states that have significant number of user feedback. For example, in Figure 3.16 (left), male users from Texas and California have the highest proportion of feedback for the item group ⟨actor , Jennifer Aniston⟩.

If the query results in reviewer groups, then each such group always specify the state as their geo-condition in order to allow rendering of the explanation in the map. The average rating of the group is used for highlighting the state. We use a red (rating 1.0) to green (rating 5.0) Likert Scale for depicting the average rating. The tag feedback provided for each group is presented as a tag cloud on the right. The reviewer attributes associated with a group are highlighted through visual cues such as icons. The color of the pin holding the icons depicts the age of the sub-population.

3.7.3 Demo Plan

Our demo allows the audience to use a web interface and specify arbitrary search query involving reviewer and/or movie attributes. The audience can spec-

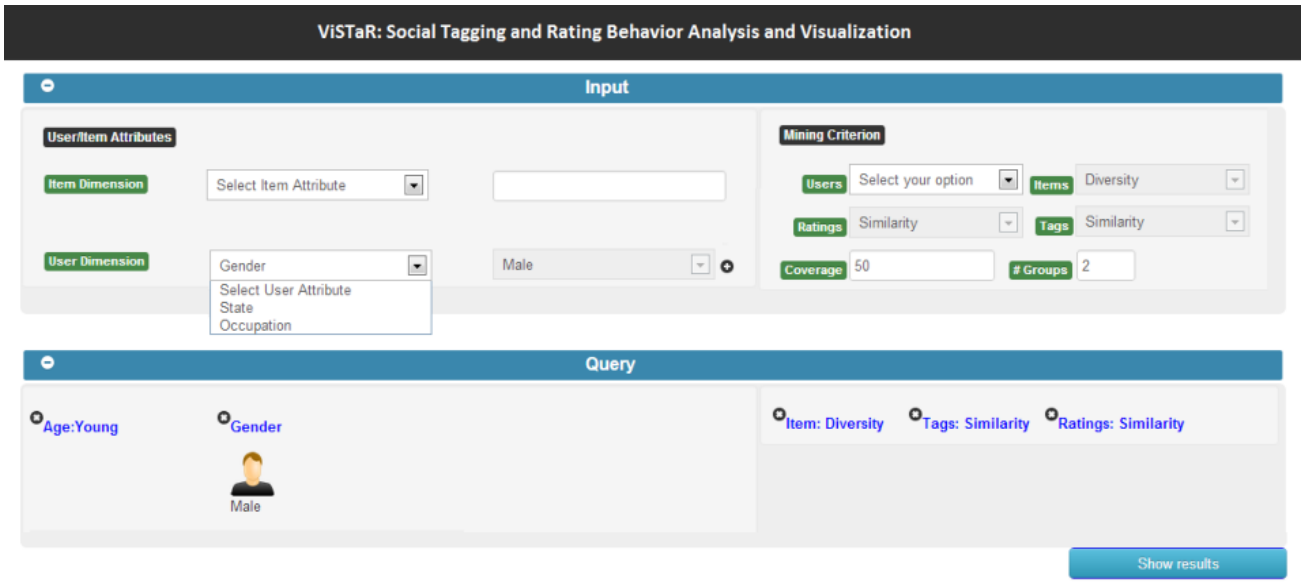


Figure 3.15. Primary User Interface of ViSTaR.

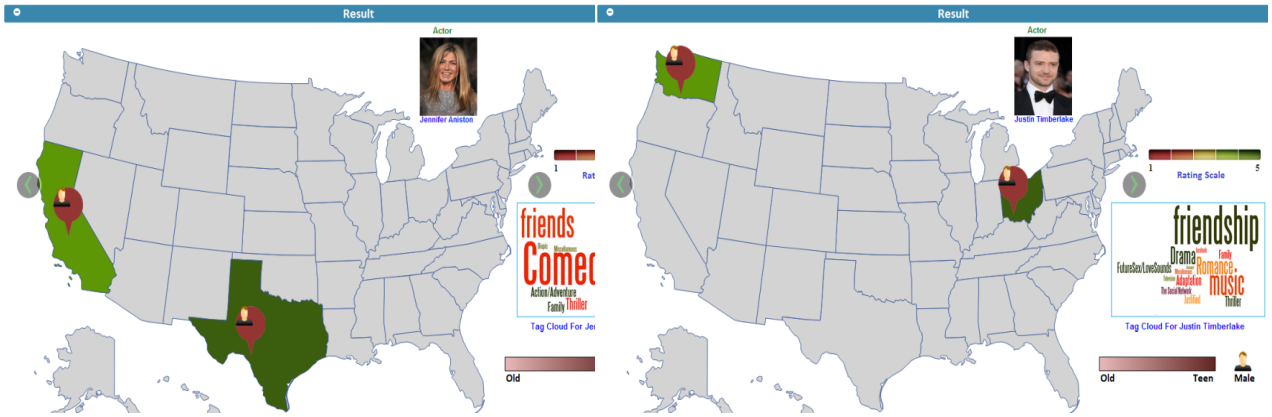


Figure 3.16. ViSTaR Result for Query in Figure 3.15 - ⟨actor, Jennifer Aniston⟩ in left, ⟨actor, Justin Timberlake⟩ in right.

ify other search settings and soft constraints on different dimensions in the mining criterion section for personalized mining. Based on the query, the constraints and the optimization criterion, our system will display the result set which the audience can examine to have a better understanding of the social feedback trends for the query. The audience can also filter the results by additional constraints for a more fine-grained analysis. Such exploration will give the audience a deeper appreciation

of our system's utility to aid users make informed judgments about movies quickly. It will also clearly show the superiority of VISTAR to explain social feedback behavior for items over information presented in existing content sites.

Thus, we develop a comprehensive framework for mining and exploring social feedback behavior in order to help future customers make purchasing decisions effectively and efficiently. Now, we move to the second part of the dissertation that concerns exploratory mining of collaborative social content to help content producers improve business.

PART II

EXPLORATORY MINING OF COLLABORATIVE CONTENT FOR CONTENT PRODUCERS

"*If you don't understand people, you don't understand business.*"

- Simon O. Sinek

CHAPTER 4

New Item Design

4.1 Introduction

The widespread use and popularity of online collaborative tagging sites has created new challenges and opportunities for designers of *web items* such as electronics products, travel itineraries, popular blogs, etc. Various websites today (e.g., Flickr for photos, YouTube for videos, Amazon for different products) encourage users to actively participate by assigning labels or *tags* to online resources with a purpose to promote their contents and allow users to share, discover and organize them. An increasing number of people are turning to online reviews and user-specified tags to choose from among competing items. Products with desirable tags (e.g., `modern`, `reliable`, etc.) have a higher chance of being selected by prospective customers. This creates an opportunity for designers to build items that are likely to attract desirable tags when published. In addition to traditional marketplaces like electronics, autos or apparel, tag desirability also extends to other diverse domains. For example, music websites such as `Last.fm` use social tags to guide their listeners in browsing through artists and music. An artist creating a new musical piece can leverage the tags that users have selected, in order to select the piece's attributes (e.g. acoustic and audio features) that will increase its chances of becoming popular. Similarly, a blogger can select a topic based on the tags that other popular topics have received. We investigates this novel tag maximization problem, i.e., how to decide the attribute values of new items and to return the top-k *best* items that are likely to attract the maximum number of desirable tags. We provide more details as follows.

Assume we are given a training data of objects (i.e., products), each having a set of well-defined features (i.e., attributes) and a set of user-submitted tags (e.g., cell phones on Amazon's website, each described by a set of attributes such as display size, Operating System and associated user tags such as `lightweight`, `easy to use`). From this training data, for each distinct tag, we assume a classifier has been constructed for predicting the tag given the attributes. Tag prediction is a recent area of research (see Section 7 for discussion of related work), and the existence of such classifiers is a key assumption in our work. In addition to the product's explicitly specified attributes, other implicit factors also influence tagging behavior, such as the perceived utility and product quality to the user, the tagging behavior of the user's friends, etc. However, pure *content-based* tag prediction approaches are often quite effective – e.g., in the context of laptops, attributes such as smaller dimensions and the absence of a built-in DVD drive may attract tags such as `portable`.

Given a query consisting of a subset of tags that are considered *desirable*, our task is to suggest a new product (i.e., a combination of attribute values) such that the expected number of desirable tags for this potential product is maximized. This can be extended to the top-k version, where the task is to suggest the k potential products with the highest expected number of desirable tags. In addition to the set of desirable tags, our problem can also consider a set of *undesirable* tags, e.g. `unreliable`. The optimization goal in this case is to maximize the number of desirable tags and minimize the undesirable ones - a simple combination function is to optimize the expected number of desirable tags minus the expected number of undesirable tags. In our discussion so far, we have not explained how the set of desirable and undesirable tags are created. Although this is not the focus of this chapter, we mention several ways in which this can be done. For example, domain experts could study the set of tags and mark them accordingly. Automated methods may involve leveraging the

user rating or the sentiment of the user review to classify tags as desirable, undesirable or unimportant.

The dynamics of social tagging has been an active research area in recent years. However related literature primarily focuses on the problems of tag prediction, including cold-start recommendation to facilitate web-based activities. To our best knowledge, tags have not been studied in the context of product design before. Of course, real-world product design is a complex task, and is an area that has been heavily studied in economics, marketing, industrial engineering and more recently in computer science. Many factors like the cost and return on investment are currently considered. We argue that the user feedback (in the form of tags of existing competing products) should be taken into consideration in the design process, especially since online user tagging is extremely widespread and offers unprecedented opportunities for understanding the collective opinion and preferences of a huge consumer base. We envision user tags to be one of the several factors in product design that can be used in conjunction with more traditional factors - e.g., our algorithms return k potential new products that maximize the number of desirable tags; and this information can assist content producers, who can then further post-process the returned results using additional constraints such as profitability, price, resource constraints, product diversity, etc. Moreover, product designers can explore the data in an interactive manner by picking and choosing different sets of desirable tags to get insight on how to build new products that target different user populations − e.g., in the context of cell phones, tags such as `lightweight` and `powerful` target professionals, whereas tags such as `cheap`, `cool` target younger users.

Solving the tag maximization problem is technically challenging. In most product bases, complex dependencies exist among the tags and products, and it is difficult to determine a combination of attribute values that maximizes the expected number

of desirable tags. Herem we consider the very popular Naive Bayes Classifier for tag prediction [1]. Extending our work for other popular classifiers is one of our future research directions. As one of our first results, we show that even for this classifier (with its simplistic assumption of conditional independence), the tag maximization problem is NP-Complete. Given this intractability result, it is important to develop algorithms that work well in practice. A highlight of our work is that we have avoided resorting to heuristics, and instead have developed principled algorithms that are practical *and at the same time* possess compelling theoretical characteristics.

Our first algorithm is a novel exact top-k algorithm ETT (Exact Two-Tier Top-k algorithm) that performs significantly better than the naive brute-force algorithm (which simply builds all possible products and determines the best ones), for moderate problem instances. Our algorithm is based on nontrivial adaptations of top-k query processing techniques (e.g., [32]), but is not merely a simple extension of TA. The complexity arises because the problem involves maximizing a sum of terms, where within each term there is a product of quantities which are interdependent with the quantities from the other terms. Our top-k algorithm and has an interesting two-tier architecture. At the bottom tier, we develop a sub-system for each distinct tag, such that each sub-system has the ability to compute on demand a stream of products in order of decreasing probability of attracting the corresponding tag, *without having to pre-compute* all possible products in advance. In effect, each sub-system simulates *sorted access* efficiently. This is achieved by partitioning the set of attributes into smaller groups (thus, each group represents a *partial* product), and running a separate merge algorithm over all the groups. The top tier considers the products retrieved from each sub-system in a round-robin manner, computes the expected number of

[1] Naive Bayes Classifiers are often effective, rival the performance of more sophisticated classifiers, and are known to perform well in social network applications. For instance, Pak and Paroubek [31] show that Naive Bayes performs better than SVM and CRF in classifying the sentiment of blogs.

desirable tags for each retrieved product, and stops when a threshold condition is reached. Although in the worst case this algorithm can take exponential time, for many datasets with strong correlations between attributes and tags, the stopping condition is reached much earlier.

However, although the exact algorithm performs well for moderate problem sizes, it did not easily scale to much larger sized datasets. Heuristic techniques like hill-climbing, branch and bound, etc. does not guarantee any sort of worst case behavior, either in running time or in item quality. Designing approximation algorithms with guaranteed behavior is challenging, since no known approximation algorithm for other NP-Complete problems can be easily modified for our case. Our exact algorithm ETT can be modified to serve as an approximation algorithm - we can *change the threshold condition* such that the algorithm stops when the threshold is within a small user-provided approximation factor of the top-k product scores produced thus far. This algorithm can guarantee an approximation factor in the quality of products returned, but would run in exponential time in the worst case. In this work, we develop a novel approximation algorithm PA (Poly-Time Approximation algorithm) that runs in worst case polynomial time, *and* also guarantees a provable bound on the approximation factor in item quality. The principal idea is to group the desirable tags into *constant-sized* groups, find the top-k items for each sub-group, and output the overall top-k items from among these computed items. Interestingly, we note that in this algorithm we create sub-problems by grouping tags; in contrast in our exact algorithm we create sub-problems (i.e., subsystems) by grouping attributes. For each sub-problem thus created, we show that it can be solved by a polynomial time approximation scheme (PTAS) given any user-defined approximation factor. For each sub-problem thus created, we show that it can be solved by a polynomial time approximation scheme (PTAS) given any user-defined approximation factor. The al-

gorithm's overall running time is exponential only in the (constant) size of the groups, thus giving overall a polynomial time complexity.

We experiment with synthetic as well as real datasets crawled from the web to compare our algorithms. User study on the real dataset demonstrates that the items suggested by our algorithms appear to be meaningful. With regard to efficiency, the exact algorithm performs well on moderate problem instances, whereas the approximation algorithm scaled very well for larger datasets.

4.2 Problem Framework

We model a collaborative tagging site $\mathcal{D}$ as a triple $\langle \mathcal{U}, \mathcal{I}, \mathcal{R} \rangle$, representing the set of reviewers (i.e., users), the set of items (i.e., products) and the feedback (i.e, tag vocabulary), respectively. Let $\{p_1, p_2, ..., p_{n_p}\}$ be a collection of n_p products, where each entry is defined over the item schema $\mathcal{I}_\mathcal{A} = \langle pa_1, pa_2, ..., pa_{m_p} \rangle$ and the tag dictionary space $\mathtt{T} = \{\mathtt{T_1}, \mathtt{T_2}, ..., \mathtt{T_{n_c}}\}$. For the ease of representation (and since we do not consider user details in the framework), let m_p be m. A product p is a set of attribute values, $p = \langle pv_1, pv_2, ..., \rangle$ where each pv_y is a value for the attribute $pa_y \in \mathcal{I}_\mathcal{A}$. Each attribute pa_y can take one of several values pv_y from a multi-valued categorical domain D_i, or one of two values 0, 1 if a boolean dataset is considered. A tag $\mathtt{T_j}$ is a bit where a 0 implies the absence of a tag and a 1 implies the presence of a tag for product p. Each product is thus a vector of size $(m + n_c)$, where the first m positions correspond to a vector of attribute values, and the remaining n_c positions correspond to a boolean vector.[2,3]

We assume such a dataset has been used as a training set to build Naive Bayes Classifiers (NBC), that classify tags given the attribute values (one classifier per tag).

[2]Our framework allows numeric attributes, but as is common with Naive Bayes Classifiers, we assume that they have been appropriately binned into discrete ranges.

[3]A more complex framework which leverages the frequencies of tags is left for future work.

The classifier for tag $\mathtt{T_j}$ defines the probability that a new item p is annotated by the tag $\mathtt{T_j}$:

$$\begin{aligned} Pr(\mathtt{T_j} \mid p) &= Pr(\mathtt{T_j} \mid pv_1, pv_2, ..., pv_m) \\ &= \frac{Pr(\mathtt{T_j}).\Pi_{i=1}^{m} Pr(pv_i \mid \mathtt{T_j})}{Pr(pv_1, pv_2, ..., pv_m)} \end{aligned} \tag{4.1}$$

where pv_i is the value of p for attribute pa_i. The probabilities $Pr(pv_i \mid \mathtt{T_j})$ are computed using the dataset. In particular, $Pr(pv_i \mid \mathtt{T_j})$ is the proportion[4] of items tagged by $\mathtt{T_j}$ that have $pa_i = pv_i$. $Pr(\mathtt{T_j})$ is the proportion of items in the dataset that has $\mathtt{T_j}$.

Similarly, we compute the probability $Pr(\mathtt{T_j}' \mid p)$ of an item p not having tag $\mathtt{T_j}$:

$$Pr(\mathtt{T_j}' \mid p) = \frac{Pr(\mathtt{T_j}').\Pi_{i=1}^{m} Pr(pv_i \mid \mathtt{T_j}')}{Pr(pv_1, pv_2, ..., pv_m)} \tag{4.2}$$

We know that $Pr(\mathtt{T_j} \mid p) + Pr(\mathtt{T_j}' \mid p) = 1$; hence from Equations 4.1, 4.2 we get :

$$\begin{aligned} Pr(pv_1, pv_2, ..., pv_m) &= Pr(\mathtt{T_j}).\Pi_{i=1}^{m} Pr(pv_i \mid \mathtt{T_j}) + \\ &\quad Pr(\mathtt{T_j}').\Pi_{i=1}^{m} Pr(pv_i \mid \mathtt{T_j}') \end{aligned} \tag{4.3}$$

From Equations 4.1, 4.3,

$$\begin{aligned} Pr(\mathtt{T_j} \mid p) &= Pr(\mathtt{T_j} \mid pv_1, pv_2, ..., pv_m) \\ &= \frac{Pr(\mathtt{T_j}).\Pi_{i=1}^{m} Pr(pv_i \mid \mathtt{T_j})}{Pr(\mathtt{T_j}).\Pi_{i=1}^{m} Pr(pv_i \mid \mathtt{T_j}) + Pr(\mathtt{T_j}').\Pi_{i=1}^{m} Pr(pv_i \mid \mathtt{T_j}')} \\ &= \frac{1}{1 + \frac{Pr(\mathtt{T_j}')}{Pr(\mathtt{T_j})} \Pi_{i=1}^{m} \frac{Pr(pv_i \mid \mathtt{T_j}')}{Pr(pv_i \mid \mathtt{T_j})}} \end{aligned}$$

For convenience we use the notation

$$R_j = \frac{Pr(\mathtt{T_j}')}{Pr(\mathtt{T_j})} \Pi_{i=1}^{m} \frac{Pr(pv_i \mid \mathtt{T_j}')}{Pr(pv_i \mid \mathtt{T_j})} \tag{4.4}$$

[4]The observed probabilities are smoothened using the Bayesian m-estimate method [33]. We note that more sophisticated Bayesian methods that use an informative prior may be employed instead.

Consider a query which picks a set of desirable tags $T^d = \{\mathtt{T_1}, \ldots, \mathtt{T_z}\} \subseteq \mathtt{T}$. The expected number of desirable tags $\mathtt{T_j} \in T^d$ that a new item p, characterized by (pv_1, pv_2, ..., pv_m) is annotated with, is given by:

$$\mathbb{E}(p, T^d) = \Sigma_{j=1}^{z} \frac{1}{1 + R_j} \tag{4.5}$$

We are now ready to formally define the main problem.

Problem 4.2.1. *Tag Maximization Problem: Given a dataset $\mathcal{D}$ of tagged items $\{p_1$, p_2, ..., $p_n\}$, and a query T^d, design k new items that have the highest expected number of desirable tags they are likely to receive, given by Equation 4.5.*

We explain our algorithms in a boolean framework, which can be readily generalized to handle the case of categorical data. We also assume that all tags are of equal *weight*– if tags are of varying importance, Equation 4.5 can be re-written as a weighted sum, and all our proposed algorithms can be modified accordingly.

4.3 Complexity Analysis

In this section, we analyze the computational complexity of the main problem. Clearly, the brute-force exhaustive search will require us to design all possible 2^m number of products and compute $\mathbb{E}(p, T^d)$ for each of them. This naive approach will run in exponential time. However, we next give a proof sketch that the Tag Maximization problem is NP-Complete, which leads us to believe that in the worst case we may not be able to do much better than the naive approach.

Theorem 4.3.1. *The Tag Maximization problem is NP-Complete even for boolean databases and for $k = 1$.*

Proof: The membership of the decision version of the problem in NP is obvious. To verify NP-hardness, we reduce the 3SAT problem to the decision version of our problem. We first reduce the 3SAT problem to the minimization version of the op-

timization problem, represented as $\mathbb{E}^{min}(p,\ T^d)$ and then reduce the minimization problem to the problem of maximization of expected number of tags being present.

Reduction of 3SAT to decision version of $\mathbb{E}^{min}(p,\ T^d)$:

3SAT is the popular NP-complete boolean satisfiability problem in computational complexity theory, an instance of which concerns a boolean expression in conjunctive normal form, where each clause contains exactly 3 literals. Each clause C_j is mapped to a tag T_j in the instance of $\mathbb{E}^{min}(p,\ T^d)$ and each variable x_i is mapped to attribute value a_i. We make the following assignments so that if there is a boolean assignment vector $\vec{a} = [a_1,\ ...,\ a_m]$ that satisfies 3SAT, then $\mathbb{E}^{min}(p,\ T^d)$ equals zero (and if $\vec{a}$ does not satisfy 3SAT, then $\mathbb{E}^{min}(p,\ T^d)$ has a non-zero sum).

- For a variable x_i specified as positive literal in 3SAT, set $\Pr(pv_i = 0 \mid \mathtt{T_j}) = 1$
- For a variable x_i specified as negative literal in 3SAT, set $\Pr(pv_i = 1 \mid \mathtt{T_j}) = 1$
- For a particular clause and for the unspecified attributes (variables), set $\Pr(pv_i = 0 \mid \mathtt{T_j}) = \Pr(pv_i = 1 \mid \mathtt{T_j}) = 1$

For example, consider 3SAT instance $(\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_4)$. For each tag, we create two products. For instance for the first tag, x_1 (that corresponds to A_1) is negative and hence for both the first and second product it is $A_1 = 1$. x_4 is missing from the first tag, hence for the first product it is $A_4 = 0$ and for the second it is $A_4 = 1$.

Table 4.1. Table of boolean attributes and tags

Attributes				Tags	
A_1	A_2	A_3	A_4	T_1	T_2
1	0	1	0	1	0
1	0	1	1	1	0
0	1	0	1	0	1
0	1	1	1	0	1

Reduction of $\mathbb{E}^{min}(p,\ T^d)$ to $\mathbb{E}(p,\ T^d)$:

If we have a boolean assignment vector $\vec{a} = [a_1, ..., a_m]$ that minimizes the expected number of tags being present, we have the corresponding $\Pr(\mathtt{T_j}' \mid pv_1, pv_2, ..., pv_m)$. Hence, we get $\Pr(\mathtt{T_j} \mid pv_1, pv_2, ..., pv_m) = 1 - \Pr(\mathtt{T_j}' \mid pv_1, pv_2, ..., pv_m)$ that maximizes the expected number of tags being present. □

Section 4.4.1 and Section 4.4.2 next describe our algorithmic solutions to this NP-Complete problem in a boolean framework.

4.4 Algorithms

4.4.1 Optimal Algorithm

A brute-force exhaustive approach (henceforth, referred to as Naive) to solve the Tag Maximization problem requires us to design all possible 2^m number of items and compute $\mathbb{E}(p, T^d)$ for each possible item. Note that the number of items in the dataset is not important for the execution cost, since an initialization step can calculate all the conditional tag-attribute probabilities by a single scan of the dataset. Although general purpose pruning-based optimization techniques (such as branch-and-bound algorithms) can be used to solve the problem more efficiently than Naive, such approaches are only limited to constructing the top-1 item, and it is not clear how they can be easily extended for $k > 1$.

In the following subsection, we propose a novel exact algorithm for any k based on interesting and nontrivial adaptations of top-k query processing techniques. This algorithm is shown in practice to explore far fewer item candidates than Naive, and works well for moderate problem instances.

4.4.1.1 Exact Two-Tier Top-k Algorithm

We develop an exact *two tier* top-k algorithm (ETT) for the Tag Maximization problem. For simplicity, henceforth we refer to desirable tags as just tags. The main

idea of *ETT* is to determine the best items for each individual tag in tier-1 and then match these items in tier-2 to compute the globally best items (across all tags). Both tiers use pipelined techniques to minimize the amount of accesses, as shown in Figure 4.1. The output of tier-1 is z unbounded buffers (one for each tag) of complete items, ordered by decreasing probability for the corresponding tag. These buffers are not fully materialized, but may be considered as sub-systems that can be accessed on demand in a pipelined manner.

In tier-2, the top items from the z buffers are combined in a pipelined manner to produce the global top-k items, akin the *Threshold Algorithm* (TA) [32]. In turn, tier-2 makes GetNext() requests (see Figure 4.1) to various buffers in tier-1 in round-robin manner. In tier-1, for each specific tag, we partition the set of attributes into subsets, and for each subset of attributes we precompute a list of all possible partial attribute value assignments, ordered by their score for the specific tag (the score will be defined later). The partial items are then scanned and joined, leveraging results from Rank-Join algorithms [34] that support top-k ranked join queries in relational databases, in order to feed information to tier-2. A single GetNext() for a specific tag may translate to multiple retrievals from the underlying lists of partial items in tier-1, which are then joined into complete items and returned.

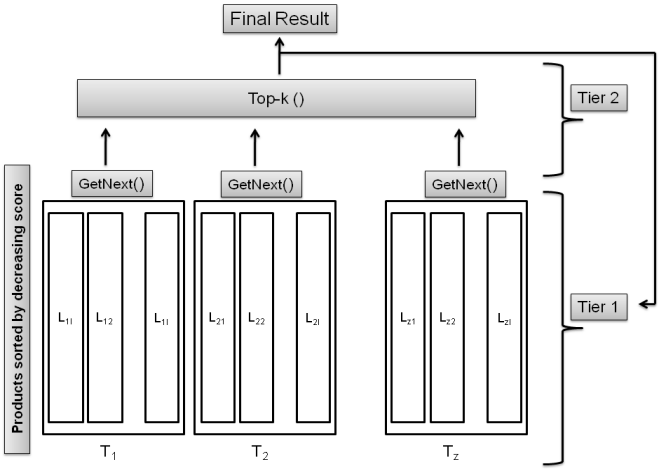


Figure 4.1. Two-Tier Top-K Algorithm Framework.

4.4.1.2 Tier-1

Suppose we partition the m attributes into l subsets, where each subset has $m' = \frac{m}{l}$ attributes as follows: $\{pv_1, \ldots, pv_{m'}\}$, $\{pv_{m'+1}, \ldots, pv_{2m'}\}$, ..., $\{pv_{m-m'+1}, \ldots, pv_m\}$. We create partial item lists $L_{j1}, \ldots, L_{jl}$ for each tag $\mathtt{T_j}$. Each list L_{ji} has $2^{m'}$ entries (partial items). Consider the first list L_{j1}. The *score* of a partial item $p^p \in L_{j1}$ with attribute values $pv_1, \ldots, pv_{m'}$ for $\mathtt{T_j}$ is

$$\mathbb{E}_{partial}(p^p, \{\mathtt{T_j}\}) \quad = \quad \sqrt[l]{P_j}.\Pi_{i=1}^{m'} \frac{Pr(pv_i \mid \mathtt{T_j}')}{Pr(pv_i \mid \mathtt{T_j})} \tag{4.6}$$

where $P_j = \frac{Pr(\mathtt{T_j}')}{Pr(\mathtt{T_j})}$. Note that the l-th root of P_j is used in order to distribute the effect of P_j from Equation 4.4 to the l lists, such that when they are combined using multiplication, we get P_j.

Lists L_{jl} are ordered by descending $\frac{1}{E_{partial}}$, since R_j appears on the denominator of Equation 4.5. The l lists are accessed in round-robin fashion and for every combination of partial items from the lists, we join them to build a complete item and resolve its exact score by Equation 4.5.

An item is returned as a result of GetNext() to tier-2 if its score is higher than the MPFS (*Maximum Possible Future Score*), which is the upper bound on the score of an unseen item. To compute MPFS, we assume that the current entry from a list is joined with the top entries from all other lists:

$$MPFS = \frac{1}{1 + max((s_{j_1}.h_{j_2}.. \cdot h_{jl}), (h_{j_1}.s_{j_2}.. \cdot h_{jl}), .., (h_{j_1}.h_{j_2}.. \cdot s_{jl}))} \tag{4.7}$$

where s_{ji} and h_{ji} are the last seen and top entries from list L_{ji} respectively.

4.4.1.3 Tier-2

In this tier, the z unbounded buffers, one for each tag, are combined using the summation function, as shown in Equation 4.5. Each item from one buffer matches

exactly one entry (the identical item) from each of the other buffers. Items are retrieved from each buffer using GetNext() operations, and once retrieved we directly compute its score for all other tags by running each Naive Bayes classifier, without using the process of tier-1. An item is output if its score is higher than the threshold, which is the sum of the last seen scores from all z buffers. A bounded buffer with the k best results so far is maintained. On termination, this buffer is returned as the top-k items.

The pseudocode of ETT is shown in Algorithm 8.

Table 4.2. Example tagged items dataset

	Attribute				Tag	
ID	A_1	A_2	A_3	A_4	T_1	T_1
1	0	0	0	1	0	0
2	0	1	0	0	0	1
3	0	1	0	1	0	0
4	0	1	1	1	1	1
5	1	0	0	0	1	0
6	1	0	0	1	0	1
7	1	0	1	1	1	1
8	1	1	0	1	0	1

Example 4.4.1. Consider the boolean dataset of 10 objects, each entry having 4 attributes and 2 tags in Table 4.2. We partition the 4 attributes into groups of 2 attributes: $(A_1,\ A_2)$ form list L_{j1} and $(A_3,\ A_4)$ form list L_{j2}. We run NBC and calculate all conditional tag-attribute probabilities. The algorithm framework for the running example is presented in Figure 4.2. List L_{11} and L_{12} under tag T_1 is sorted in decreasing order of $\frac{1}{E_{partial}}$, given by Equation 4.6 (and, similarly for L_{21} and L_{22} under tag T_2).

In iteration 1, call to Threshold() in tier-2 calls GetNext() for T_1 and T_2 respectively in tier-1. During GetNext(T_1), join-1 builds item 1010, whose $score_1(1010) =$

Algorithm 8 ETT (Naive Bayes probabilities, attributes per group m', k): top-k exact items

//Main Algorithm

Top-k-Buffer $\leftarrow$ {}
for $j = 1$ to z **do**
  $B_j \leftarrow$ {} // unbounded buffer of candidate results-items per tag
  for $i = 1$ to l **do**
    s_{ji}, $h_{ji} \leftarrow$ top entry from list L_{ji}
  end for
end for
Call Threshold()

//Method Threshold() – Tier-2

while true **do**
  for $j = 1$ to z **do**
    $(p_j, score_j(p_j) \leftarrow$ GetNext(j)
    ExactScore(p_j) $\leftarrow$ Compute for p_j by Equation 4.5
  end for
  Update Top-k-Buffer with new items if necessary
  MinK $\leftarrow$ lowest score in Top-k buffer
  $\alpha \leftarrow \sum_j score_j(p_j)$ // Threshold
  if MinK $\geq \alpha$ **then**
    return top-k items
  end if
end while

//Method GetNext(j) : $(p_j, score_j(p_j))$ – Tier-1

while true **do**
  Compute MPFS by Equation 4.7
  // $score_j(p)$ for item p is defined as $1/(1 + R_j)$ (R_j defined by Equation 4)
  if B_j has an item p with $score_j(p) > MPFS$ **then**
    return (p, $score_j(p)$) AND remove it from B_j
  end if
  Retrieve next entry p^p from a list L_{ji} in round robin and advance s_{ji}
  Join p^p with all combinations of partial items from other lists and create all items $NewItems$
  Add $NewItems$ to buffer B_j of candidate results-items
end while

0.95 and MPFS(1010) =0.95. Since $score_1 \geq$ MPFS, (1010,0.95) is returned to tier-2. GetNext(T_2) returns (1111,0.93) to tier-2. In tier-2, call to Threshold() returns ExactScore(1010) =1.70, ExactScore(1111) =1.75. Now, top-k-buffer gets 1111; MinK=1.75 and α=1.88. Since MinK $\leq \alpha$, we continue to iteration 2. In iteration 2, we proceed similarly. GetNext(T_1) returns (1011, 0.92) and GetNext(T_2) returns (1110,0.88) to tier-2. Call to Threshold() in tier-2 gives ExactScore(1011) =1.76, ExactScore(1110) =1.77. The top-k-buffer is updated to 1110; MinK=1.77 and α=1.79. Since MinK $\leq \alpha$, we continue to iteration 3. In tier-1 of iteration 3, GetNext(T_1) returns (0010,0.89) and GetNext(T_2) returns (0111,0.84) to tier-2. Then Threshold() is called and we get ExactScore(0010) =1.76, ExactScore(0111) =1.77. The Bounded Buffer continues to be 1110; MinK=1.77 and α=1.74. We see that MinK $\geq \alpha$. Hence, ETT terminates and returns 1110 as the top-1 item. Thus, ETT returns the best item by just looking up 6 items, instead of 16 items (as in Naive algorithm). □

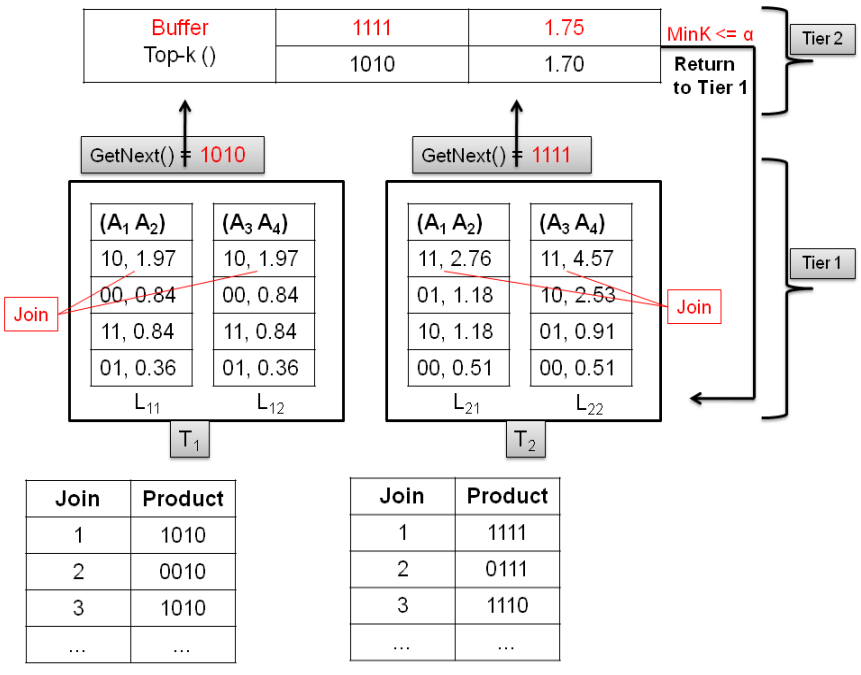


Figure 4.2. Iteration 1: Exact Two-Tier Top-K Algorithm for Example in Table 4.2.

Grouping of Attributes: The ETT algorithm partitions the set of attributes into smaller groups (each group representing a partial product), which we join to retrieve the best product to feed to tier-2. We can employ state-of-art techniques to create a graph, where each node corresponds to an attribute and an edge between two attributes is weighed by the absolute value of the correlation between them, and then perform graph clustering techniques for partitioning the attributes into as many groups as the desired number of lists. If the sets of attributes are highly correlated, such grouping of attributes would make our ETT algorithm reach the stopping condition earlier than it would if the attributes are grouped arbitrarily.

4.4.2 Approximation Algorithm

The exact algorithm of Section 4.4.1.1 is feasible only for moderate instances of the problem. For larger problem instances, in this section we discuss a principled approximation algorithm (PA, or *polynomial time approximation algorithm*) that provides guarantee in the quality of the top-k results *as well as* running time.

4.4.2.1 Polynomial Time Approximation Algorithm

The main idea is to group the desirable tags into *constant-sized* groups of z' tags each, find the top-k items for each subgroup, and output the overall top-k items from among these computed items.[5] For each sub-problem thus created, we show that it can be solved by a *polynomial time approximation scheme* (PTAS) [35], i.e., can be solved in polynomial time given any user-defined approximation factor ϵ (function of compression factor σ and m; details later in Theorem 2). The overall running time of the algorithm is exponential only in the (constant) size of the groups, thus giving overall a polynomial time complexity.

[5]Interestingly, we note that in this algorithm we create (z/z') sub-problems by grouping tags; in contrast in our exact ETT algorithm we create sub-problems (i.e., subsystems) by grouping attributes.

We now consider a sub-problem consisting of only a constant number of tags, z'. We also restrict our discussion to the case $k = 1$ (more general values of k are discussed later). We shall design a polynomial time approximation scheme (PTAS) for this sub-problem. A PTAS is defined as follows. Let $\epsilon > 0$ be any user-defined parameter. Given any instance of the sub-problem, let PTAS return the item p_a. Let the optimal item be p_g. The PTAS should run in polynomial time, and $ExactScore(p_a) \geq (1 - \epsilon).ExactScore(p_g)$.

In describing the PTAS, we first discuss a simple exponential time exact top-1 algorithm for the subproblem, and then show how it can be modified to the PTAS. Given m boolean attributes and z' tags, the exponential time algorithm makes m iterations as follows: As an initial step, it produces the set S_0^u consisting of the single item $\{0^m\}$ along with its z' scores, one for each tag. In the first iteration, it produces the set containing two items $S_1^u = \{0^m, 10^{m-1}\}$ each accompanied by its z' scores, one for each tag. More generally, in the ith iteration, it produces the set of items $S_i^u = \{\{0,1\}^i \times 0^{m-1}\}$ along with their z' scores, one for each tag. Each set can be derived from the set computed in the previous iteration. Once m iterations have been completed, the final set S_m^u contains all 2^m items along with their exact scores, from which the top-1 item can be returned, which is that product for which the sum of the z' scores is highest. However, this algorithm takes exponential time, as in each iteration the sets double in size.

The main idea of the PTAS is to not allow the sets to become exponential in size. This is done by *compressing* each set S_i, having the same form as S_i^u and $S_i \subseteq S_i^u$, produced in each iteration to another smaller set S_i', so that they remain polynomial in size. Each item entry in S_i can be viewed as points in a z'-dimensional space,whose z' co-ordinates correspond to the item scores for z' individual tags respectively, by Equation 4.5. Essentially, we use a clustering algorithm in z'-dimensional space. For

each cluster, we pick a *representative item* that stands for all other items in the cluster, which are thereby deleted. The clustering has to be done in a careful way so as to guarantee that for the items that are deleted, the representative item's exact score is be close to the deleted item's exact score. Thus when the top-1 item of the final compressed set S'_m is returned, its exact score should not be too different from exact score of the top-1 item assuming no compression was done.

The pseudocode of PA is shown in Algorithm 9.

Algorithm 9 PA (Naive Bayes probabilities, attributes per group z', compression factor σ): top-1 approximate item in polynomial time

//Main Algorithm

Partition tags $\mathtt{T}$ into z/z' groups $\mathtt{T_1}, \ldots, \mathtt{T_{z/z'}}$
for $r = 1$ to $\frac{z}{z'}$ **do**
  $p_r \leftarrow \text{PTAS}(\mathtt{T_r})$
  Compute ExactScore(p_r) by Equation 4.5
end for
return p_r with max ExactScore

//Method $PTAS(\mathtt{T_r}) : p$

$S'_0 \leftarrow \{0^m\}$ // boolean vector of size m with all 0's
for $i = 1$ to m **do**
  $S_i = S'_{i-1} \cup S''_{i-1}$ // $S''_{i-1} : S'_{i-1}$ with ith attribute value set to 1
  // Compress S_i to S'_i using compression factor σ
  $S'_i \leftarrow \{\}$
  repeat
    $p \leftarrow$ representative item in S'_{i-1}
    $S'_i \leftarrow S'_i \cup \{p\}$
    Delete from S_i all items p' such that $\forall \mathtt{T_j} \in \mathtt{T_r}$, $|\mathbb{E}(p, \{\mathtt{T_j}\}) - \mathbb{E}(p', \{\mathtt{T_j}\})| \leq \sigma \mathbb{E}(p, \{\mathtt{T_j}\})$
  until S_i is empty
end for
return item p in S'_m with largest $|\mathbb{E}(p, \mathtt{T_r})|$

Example 4.4.2. We execute PA on the example in Table 4.2 without any grouping of tags (i.e., $z = z' = 2$, $\frac{z}{z'} = 1$), so that execution of PA is equivalent to the execution

of PTAS. We also execute the exponential time top-1 algorithm (henceforth referred to as Exponential) which was adapted to the PTAS. Let the compression factor σ be 0.5. We start with $S'_0 = \{0000\}$. In iteration 1 of PA as well as Exponential, $S_1 = \{0000, 1000\}$ and $S_1^u = \{0000, 1000\}$ with each item having two-dimensional co-ordinates (0.31, 0.20) and (0.51, 0.38) respectively. We compress S_1 in PA and get $S'_1 = \{1000\}$. 1000 is the representative item of 0000. In iteration 2 of PA, $S_2 = \{1000, 1100\}$ with two-dimensional co-ordinates (0.51, 0.38) and (0.31, 0.58) respectively. However, iteration 2 of Exponential has $S_2^u = \{0000, 1000, 0100, 1100\}$ from S_1^u, i.e., $2^2 = 4$ items with two-dimensional co-ordinates (0.31, 0.20), (0.51, 0.38), (0.16, 0.38) and (0.31, 0.58) respectively. After compression in PA, we get $S'_2 = \{1000, 1100\}$. In other words, no compression is possible for the σ under consideration. In iteration 3 of PA, we get $S_3 = \{1000, 1100, 1010, 1110\}$ with co-ordinates (0.51, 0.38), (0.31, 0.58), (0.95, 0.75) and (0.89, 0.88) respectively from S'_2. In iteration 3 of Exponential, we have $S_3^u = \{0000, 1000, 0100, 1100, 0010, 1010, 0110, 1110\}$ from S_2^u, i.e., $2^3 = 8$ items. After compression in PA, we get $S'_3 = \{1000, 1100, 1110\}$. Item 1010 is represented by 1110. In the final iteration 4 of PA, $S_4 = \{1000, 1100, 1110, 1001, 1101, 1111\}$ with co-ordinates (0.51, 0.38), (0.31, 0.58), (0.89, 0.88), (0.37, 0.52), (0.20, 0.72) and (0.82, 0.93). After compression we get $S'_4 = \{1000, 1100, 1111\}$. On the other hand, S_4^u has all 16 items as we see in Figure 4.3. The top-1 approximate item is 1111 with score 1.75 while the optimal item is 1110 with score 1.77. Figure 4.3 shows the compression in the four iterations. The boolean items in underlined red font are the cluster representatives.□

Theorem 4.4.1. *Given a user defined approximation factor ϵ, a constant sized group $\mathtt{T_r}$ of z' tags, and for $k = 1$, if we set the compression factor $\sigma = \epsilon/2m$, then:*

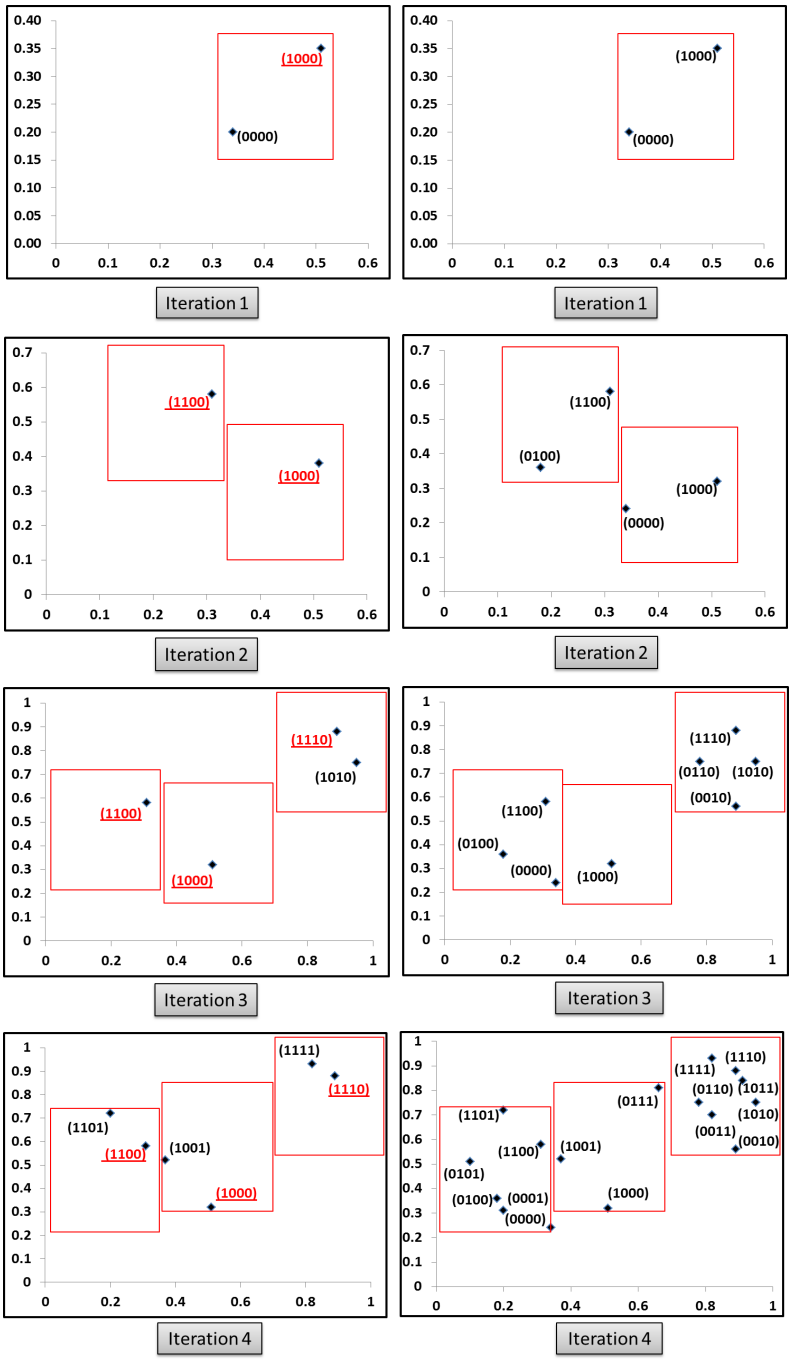


Figure 4.3. Compression in PA Algorithm (left column) vs. Exponential time Algorithm (right column) for Example Dataset of Two Tags in Table 4.2.

1. *For every product p in the uncompressed set S^u_m, there is a product p' in the compressed set S'_m for which $\mathbb{E}(p, \mathtt{T_r}) \leq (1+\sigma)^m \mathbb{E}(p', \mathtt{T_r})$*
2. *The output of PTAS($\mathtt{T_r}$) has an exact score that is at least $\frac{1}{(1+\epsilon)}$ times the exact score of the optimal product*

Proof of Part 1: Let p^u_i indicate a product belonging to uncompressed set $S^u_i = \{\{0,1\}^i \times 0^{m-i}\}$ in the i^{th} iteration where S^u_i has all 2^i products. Let p_i indicate a product belonging to set S_i having the same form as S^u_i, $S_i \subseteq S^u_i$. Let p'_i indicate an product in compressed set S'_i, $S'_i \subseteq S_i$. Note that in the i^{th} iteration, S_i is built from products in the compressed set in $(i-1)^{th}$ iteration S_{i-1}, while S'_i is built by compressing S_i. Intuitively, the idea is : for a single tag $\mathtt{T_j}$, if scores of two products p^u_i and p'_i in S_i are *close* to each other (so that p^u_i is represented by p'_i in S'_i, and p^u_i does not exist in S'_i), scores of products p^u_{i+1} and p'_{i+1} are also close to each other, where p^u_{i+1} is p^u_i and p'_{i+1} is p'_i with $(i+1)^{th}$ bit flipped. In Example 4.4.2, product 0000 in uncompressed set S_1 is represented by product 1000 in S'_1 since they are close to each other; therefore, a product 0100 in uncompressed set S^u_2 (but not in S_2 due to its removal from S'_1) must be close to some product belonging to S'_2 (which happens to be 1100 in our example).

The score of product p^u_i in uncompressed set S^u_i for tag $\mathtt{T_j}$ from Equation 4.5 is:

$$\begin{aligned} \mathbb{E}(p^u_i, \mathtt{T_j}) &= \frac{1}{1 + \frac{Pr(\mathtt{T_j}')}{Pr(\mathtt{T_j})} \Pi^m_{i=1} \frac{Pr(pv_i|\mathtt{T_j}')}{Pr(pv_i|\mathtt{T_j})}} \\ &= \frac{1}{1+PQ} (say) \end{aligned}$$

where $P = \frac{Pr(\mathtt{T_j}')}{Pr(\mathtt{T_j})} \Pi^i_1 \frac{Pr(pv_i|\mathtt{T_j}')}{Pr(pv_i|\mathtt{T_j})}$ and $Q = \frac{Pr(\mathtt{T_j}')}{Pr(\mathtt{T_j})} \Pi^m_{i+1} \frac{Pr(pv_i|\mathtt{T_j}')}{Pr(pv_i|\mathtt{T_j})}$ are proportional to the product of probabilities for the first i in p^u_i and the remaining $(m-i)$ attributes in p^u_i respectively. Similarly, the scores of products p'_i, p^u_{i+1} and p'_{i+1} for tag $\mathtt{T_j}$ are:

$$\begin{aligned}
\mathbb{E}(p'_i, \mathtt{T_j}) &= \frac{1}{1+P'Q} \\
\mathbb{E}(p^u_{i+1}, \mathtt{T_j}) &= \frac{1}{1+PQ'} \\
\mathbb{E}(p'_{i+1}, \mathtt{T_j}) &= \frac{1}{1+P'Q'}
\end{aligned}$$

where P' and Q' are proportional to the product of probabilities for the first i in p'_i and the remaining $(m-i)$ attributes in p^u_{i+1}, p'_{i+1} in which the $(i+1)^{th}$ attribute value is flipped from p^u_i.

Note that, for a single tag T_j, the score of a product is merely proportional to $\Pi_1^i \frac{Pr(pv_i|\mathtt{T_j}')}{Pr(pv_i|\mathtt{T_j})}$. Therefore, the scores of products p^u_i, p'_i, p^u_{i+1} and p'_{i+1} for tag $\mathtt{T_j}$ can be simplified and re-written as:

$$\begin{aligned}
\mathbb{E}'(p^u_i, \mathtt{T_j}) &= PQ \\
\mathbb{E}'(p'_i, \mathtt{T_j}) &= P'Q \\
\mathbb{E}'(p^u_{i+1}, \mathtt{T_j}) &= PQ' \\
\mathbb{E}'(p'_{i+1}, \mathtt{T_j}) &= P'Q'
\end{aligned}$$

We need to show that for a tag $\mathtt{T_j}$, if $\Delta_1 = \frac{\mathbb{E}'(p'_i, \mathtt{T_j})}{\mathbb{E}'(p^u_i, \mathtt{T_j})} \leq (1+\epsilon_1)$, then $\Delta_2 = \frac{\mathbb{E}'(p'_{i+1}, \mathtt{T_j})}{\mathbb{E}'(p^u_{i+1}, \mathtt{T_j})} \leq (1+\epsilon_2) \leq P(n)(1+\epsilon_1)$, where P(n) is a polynomial in n.

Assume $\mathbb{E}'(p^u_i, \mathtt{T_j})$ is close to $\mathbb{E}'(p'_i, \mathtt{T_j})$, so that p'_i represents p^u_i and that $P' \leq P$ so that the product of probabilities decrease (i.e., score of product increases) when the i^{th} attribute value is flipped. The difference in exact score between p^u_i and p'_i can be expressed as:

$$\begin{aligned}
\Delta_1 &= \frac{\mathbb{E}'(p_i, \mathtt{T_j})}{\mathbb{E}'(p^u_i, \mathtt{T_j})} \\
&= \frac{P'Q}{PQ} \\
&= \frac{b}{a} \leq (1+\epsilon_1)(say) \qquad (4.8)
\end{aligned}$$

The relationship between $\mathbb{E}'(p^u_{i+1}, \mathtt{T_j})$ and $\mathbb{E}'(p'_{i+1}, \mathtt{T_j})$ can be similarly expressed as:

$$\begin{aligned}
\Delta_2 &= \frac{\mathbb{E}'(p'_{i+1}, \mathtt{T_j})}{\mathbb{E}'(p^u_{i+1}, \mathtt{T_j})} \\
&= \frac{P'Q'}{PQ'} \\
&= \frac{P'Q\frac{Q'}{Q}}{PQ\frac{Q'}{Q}} \\
&= \frac{bc}{ac} \leq (1+\epsilon_2)(say), \text{ where } c = \frac{Q'}{Q} \\
&= \frac{b}{a} \leq (1+\epsilon_1) \text{ i.e., } \epsilon_1 = \epsilon_2
\end{aligned} \tag{4.9}$$

Note that the above formulation is analogous to the classic NP-Complete Subset Product problem. Since there are m iterations, the score of a product p in an uncompressed set can at most be $(1+\sigma)^m$ times away from the score of a product p' representing p in the uncompressed set. Therefore, for a single tag T_j, $\mathbb{E}(p, \mathtt{T_j}) \leq (1+\sigma)^m \mathbb{E}(p', \mathtt{T_j})$.

Next, we investigate how the relationship between the score of product p and p' extends for a set of tags T_r, i.e., multi-dimensional Subset Product problem. Let us consider two tags $T_r = \{T_1, T_2\}$, for which the products (optimal) in the uncompressed set are p_1 and p_2 and the products (approximation) in the compressed set are p'_1 and p'_2 respectively. Recall that, our algorithm clusters two products if their scores along each each dimension (i.e., for each tag) lie within the compression factor. We get:

$$\begin{aligned}
\mathbb{E}(p, \mathtt{T_1}) &\leq (1+\sigma)^m \mathbb{E}(p', \mathtt{T_1}) \\
\mathbb{E}(p, \mathtt{T_2}) &\leq (1+\sigma)^m \mathbb{E}(p', \mathtt{T_2})
\end{aligned} \tag{4.10}$$

$\therefore \mathbb{E}(p, \mathtt{T_1}) + \mathbb{E}(p, \mathtt{T_2}) \leq (1+\sigma)^m \{\mathbb{E}(p', \mathtt{T_1}) + \mathbb{E}(p', \mathtt{T_2})\}$

i.e., $\mathbb{E}(p, \{\mathtt{T_1}, \mathtt{T_2}\}) \leq (1+\sigma)^m \mathbb{E}(p', \{\mathtt{T_1}, \mathtt{T_2}\})$

or, $\mathbb{E}(p, \mathtt{T_r}) \leq (1+\sigma)^m \mathbb{E}(p', \mathtt{T_r})$

Proof of Part 2: Consider any tag group T_r, and let p^{OPT} be the optimal product for this group, and p^{APP} be the product returned by PTAS. From Part 1, for every product p in the set S_m^u (assuming no compression was used in any iterations), there is a product o' in the compressed set S'_m that satisfies

$$\mathbb{E}(p, \mathtt{T_r}) \leq (1+\sigma)^m \mathbb{E}(p', \mathtt{T_r}) \tag{4.11}$$

In particular, the following holds

$$\mathbb{E}(p^{OPT}, \mathtt{T_r}) \leq (1+\sigma)^m \mathbb{E}(p^{APP}, \mathtt{T_r}) \tag{4.12}$$

Since $\sigma = \epsilon/2m$, we get:

$$\begin{aligned} \mathbb{E}(p^{OPT}, \mathtt{T_r}) &\leq (1+\frac{\epsilon}{2m})^m \mathbb{E}(p^{APP}, \mathtt{T_r}) \\ &\leq e^{\frac{\epsilon}{2}} \mathbb{E}(p^{APP}, T_r) \\ &\leq (1+\epsilon)\mathbb{E}(p^{APP}, \mathtt{T_r}) \end{aligned} \tag{4.13}$$

Therefore, the output of PTAS(T_r) p^{APP} has an exact score that is at least $\frac{1}{(1+\epsilon)}$ times the exact score of the optimal product p^{OPT}. □

Theorem 4.4.2. *Given a user defined approximation factor ϵ, a non-constant number of tags z grouped into $\frac{z}{z'}$ groups of z' tags per group, and for $k = 1$, if we set the compression factor $\sigma = \epsilon/2m$, then:*

1. *The output of PA has an exact score that is at least $\frac{z'}{z(1+\epsilon)}$ times the exact score of the optimal product*
2. *PA runs in polynomial time*

Proof of Part 1: The analysis in Theorem 5.4.1 is for a single tag group $\mathtt{T_r}$ having constant number of tags z'. Since there are z/z' tag groups in totality, it is easy to see that this introduces an additional factor of z'/z to the overall approximation factor, i.e., the output of PA has an exact score that is at least $\frac{z'}{z(1+\epsilon)}$ times the exact score of the optimal product.

Proof of Part 2: To show that PA is a polynomial time algorithm, the main task is to show that the compressed lists are always polynomial in length. We first observe that probability quantities such as $Pr(pv_i \mid \mathtt{T_j})$ are rational numbers, where both the numerator as well as the denominator are integers bounded by n (i.e., the number of products in the dataset). From Equation 4.5, note that the score of a product involves m such probability quantity multiplications, where m is the number of attributes. Therefore, the score of any product for any single tag can be represented as a rational number, where the numerator and denominator are integers bounded by $O(n^m)$. Thus, we can normalize each such score into an integer by multiplying it with $O(n^m)$.

Next, consider a z'-dimensional cube with each side of length $L = O(n^m)$. We partition the cube into z'-dimensional *cells* as follows: Along each axis, start with the furthest value L, and then proceed towards the origin by marking the points $L/(1+\sigma)$, $L/(1+\sigma)^2$, and so on. The number of points marked along each axis is $\log_{(1+\sigma)} L = O(m \log_{(1+\sigma)} n)$ which is a polynomial in m and n. Then at each marked point we pass $(z'-1)$-dimensional hyperplanes perpendicular to the corresponding axis. Their intersections creates $O(poly(m,n)^{z'})$ cells within cube $L^{z'}$.

Due to this skewed method of partitioning cube into cells, we see that the cells that are further away from the origin are larger. Consider the ith iteration of the PTAS algorithm. Each product in S_i may be represented as a point in this cube. Though within any cell there may be several points corresponding to products of S_i, after compression *there can be at most only one* point corresponding to a product of S'_r, because two or more points could not have survived the compression process. The length of any compressed list in the PTAS algorithm is at most $O(poly(m,n)^{z'})$. When z' is a constant, it leads to an overall polynomial running time for PA. □

Extending from Top-1 to Top-k: Our PA algorithm can be modified to return top-k products instead of just the best product. For the tag group $\mathtt{T_r}$, once a set of products S_i is built, we compress to form the set S'_i. However, every time a cluster representative is selected, instead of deleting all the remaining points in the cluster, we remember $k-1$ products within the cluster and associate them with the cluster representative (and if the cluster has less than k products, we remember and associate all the products with the cluster representative). When all the m iterations are completed, we can return the top-k products as follows: we first return the best product of S'_m along with the $k-1$ products associated with it. If the number of associated products are less than $k-1$, the second best cluster representative of S'_m and the set of products associated with it are returned, and so on. When the approximate top-k products from all tag groups have been returned, the main algorithm returns the overall best top-k products from among them. It can be shown that this approach guarantees an approximation factor for the score of the top-k products returned.

Grouping of Tags: The PA algorithm partitions the set of tags into constant-sized groups. We can employ techniques similar to the grouping of attributes technique for ETT algorithm in order to group related tags together in a principled fashion. However, the bounds and properties of PA algorithm are not affected by this.

4.5 Experiments

We conduct a set of comprehensive experiments using both synthetic and real datasets for quantitative and qualitative analysis of our proposed algorithms. Our quantitative performance indicators are *efficiency* of the proposed exact and approximation algorithm, and *approximation factor* of results produced by the approximation algorithm. The efficiency of our algorithms is measured by the overall execution time

and the number of products that are considered from the pool of all possible products, whereas approximation factor is measured as the ratio of the acquired approximate result score to the actual optimal result score. We also conduct a user study through Amazon Mechanical Turk study as well as write interesting case studies to qualitatively assess the results of our algorithms.

Real Camera Dataset: We crawl a real dataset of 100 cameras[6] listed at Amazon[7]. The products contain technical details (attributes), besides the tags customers associate with each product. The tags are cleaned by domain experts to remove synonyms, unintelligent and undesirable tags such as `nikon coolpix`, `quali`, `bad`, etc. Since the camera information crawled from Amazon lacks well-defined attributes, we look up Google Products[8] to retrieve a rich collection of technical specifications for each product. Each product has 40 boolean attributes, such as self-timer, face-detection, red-eye fix, etc; while the tag dictionary includes 40 unique keywords like `lightweight`, `advanced`, `easy`, etc.

Real Car Dataset: We crawl another real dataset from Yahoo! Autos[9]. We focus on new cars listed for the year 2010 spanning 34 different brands. There are several models for each brand, and each model offers several trims.[10] Since each trim defines a unique attribute-value specification, the total number of trims that we crawl are the 606 products in our dataset. The products contain technical specifications as well as ratings and reviews, which include pros and cons. We parse a total of 60 attributes: 25 numeric, and 35 boolean and categorical (which we generalize to boolean) such as air-conditioning, sunroof, etc . The total number of reviews we extract is 2180. We

[6]As discussed earlier, the number of products in the dataset is not important for the execution cost; analysis in Figure 4.9.

[7]http://www.amazon.com

[8]http://www.google.com/products

[9]http://autos.yahoo.com/

[10]Trims denote different configurations of standard equipment.

extract tags from the reviews using the keyword extraction toolkit AlchemyAPI[11]. We process the text listed under *pros* in each review to identify a set of 15 desirable tags such as `fuel economy`, `comfortable interior` and `stylish exterior`. A car is assigned a tag if one of its reviews contains that keyword.

Synthetic Dataset: We generate a large boolean matrix of dimension 10,000 (products) $\times$ 100 (50 attributes + 50 tags) and randomly choose submatrices of varying sizes, based on our experimental setting. We split the 50 independent and identically distributed attributes into four groups, where the value is set to 1 with probabilities of 0.75, 0.15, 0.10 and 0.05 respectively. For each of the 50 tags, we pre-define relations by randomly picking a set of attributes that are correlated to it. A tag is set to 1 with probability p if majority of the attributes in its pre-defined relation have boolean 1. For example, assume tag T_1 is defined to depend on attributes A_{13}, A_{25} and A_{40}. T_1 is set to 1 with a probability of 0.67 if 2 out of A_{13}, A_{25} and A_{40} are 1.

We use the synthetic datasets for quantitative experiments, while the real data are used in user and case study.

System configuration: Our prototype system is implemented in Java with JDK 5.0. All experiments were conducted on an Windows XP machine with 3.0Ghz Intel Xeon processor and 2GB RAM. The JVM size is set to 512MB. All numbers are obtained as the average over three runs.

4.5.1 Quantitative Evaluation

Exact Algorithm: We first compare the Naive approach with our ETT. Since the Naive algorithm can only work for small problem instances, we a pick a subset from the synthetic dataset having 1000 products, 16 attributes and 8 tags. Figures 4.4

[11] http://www.alchemyapi.com/api/keyword/

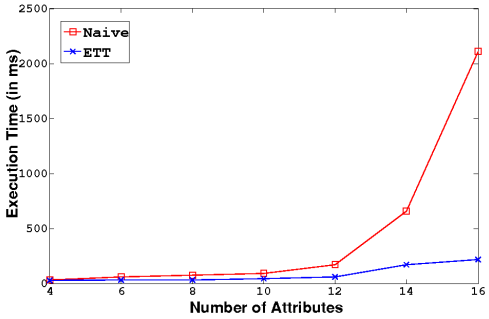


Figure 4.4. Execution time for varying m (Synthetic data).

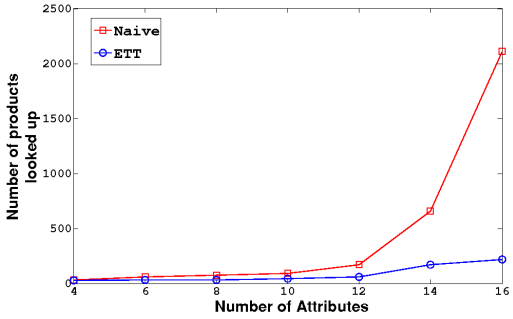


Figure 4.5. Number of products for varying m (Synthetic data).

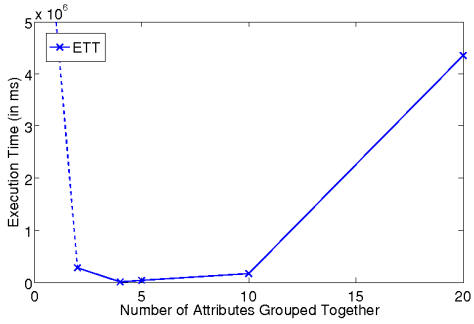


Figure 4.6. Execution time for varying m' (Synthetic data).

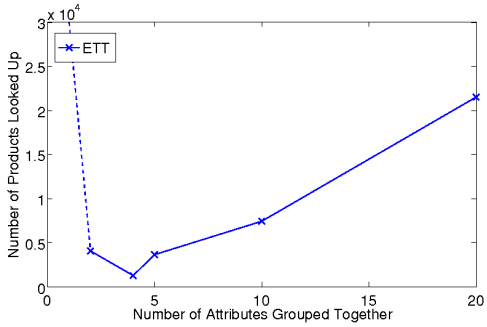


Figure 4.7. Number of products for m' (Synthetic data).

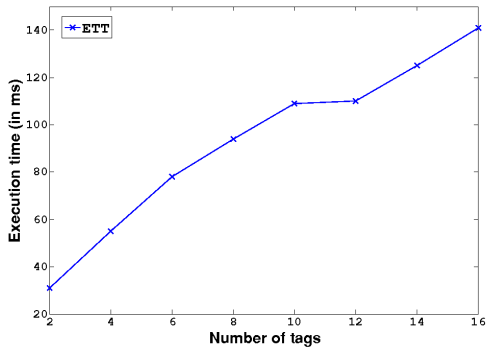


Figure 4.8. Execution time for varying z (Synthetic data).

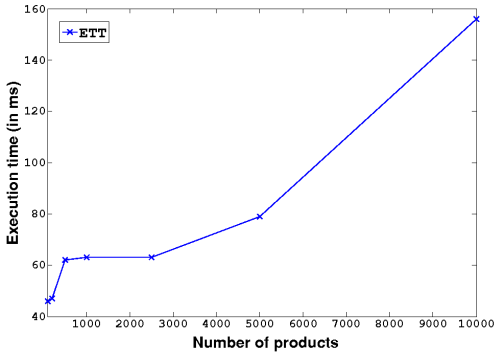


Figure 4.9. Execution time for varying n (Synthetic data).

and 4.5 compare the execution time and the number of candidate products considered, for Naive and ETT respectively, when the number of attributes (m) varies (number of products = 1000, number of tags = 8). Note that the number of products considered by ETT is the number of products created in tier-2 by joining products from tier-1.

The Naive algorithm considers all 2^m products. We used as number of attributes per group $m' = 2, 2, 4, 5, 4, 7, 4, 6$ for $m = 4, 6, 8, 10, 12, 14, 16, 18$ respectively in ETT (more analysis of m' in Figures 4.6 and 4.7). As can be seen, Naive is orders of magnitude slower than ETT.

Next, we study the behavior of attribute groupings on ETT. For a sub-sample picked from our synthetic dataset having 20 attributes, 1000 products and 8 tags, we experiment with different possible attribute groupings, m' = 1, 2, 4, 5, 10, 20. Figures 4.6 and 4.7 shows the effect of m' on the performance of ETT when attributes are grouped arbitrarily. The execution time and number of products considered for $m' = 1$ is not reported in Figures 4.6 and 4.7 as it was too slow. The trade-off of choosing m' is: a small m' means there are many short lists in tier-1, so that the cost of joining the lists is high. In contrast, a large m' indicates fewer but longer lists in tier-1 resulting in increased cost of creating the lists. We observe that the best balance is struck when $m' = 4$ attributes forming 5 lists, each having 2^4=16 products.

We also employed the grouping of attributes technique in Section 4.4.1.1 to partition the set of 20 attributes, and investigate if the execution time and number of products considered improves (i.e., decreases). We create a graph of 20 nodes (corresponding to the 20 attributes) and ${}^{20}C_2 = 190$ edges. We use the absolute value of the Pearson correlation to determine the edge weight because even if two attributes are anti-correlated, they should be grouped together and the 1 from one will be combined with the 0 from the other to create a high-scored entry (we use 0.5 for dont care). Then, we employ a graph partitioning algorithm for partitioning the 20 attributes into as many groups as the desired number of lists. Specifically, we use the publicly available software METIS-4.0[12] for partitioning the attributes. We observe that when we partition the 20 attributes into 4 clusters (i.e., $m' = 5$

[12]http://glaros.dtc.umn.edu/gkhome/views/metis

attributes forming 4 lists), the execution time is $\frac{1}{6}$ the execution time in case of arbitrary grouping of attributes (Figure 4.6); the number of products looked up by ETT decreases from 3639 to 1713 (Note that the number of products looked up by Naive for this data is 1048576). Again, if we partition the 20 attributes into 5 clusters (i.e., $m' = 4$ attributes forming 5 lists), the execution time and the number of products looked up by ETT remains the same as in case of arbitrary grouping of attributes (Figure 4.6). This is because clustering of attributes into 4 groups generated stronger partitioning (i.e., attributes grouped together have stronger correlation) than clustering of attributes into 5 groups. Therefore, if the data is highly correlated and yields well-defined clusters or partitions, ETT benefits significantly by employing principled grouping of attributes.

Next, we vary the number of tags z and number of products n in the dataset to study the behavior of ETT. We pick a subset from the synthetic dataset having 1000 products, 16 attributes and 16 tags, and consider further subsets of this dataset. Figure 4.8 reflects the change in execution time with increasing number of tags for the synthetic data (number of products = 1000, number of attributes = 12, attribute grouping = 3). The increase in number of tags increases the number of GetNext() operations in ETT, and hence the running time rises steadily. Figure 4.9 depicts how an increase in the number of products in the dataset (number of attributes = 12, number of tags = 8, attribute grouping = 3) barely affects the running time of ETT since an initialization step calculates all conditional tag-attribute probabilities.

Approximation Algorithms: We observe in Figure 4.4 that the execution time of ETT outperforms that of Naive, for moderate data instances. However, ETT is extremely slow beyond number of attributes (m) = 16, which makes it unsuitable for large real-world datasets having many attributes and tags. Therefore, we move to our approximation algorithm PA, and compare their execution time and sub-optimal

product score in Table 4.3. We pick three different subsets of 1000 products from the synthetic dataset: (number of attributes = 8, number of tags = 4), (number of attributes = 12, number of tags = 8) and (number of attributes = 16, number of tags = 8). We execute PA algorithm at an approximation factor (0.8). The execution time of PA for a moderately large dataset (1000 products, 16 attributes and 12 tags) indicates that it is unlikely to scale to large (real) datasets. Nevertheless, it is an algorithm which provides worst case guarantees in both time complexity and result quality.

Table 4.3. PA Performance (Synthetic data)

User-Defined Approx Factor(ϵ)	Execution Time (in ms)	Obtained Approx Factor
0.5	406.0	0.93
0.4	749.0	0.94
0.3	2796.0	0.97
0.2	41094.0	0.98

4.5.2 Qualitative Evaluation

4.5.2.1 User Study

We now validate how designers can leverage existing product information to design new products catering different groups of people in a user study conducted on Amazon Mechanical Turk (https://www.mturk.com) on the real camera dataset. We also consult DPreview (http://www.dpreview.com), a website about digital cameras and digital photography. There are two parts to our user study. Each part of the study involves thirty independent single-user tasks. Each task is conducted in two phases: *User Knowledge Phase* where we estimate the users' background and *User Judgment Phase* where we collect users' responses to our questions.

In the first part of our study, we build four new cameras (two digital compact and two digital slr) using our HC algorithm by considering tag sets corresponding to compact cameras and slr cameras respectively. We present these four new cameras along with four existing popular cameras (presented anonymously) and observe that 65% of users choose the new cameras, over the existing ones. For example, users overwhelmingly prefer our new compact digital camera over Nikon Coolpix L22 because the former supports both automatic and manual focus while the latter does not, thus validating how our techniques can benefit designers.

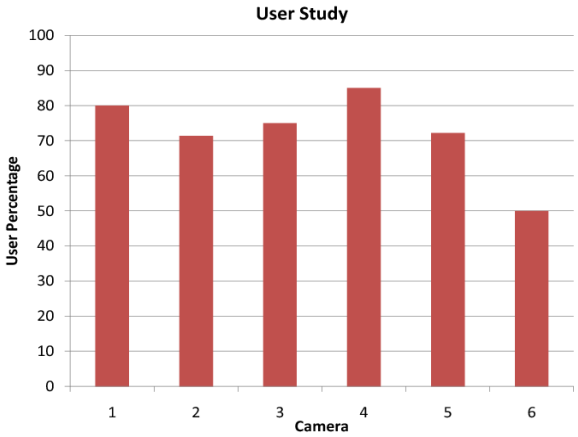


Figure 4.10. Users Classify Cameras Correctly.

The second part of the study concerns six new cameras designed for three groups of people: young students, old retired and professional photographers. Domain experts identify and label three overlapping sets of tags from the camera dataset's complete tag vocabulary, one set for each group and we then build two potential new cameras for each of the three groups. For each of the six new cameras thus built, we ask users to assign at least five tags by looking up the complete camera tag vocabulary, provided to them. We observe that majority of the users rightly classify the six cameras into the three groups. The correctness of the classification is vali-

dated by comparing the tags received for a camera to the three tag sets identified by domain experts; we also validate the correctness by consulting data available in Dpreview. For example, the cameras designed by leveraging tags corresponding to professional photographers draw tags like `advanced`, `high iso`, etc. while cameras designed by leveraging tags corresponding to old retired draw tags like `lightweight`, `easy`, etc. Figure 4.10 shows the percentage of users classifying the six cameras correctly. Thus, our technique can help designers build new products that are likely to attract desirable tags from different groups of people.

4.5.2.2 Case Study

We present few interesting anecdotal results returned by our framework on the real car dataset to validate that our algorithms help us draw interesting conclusions about the desirability of certain car specifications (attribute values). Our HC algorithm indicates that cars having child safety door locks, 4-wheel anti lock brakes, AM/FM radio, keyless entry, telescoping steering wheel, compass, air filter, trunk light, smart coinholder and cup holder, etc. are the features likely to elicit positive feedback from the customers (i.e., features that maximize the set of desirable tags for real car dataset). When we design new cars by considering only those car instances as our training set which have received the tag `economy`, we observe that some luxury features like heated seats, in-dash CD changer system, sunroof/moonroof and leather upholstery are returned by our framework. This indicates that users prefer selective luxury features when buying economy cars. Also, sports cars designed using our algorithm (by considering only those car instances as our training set which have received the tag `sports`) are found to contain safety features, thereby indicating that safety features have become a high priority requirement for users buying sports cars.

CHAPTER 5

Item Snippet Generation

5.1 Introduction

The widespread use and growing popularity of online collaborative content sites has created rich resources for users to consult in order to make purchasing decisions on various items such as e-commerce products, travel movies, restaurants, etc. Collaborative content sites (e.g., Amazon, Yelp, etc.) contain millions of web items (i.e., products available over web). For example, the review site Yelp contains more than 25,000 restaurants listed only for New York. Faced with such overwhelming choices, it is becoming increasingly important for the producers/manufacturers (e.g., owner of a restaurant, Dell - for laptops, Canon - for cameras) or retailers (e.g., Amazon, eBay, Etsy) of such products to help an user quickly discover the products she is interested in from the list of products returned as a result of her search query. A popular technique is to associate each product with a short description that provides the *first impression* of the profuct, i.e., only the necessary and interesting details to help a user in making a decision. Such succinct summarizations of web item descriptions are referred to as *snippets*. Typically, an item snippet only involves a fraction of the item attributes. For example, a laptop that highlights the features 2^{nd} `Gen Intel-R Core i7 processor` and `14″ LED display, 0.9″ thin, 4lbs weight` in its snippet is likely to influence a prospective customer decision in its favor, especially if she is looking for portable and powerful laptop. However, the snippets shown with items today (faceted navigation) are pre-defined and static. A user searching for stylish laptop would not benefit from the above item snippet.

Various websites today (e.g., Amazon for e-commerce products, Flickr for photos, etc.) encourage users to actively participate by assigning *tags* to online resources with a purpose to promote their contents and allow users to share, discover and organize them. We exploit the availability of user feedback in collaborative content sites in the form of tags to *automatically* generate item snippets that are likely to maximize its visibility among users. We perform aggregate analytics over item attributes and tags to identify the *salient features* that were responsible for the positive feedback the item has received so far and that would be highlighted in its snippet. In addition to traditional marketplaces, such snippets can also increase chances of a musical piece becoming popular, an e-book (in which case, the snippet is referred to as *blurb*) receiving many view/download requests, etc.

Web advertising dates back more than ten years, when short text advertisements began to appear among the results of queries to search engines (known as Sponsored Search) and in the content of web pages (known as Content Match and Display Ad). In recent times, computational advertisement has emerged as a new scientific sub-discipline that aims to find the *best ad* to be presented to a user engaged in a given context. There are several research challenges associated with finding the best ad, such as designing the appropriate mathematical optimization model to maximize value for users, advertisers, and publishers; integrating user's past browsing history and behavioral attributes; minimizing expected user effort to comprehend snippets, etc. We focus on the novel aspect of building an item snippet as a succinct summary of its specifications that highlights the features, that were responsible for the positive feedback left by the past users. For users with specific preferences (expressed through search query), the item snippet highlights those necessary and interesting details which best match the user requisite. For example, if a user is looking for an `adventurous` and `budget-friendly` Europe backpacking trip package, a returned

trip snippet must highlight the relevant related features youth-hostel, Eurail Youth Pass and free city-attractions to draw her attention. Intuitively, we discover in the database of trips, those attributes that are responsible for the trip receiving the tags adventurous and budget-friendly by past users.

We refer to this problem as the Informative Snippet Design (ISD) Problem for a single item, that identifies the relevant item attributes to be highlighted in its snippet in order to maximize the probability of the user preference (available in the form of search query) being satisfied. This can be extended to the top-k version where we return the top-k snippets, which can be post-processed by the manufacturers, retailers, etc. to accommodate the more traditional factors. We also envision the utility of dynamic snippet, as opposed to regular pre-defined and static summaries (e.g., in faceted navigation of Amazon and eBay) to match a user's search query effectively. For example, a user looking for stylish digital camera would benefit more from the snippet in the bottom row of Figure 5.1 than that in the general (pre-defined, static) description in the top row of Figure 5.1[1]. Though our work is similar in essence to research dedicated to automatically generating textual summaries of web pages, we tap into the rich resource of collaborative content and perform aggregate analytics to build snippets. Such snippets would highlight the outstanding item features as per the past users, and therefore would also depict a succinct summary of the positive reviews.

The ISD problem intends to identify the relevance of a snippet to a user search query. In this work, we consider the following three categories of conjunctive queries[2]: (i) General-purpose queries - User search queries that do not express specific user preferences (e.g., laptop, digital camera, cellphone, etc.). For this type of queries, the

[1]Our snippets are not influenced by, or biased towards, any brand.

[2]However, our framework (i.e., function) can be modified to handle other complex queries.

Figure 5.1. Top: Pre-defined and static snippet for camera, Bottom: Informative snippet for camera for query stylish digital camera.

ISD goal is to maximize the probability of an item snippet receiving all positive tags that existing items have received in the past.

(ii) Tag-driven queries - User search queries that express a user's preference by short keywords or tags (e.g., lightweight laptop, modern cellphone). For this type of queries, the ISD goal is to maximize the probability of an item snippet receiving the user-specified tags (and its synonymns).

(iii) Attribute-driven queries - User search queries that express a user's preference by attribute values (e.g., SLR camera, 3g cellphone). For this type of queries, the ISD goal is to maximize the probability of an item snippet receiving all positive tags that existing items (in the same category) with similar attribute values have received so far.

Henceforth, we refer to the set of tags associated with the user search query as the *desirable* tags. Note that, in addition to a product's technical specifications, several other implicit factors such as product quality and utility, user behavior, etc. influence tagging behavior. We refer to related literature [5] and choose to focus on content-based tagging feedback to identify the salient features of an item.

The list of products returned as a result of user search query are often very similar to each other, and hence would have similar snippets generated. Therefore, it is necessary to *diversify* the snippets associated with the returned products in order to increase the chances of an user liking any one of the top returned products. Extracting a set of diverse features, that covers the various aspects of the underlying dataset, is a problem of automated facet discovery which is known to improve user experience in the web. While faceted search employed by sites (e.g., Amazon, eBay) performs pre-defined top-down navigation on the concept hierarchy, where all features of the currently selected concept are displayed, our objective is to highlight the important features as well as diverse features. In this chapter, we study the problem of Diversified Informative Snippets Design (DISD) Problem for a list of products returned by a search query, to find snippets that highlight the most relevant and the most diverse features. Our notion of diversity is based on exclusivity and coverage since our goal is to help a user quickly choose from several similar items returned as a result of her search query. For example, a user looking for travel-purpose point-and-shoot digital camera would benefit more from the diversified snippets in the bottom row of Figure 5.2 for Cameras 1 and 2, than the snippets in the top row of Figure 5.2. This specifically distinguishes our work from the Google Webmaster Tool for generating *rich snippets* (containing meta-description, average rating, photos, etc.) for webpages. The Google tool creates a snippet per webpage (i.e., item) and does not consider diversification of the snippets, as we do to maximize visibility. We measure diversity as a function of exclusivity and coverage of attributes in the snippets of items, while ensuring that the snippet selected for each of the items has a score close to the best possible snippet score for that item. In other words, we aim to find snippets that highlight the most relevant and the most diverse features. Note that, the focus of this problem is not to advocate one particular diversity measure over another. Rather,

Figure 5.2. Top: Informative snippets, Bottom: Diversified Informative snippets for Cameras 1 and 2 returned for query travel-purpose digital camera.

we focus on formalizing the problem and and proposing algorithmic solutions that benefit from the triangular inequality satisfiability of distance/diversity metric. Our framework can be extended to handle other definitions of diversity.

Solving the informative snippet design problem is technically challenging. Complex dependencies exist among tags and item attributes. Additionally, the task of finding the best set of attributes maximizing the probability of an item snippet receiving all desirable tags requires us to exhaustively evaluate an exponential number of combinations. A general Bayesian Network model (where item attributes and tags would form nodes and edges would represent conditional dependencies between a pair of nodes) to identify the best snippet for an item would be computationally expensive. In this chapter, we consider the very popular Naive Bayes Classifier with the simplistic conditional independence assumption for tag and attribute modeling because of its success in [5]. We analyze its computational complexity when Naive

Bayes classifier is used for finding the probability of an item snippet drawing tag, introduce the idea of *composite tag*, and propose efficient algorithmic solutions that are shown to work well in practice. Our first algorithm is a exact top-k algorithm that performs significantly better than the naive brute-force algorithm. We formulate the ISD problem of retrieving the top-k snippets for an item as *subset sum* combinatorial optimization problem [39] and propose a novel approximation algorithm that runs in worst case polynomial time, and guarantees a provable bound on the approximation factor in snippet quality.

Diversification of search has been studied in recent times in several contexts with many different approaches, majority of which focuses on a scoring function that takes both query relevance and diversity into consideration [40][41]. In this chapter, we emphasize word sense diversification in the snippets for diversity and measure categorical distance, while ensuring that the selected snippet's score is not much less than the best snippet score for that item. Our DISD problem is conspicuously different from diversity aware search: diversity aware search aims to find the top-k relevant items from the set of all n relevant items returned as a result of search query; the DISD problem aims to find a result (i.e., snippet) for each of the n relevant items, where each item has a set of top-k results to choose from. The first of our algorithmic solutions for DISD is a novel exact top-k algorithm based on non-trivial adaptation of top-k query processing technique in [40][42]. We also propose an approximation algorithm that is based on techniques employed in computational geometry for the *facility dispersion problem* [43] and has theoretical bounds on quality. We experiment with both synthetic and real data crawled from the web to demonstrate the effectiveness of our algorithms for the problems and conduct user studies to validate that our item snippets are helpful to draw user attention, especially when several competing items are available as a result of a query.

5.2 Problem Framework

Once again, we model a collaborative tagging site $\mathcal{D}$ as a triple $\langle \mathcal{U}, \mathcal{I}, \mathcal{R} \rangle$, representing the set of reviewers (i.e., users), the set of items (i.e., products) and the feedback (i.e, tag vocabulary), respectively. Let $\{p_1, p_2, ..., p_{n_p}\}$ be a collection of n_p products, where each entry is defined over the item schema $\mathcal{I}_\mathcal{A} = \langle pa_1, pa_2, ..., pa_{m_p} \rangle$ and the tag dictionary space $\mathtt{T} = \{\mathtt{T_1}, \mathtt{T_2}, ..., \mathtt{T_{n_c}}\}$. For the ease of representation (and since we do not consider user details in the framework), let m_p be m. A product p is a set of attribute values, $p = \langle pv_1, pv_2, ..., \rangle$ where each pv_y is a value for the attribute $pa_y \in \mathcal{I}_\mathcal{A}$. Each attribute pa_y can take one of several values pv_y from a multi-valued categorical domain D_i, or one of two values 0, 1 if a boolean dataset is considered. A tag $\mathtt{T_j}$ is a bit where a 0 implies the absence of a tag and a 1 implies the presence of a tag for product p. Each product is thus a vector of size $(m + n_c)$, where the first m positions correspond to a vector of attribute values, and the remaining n_c positions correspond to a boolean vector.[3,4]

Consider a query which picks a set of desirable tags $T^d = \{\mathtt{T_1}, ..., \mathtt{T_z}\} \subseteq \mathtt{T}$. The objective of ISD problem is to determine s of m attributes for building the snippet S_p of a product p, such that the probability of attracting all desirable tags $\mathtt{T_j} \in T^d$ is maximized. The top-k snippets of product p are represented as $S_p^1, S_p^2, \ldots, S_p^k$.

Given a training set as the dataset described above, we build Naive Bayes Classifier (NBC), that classify tags given attributes (one classifier per tag). defines the probability that a snippet S_p is annotated by tag $\mathtt{T_j}$. If $\{pv_1, pv_2, ..., pv_s\}$ are the attribute values in S_p, the classifier for tag $\mathtt{T_j}$ defines the probability that snippet S_p of product p draws tag $\mathtt{T_j}$, as:

[3]Our framework allows numeric attributes, but as is common with Naive Bayes Classifiers, we assume that they have been appropriately binned into discrete ranges.

[4]A more complex framework which leverages the frequencies of tags is left for future work.

$$
\begin{aligned}
Pr(\mathtt{T_j} \mid S_p) &= Pr(\mathtt{T_j} \mid pv_1, pv_2, ..., pv_s) \\
&= \frac{Pr(\mathtt{T_j}).\Pi_{i=1}^{s} Pr(pv_i \mid \mathtt{T_j})}{Pr(pv_1, pv_2, ..., pv_s)} \qquad (5.1)
\end{aligned}
$$

$$
Pr(\mathtt{T_j}' \mid S_p) = \frac{Pr(\mathtt{T_j}').\Pi_{\mathtt{i=1}}^{\mathtt{s}} \mathtt{Pr}(pv_i \mid \mathtt{T_j}')}{Pr(pv_1, pv_2, ..., pv_s)} \qquad (5.2)
$$

Since $Pr(\mathtt{T_j} \mid S_p) + \mathtt{Pr}(\mathtt{T_j}' \mid S_p) = \mathtt{1}$, from Equations 5.1, 5.2:

$$
\begin{aligned}
Pr(pv_1, pv_2, ..., pv_s) &= Pr(\mathtt{T_j}).\Pi_{i=1}^{s} Pr(pv_i \mid \mathtt{T_j}) + \\
&\quad Pr(\mathtt{T_j}').\Pi_{\mathtt{i=1}}^{s} \mathtt{Pr}(pv_i \mid \mathtt{T_j}') \qquad (5.3)
\end{aligned}
$$

From Equations 5.1, 5.3:

$$
\begin{aligned}
Pr(\mathtt{T_j} \mid S_p) &= Pr(\mathtt{T_j} \mid pv_1, pv_2, ..., pv_s) \\
&= \frac{Pr(\mathtt{T_j}).\Pi_{\mathtt{i=1}}^{s} \mathtt{Pr}(pv_i \mid \mathtt{T_j})}{Pr(\mathtt{T_j}).\Pi_{\mathtt{i=1}}^{s} \mathtt{Pr}(pv_i \mid \mathtt{T_j}) + \mathtt{Pr}(\mathtt{T_j}').\Pi_{\mathtt{i=1}}^{s} \mathtt{Pr}(pv_i \mid \mathtt{T_j}')} \\
&= \frac{1}{1 + \frac{Pr(\mathtt{T_j}')}{Pr(\mathtt{T_j})} \Pi_{i=1}^{s} \frac{Pr(pv_i \mid \mathtt{T_j}')}{Pr(pv_i \mid \mathtt{T_j})}} \qquad (5.4)
\end{aligned}
$$

The probability of snippet S_p of an item p drawing all desirable tags $T^d = \{T_1, ..., T_z\} \subseteq \mathtt{T}$ is:

$$
\begin{aligned}
f(S_p, T^d) &= Pr(\mathtt{T_1}, ..., \mathtt{T_z} \mid S_p) \\
&= Pr(\mathtt{T_1} \mid S_p)....Pr(\mathtt{T_z} \mid S_p) \\
&= \Pi_{j=1}^{z} \frac{1}{1 + \frac{Pr(\mathtt{T_j}')}{Pr(\mathtt{T_j})} \Pi_{i=1}^{s} \frac{Pr(pv_i \mid \mathtt{T_j}')}{Pr(pv_i \mid \mathtt{T_j})}} \qquad (5.5)
\end{aligned}
$$

Note that the task of finding the best snippet that maximizes Equation 5.5 is difficult for boolean data and $k = 1$, and it follows from [5]. The best item snippet that maximizes $\mathtt{T_1}$ may not be the same as the one that maximizes $\mathtt{T_2}$, and so on. Hence, we introduce the idea of composite tag.

Composite Tag: Composite tag is a single tag T that consists of the collection of desirable tags in T^d. The consideration of composite tag alleviates the computational challenges associated with the ISD problem for $k = 1$ (the ISD problem is NP-Complete for $k > 2$, as shown in Section 5.3), since from Equations 5.4 and 5.5, we have:

$$\begin{aligned} f(S_p, T) &= f(S_p, T^d) \\ &= Pr(\mathtt{T_1}, ..., \mathtt{T_z} \mid S_p) \\ &= Pr(T \mid S_p) \\ &= \frac{1}{1 + \frac{Pr(T')}{Pr(T)} \Pi_{i=1}^{s} \frac{Pr(pv_i|T')}{Pr(pv_i|T)}} \end{aligned} \tag{5.6}$$

If there are sufficient instances in the training dataset that have $T^d = \{\mathtt{T_1}, ..., \mathtt{T_z}\} \subseteq T$, we can compute probabilities of the form $Pr(pv_i \mid T), Pr(pv_i \mid T')$, $i = 1(1)m$. If the number of instances is insufficient, we compute the probabilities by considering conditional independence in the following way:

$$\begin{aligned} Pr(pv_i \mid T) &= Pr(T \mid pv_i).\frac{Pr(pv_i)}{Pr(T)} \\ &= Pr(T_1, T_2, \ldots, T_z \mid pv_i).\frac{Pr(pv_i)}{Pr(T)} \\ &= Pr(T_1 \mid pv_i).Pr(T_2 \mid pv_i) \ldots Pr(T_z \mid pv_i).\frac{Pr(pv_i)}{Pr(T)} \end{aligned}$$

Quantities of the form $Pr(pv_i \mid T')$ are difficult to resolve as they cannot be reduced using the conditional independence assumption. However, since we know that $Pr(T)$ is small, $Pr(T')$ is large ≈ 1. Therefore, we approximately estimate $Pr(pv_i \mid T')$ by computing $Pr(pv_i \mid \mathcal{I})$. The consideration of composite tag eliminates some of the computational challenges associated with the problem (details in Section 5.3).

We are now ready to formally define the two main problems considered in this work.

Problem 5.2.1. *Informative Snippet Design (ISD) Problem:* Given a user search query expressed as a composite tag T (i.e., the set of desirable tags T^d) and an item p from a dataset $\mathcal{D}$ of tagged items $\{p_1, p_2, ..., p_{n_p}\}$, design k snippets $S_p^1, S_p^2, \ldots, S_p^k$ of size s for p that have the highest score of receiving all desirable tags, given by Equation 5.6.

Problem 5.2.2. *Diversified Informative Snippet Design (DISD) Problem:* Given a list of n items $\{p_1, p_2, \ldots, p_n\}$ returned by search engine from a dataset $\mathcal{D}$ of n_p items for user search query T, and the top-k snippets $\{\{S_{p_1}^1, S_{p_1}^2, \ldots, S_{p_1}^k\}, \{S_{p_2}^1, S_{p_2}^2, \ldots, S_{p_2}^k\}, ..., \{S_{p_n}^1, S_{p_n}^2, \ldots, S_{p_n}^k\}\}$ of size s for each of the n items, determine n snippets $S_{p_1}, S_{p_2}, \ldots, S_{p_n}$ for the n items respectively such that:

- $S_{p_i} \epsilon \{S_{p_i}^1, S_{p_i}^2, \ldots, S_{p_1}^k\}, \forall i = 1(1)n$
- diversity$(S_{p_x}, S_{p_y}) \geq \tau, \forall (p_x, p_y) \in \{p_1, \ldots, p_n\}$
- sum $(f(S_{p_1}, T), \ldots, f(S_{p_n}, \mathrm{T}))$ is maximized

For the rest of the chapter, we consider boolean attributes and explain our algorithms in a boolean framework; they can be readily generalized to handle other data types.

5.3 Complexity Analysis

In this section, we discuss the computational complexity of our ISD and DISD problems

ISD Problem Computational Complexity: The ISD Problem objective is to identify the top-k snippets of size s (we select s out of m attributes) which maximizes the scoring function in Equation 5.6. The scoring function involves the quantity $\Pi_{i=1}^{s} \frac{Pr(pv_i|T')}{Pr(pv_i|T)}$, which can be expressed as a (logarithmic) sum, $\Sigma_{i=1}^{s} \log \frac{Pr(pv_i|T')}{Pr(pv_i|T)}$. Therefore, the prob-

lem can be expressed as s-SUM Problem, which is a parameterized version of the well known combinatorial optimization problem SUBSET SUM and aims to find a subset of size s from input set $X = \{x_1, x_2, \ldots, x_n\}$ such that $\Sigma_{i=1}^{s} x_i \leq t$. The target sum t is the score of the best size-s subset, which is trivial to obtain. The s-SUM problem is fundamentally connected to several NP-hard problems and is proved to be W[1]-hard [44], and it has been shown to be solved in $O(n^{\lceil \frac{k}{2} \rceil})$ time. Our ISD problem, when formulated as s-SUM problem would return the top-1 subset, and we extend this idea to develop our approximation algorithm in Section 5.4.1.2.

DISD Problem Computational Complexity: The DISD Problem objective is to identify the best combination of snippets for n items, where each item has a set of top-k snippets to choose from, such that the total score (i.e., relevance to query) of chosen snippets is maximized, subject to diversity constraints being satisfied. In other words, if there are n subsets having k entries in each so that there are nk entries in total, the goal is to choose n out of nk entries, so that no two of the n results belong to the same subset, subject to relevance and diversity optimization. This problem can be expressed as the Facility Dispersion Problem in computational geometry literature, where the task is choose p out of n facilities, so as to maximize some function of the distances between facilities. Because of the consideration of both relevance and diversity in the optimization task, we consider the variant MAXSUMDISPERSION [45] having the objective of maximizing the sum of all pairwise distances (measuring relevance and dissimilarity) between the selected faciliities. The problem is proved to be NP-Hard, and there exists a 2-approximation algorithm for the problem if the distance measure is a metric. Our DISD problem can be formulated as the MAXSUMDISPERSION problem, and we extend the idea to develop our approximation algorithm in Section 5.4.1.2.

5.4 Algorithms

5.4.1 ISD Algorithms

In this section, we propose efficient algorithms for solving the Informative Snippet Design (ISD) problem.

A brute-force exhaustive approach (henceforth, referred to as Naive-ISD) to solve the problem requires us to design all ${}^{m}C_{s}$ possible snippets S_p^1, S_p^2, ..., $S_p^{{}^{m}C_{s}}$ for item p and composite tag T, and compute $f(S_p, T)$ for each possible snippet in order to identify the top-k snippets. If the snippet size s and the total number of attributes m are small, Naive-ISD is capable of returning the top-k results in reasonable amount of time. However, since m and s are usually large in real data, we develop efficient and practical algorithms that work well in practice. Note that, the number of items in the dataset is not important for the execution cost, since an initialization step can calculate all the conditional tag-attribute probabilities by a single scan of the dataset.

5.4.1.1 Exact-ISD : E-ISD

Our first algorithm is an exact top-k technique, Exact-ISD (E-ISD) based on an interesting adaptation of Fagin's Threshold Algorithm (TA) [32].

We create s identical lists $\mathcal{L} = \{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_s\}$ for identifying snippets $\{S_p^1, S_p^2, \ldots, S_p^k\}$ of size s for item p where each list $\mathcal{L}_i$ contains m values of the form $\sqrt[s]{\frac{Pr(T)}{Pr(T')}} \cdot \frac{Pr(pv_i|T)}{Pr(pv_i|T')}$ corresponding to the m attributes in descending order of magnitude. The sorting is done on the contributions made by attributes to maximize the scoring function in Equation 5.6. The lists are accessed in round robin fashion and for every combination of attributes from the lists, we join them to build a snippet. A join is considered to be *valid* if the number of distinct attributes in the join is equal to s and the join combination (without considering the order of attributes participating in join) is not

already included in the result set. The complete score of the valid join (i.e., the snippet) is resolved by Equation 5.6. We maintain a buffer of size k, called $top-k$ buffer, in order to store the k best snippets $\{\{S_p^1, S_p^2, \ldots, S_p^k\}$ for item p. A snippet is stored in the $top-k$ buffer if its score is higher than the MPFS (Maximum Possible Future Score) at a point, which is the upper bound on the score of an unseen snippet. MPFS is computed using the currently indexed entry of a list and top $(s-1)$ entries of any one of the lists, since they are identical.

$$MPFS = \frac{1}{1+(c.h_1.h_2.\ldots.h_{s-1})} \tag{5.7}$$

where, c is the score of the currently indexed entry and h_1 to h_{s-1} are the scores of the top $(s-1)$ entries from any list. Algorithm 10 is the pseudo-code of our E-ISD algorithm.

Algorithm 10 E-ISD (Naive Bayes probabilities, p, s, k): $S_p^1, S_p^2, \ldots, S_p^k$

Create identical lists $\mathcal{L} = \{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_s\}$ *for* p *where each list contains* m *values of the form* $\sqrt[s]{\frac{Pr(T)}{Pr(T')}} \cdot \frac{Pr(pv_i|T)}{Pr(pv_i|T')}$ *in descending order of magnitude*

//Main Algorithm

```
1: Top-k-Buffer ← ∅
2: while Top-k-Buffer.size() ≠ k do
3:    Retrieve next snippet S in round robin manner by Join(L_1, L_2, ..., L_s)
4:    if VALID JOIN: S.size()=s and S∉Top-k-Buffer then
5:       Add entry {S, Score(S)}to CandidateList                    //Equation 5.6
6:       Calculate MPFS                                             //Equation 5.7
7:       if CandidateList has entry with Score≥MPFS then
8:          Add entry to Top-K Buffer
9:          Remove entry from CandidateList
10:      end if
11:   end if
12: end while
13: return Top-k-Buffer
```

Example 5.4.1. *Consider a boolean dataset of* 10 *items where each item has* 6 *attributes and* 2 *tags, as shown in Table 5.4.1.1. Our objective is to find the top-*3 *snippets of size* 3 *(i.e.,* $s = 3$*,* $k = 3$*) for item with ID=*4*,* O_4 *in Table 5.4.1.1. We create* 3 *identical lists* $\mathcal{L}_1$*,* $\mathcal{L}_2$ *and* $\mathcal{L}_3$ *of size* 6*, where each list is sorted in decreasing order of* $\sqrt[3]{\frac{Pr(T)}{Pr(T')}} \cdot \frac{Pr(pv_i|T)}{Pr(pv_i|T')}$*, where* T *is the composite tag and* $pv_i \in O_4$*. At each iteration, entries from the lists are accessed in round robin fashion. The first four joins are* (A_3, A_3, A_3)*,* (A_5, A_3, A_3)*,* (A_5, A_5, A_3)*,* (A_5, A_5, A_5)*,* (A_4, A_3, A_3) *are all invalid joins. The first valid snippet/join is* (A_3, A_4, A_5) *for which E-ISD computes Score and MPFS. Since score of* $(A_3 = 1, A_4 = 1, A_5 = 1) \geq$ *MPFS, as shown in Figure 5.3, it is stored in the Top-K Buffer. We next skip several invalid joins and obtain the second valid join* (A_3, A_5, A_6)*, compare its Score with MPFS and decide to store it in the Top-K Buffer. Thus, the top-*2 *snippets are obtained. The third and fourth valid joins do not satisfy the condition to be stored in the Top-K Buffer, as we see in Figure 5.3. The fifth valid join* (A_1, A_3, A_5) *again satisfies the threshold condition and is passed on to the Top-K Buffer. Thus we obtain our top-3 snippets by looking up only* 5 *snippets, without having to explore all* ${}^6C_3 = 20$ *snippets.* □

Table 5.1. Example Tagged Item Dataset

Attribute							**Tag**	
ID	$A1$	$A2$	$A3$	$A4$	$A5$	$A6$	$T1$	$T2$
1	0	0	0	1	0	0	0	0
2	0	1	0	0	0	1	0	1
3	0	1	0	1	0	0	0	0
4	0	1	1	1	1	0	1	1
5	1	0	0	0	1	0	1	0
6	1	0	0	1	0	1	0	1
7	1	0	1	1	1	1	1	1
8	1	1	0	1	0	1	0	1
9	1	0	1	0	1	1	1	0
10	1	0	1	1	0	1	0	1

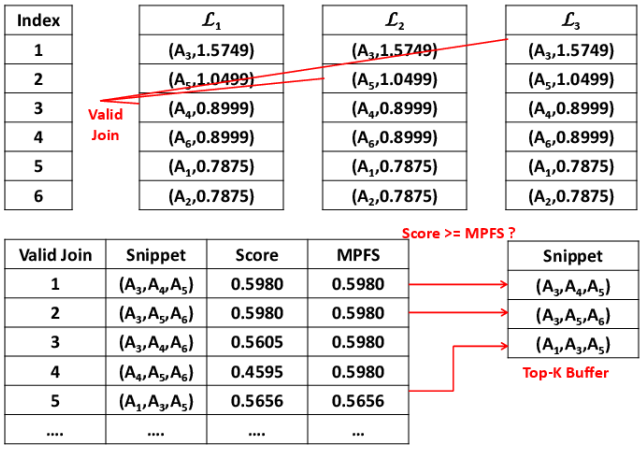


Figure 5.3. Exact-ISD Algorithm for Item ID 4 in Table 5.4.1.1 .

5.4.1.2 Approximation-ISD: A-ISD

The exact algorithm of Section 5.4.1.1 has exponential time complexity in the worst case. Therefore, for larger problem instances (i.e., larger number of attributes and snippet size), we propose an approximation algorithm, A-ISD that provides guarantee in the quality of results as well as run time.

We model the ISD problem as the subset sum problem which , given a set of integers and an integer target, finds out if there is a non-empty subset that sums to the target. An instance of the subset sum problem is a pair $(X,\ t)$, where $X = \{x_1, x_2, \dots\}$ is a set of positive integers and t (known as the target) is a positive integer. The decision version of the subset sum problem asks for a subset Y of X whose sum is as large as possible, but not larger than t; the optimization version returns the subset Y of X, if it exists. In our problem, the input set X, consists of the quantities $\frac{Pr(pv_i|T)}{Pr(pv_i|T')}$ for item p, i=1(1)m, $pv_i = \{0,1\} \in o$ and $T \in \{0,1\}$; pv_i values are dependent on p whose snippet is being built and T values are dependent on the user query. The target sum t is the maximum logarithmic sum of the s highest

quantities in X, and the pv_i-s corresponding to these s quantities in our subset Y_1 gives snippet S_p^1 for p. Besides X and t, we have two additional parameters - snippet size, s (i.e., number of quantities summing up to t) and the k of top-k (i.e., number of subsets/snippets to be returned). Note that the quantities in X are real values and not integers, which do not come in way of adapting the PTAS algorithm for traditional subset sum to our problem. The second best snippet S_p^2 looks for a subset Y_2 of size s from X whose sum is as large as possible, but not larger than t. In this way, we find $(k-1)$ size-s subsets $Y_2, Y_3, \ldots, Y_k$ (i.e., snippets $S_p^2, S_p^3, \ldots, S_p^k$) less than target t, where $Y_2, Y_3, \ldots, Y_k$ are sorted in decreasing logarithmic sum of quantities in the respective subsets.

The exponential algorithm for solving the subset sum takes input set X and target value t and iteratively computes sorted list L_i, the list of sums of all subsets of $\{x_1, ..., x_i\}$ that do not exceed t by *merging* L_{i-1} with $\{L_{i-1} + x_i\}$. The algorithm returns the the maximum value in L_n, and is exponential since L_i can have as many as 2^i items. The polynomial-time approximation scheme (PTAS) algorithm [39] for subset sum *trims* (i.e., clusters) each list L_i by removing as many elements as possible such that every element that is removed is approximated by some remaining element; and then calls the merge operation. The trimming factor δ is a function of the user-defined approximation factor ϵ and the number of items in X, and equals $\epsilon/2\times$(number of items in X). The solution returned is proved to be within a factor of $(1 + \epsilon)$ of the optimum and running time is polynomial in both number of items in X and ϵ. We adapt this idea to solve ISD problem having two additional parameters - s and k - in the following way:

- Instead of maintaining a single list L_i in each iteration, we maintain s lists, where a list L_i^j $(1 \leq i \leq m,\ 1 \leq j \leq s)$ stores all subset sums till i^{th} iteration comprising of j quantities from X. Therefore, we merge L_i^j with $\{L_i^{j-1} + x_i\}$ to

obtain L_{i+1}^j and we employ s trim operations with compression factor $\delta' = \frac{\delta}{s} = \frac{\epsilon}{2ms}$.

- During trim operation, instead of storing one quantity that represents all the remaining quantities that it approximates, we store upto $(k-1)$ quantities. Therefore, the $(k-1)$ members belonging to the final compression cluster in L_m^s would be the size-s subsets $Y_2, Y_3, \ldots, Y_k$ (i.e., snippets $S_p^2, S_p^3, \ldots, S_p^k$ respectively).

Algorithm 11 is the pseudo-code of our A-ISD algorithm; and Theorem 5.4.1 discusses its theoretical properties.

Theorem 5.4.1. *A-ISD is polynomial time approximation scheme for the ISD problem.*

Proof : First, we examine the approximation quality of results returned by A-ISD. When a list L_i^j is trimmed, we introduce a relative error of at most $(1+\delta/s)$ between the representative quantities remaining and the members of the list; the aggregated relative error is therefore $(1+\delta)$. If y_{opt} is the top-1 solution to the exact ISD problem, then there exists a top-1 solution y_{approx} using A-ISD such that:

$y_{approx} \leq y_{opt} \leq y_{approx}.(1+\delta)^m$

Now, $(1+\delta)^m = (1+\epsilon/2m)^m \leq (1+\epsilon)$.

Thus, the top-1 result is guaranteed to lie within a factor of $(1+\epsilon)$ of the optimal top-1. A-ISD retrieves the $(k-1)$ members belonging to the final compression cluster in L_m^s, all of which are within a factor of $(1+\epsilon)$ of the optimal, as the final solution. The guarantee that the final compression cluster in L_m^s would have at least $(k-1)$ subsets to be returned as the final solution follows from the following lemma, the proof of which is skipped for space constraints.

Lemma 5.4.1. *If the trim operation in $(i-1)^{th}$ iteration maintains k quantities during a compression, the trim operation in i^{th} iteration would store at least k quantities.*

Algorithm 11 A-ISD (Naive Bayes probabilities, p, s, k, ϵ): $S_p^1, S_p^2, \ldots, S_p^k$

Build set X of m quantities of the form $\frac{Pr(pv_i|T)}{Pr(pv_i|T')}$ for p and determine target t by computing best snippet S_p^1 for p

//Main Algorithm

1: $\{L_0^0, L_0^1, \ldots, L_0^s\} \leftarrow \langle 0 \rangle$
2: **for** $i = 1 \rightarrow m$ **do**
3: **for** $j = 1 \rightarrow s$ **do**
4: $L_i^j \leftarrow \text{Merge}(L_{i-1}^j, \{L_{i-1}^{j-1} + x_i\})$
5: **end for**
6: **for** $j = 1 \rightarrow s$ **do**
7: $L_i^j \leftarrow \text{Trim}(L_i^j, \frac{\epsilon}{2ms}, k)$
8: Remove from L_i^j every quantity greater than t
9: **end for**
10: **end for**
11: **return** Top-k-Buffer

//Method $Trim(L_i^j, \delta', k)$: L'

1: $\eta \leftarrow |L_i^j|$
2: $L' \leftarrow \langle x_1 \rangle$
3: last $\leftarrow x_1$
4: **for** $l = 2 \rightarrow \eta$ **do**
5: **if** last $> (1 + \delta')x_l$ **then**
6: Append x_l onto end of L'
7: last $\leftarrow x_l$
8: **else**
9: Store upto $(k-1)$ quantities x_l with *last* in L'
10: **end if**
11: **end for**
12: **return** L'

Let us now look at the size of L_i^j to analyze running time. After trimming, successive representative quantities in L_i^j must differ by at least $(1 - \epsilon/2ms)$, and each representative quantities maintains at most $(k-1)$ remaining quantities that it approximates. Therefore, the total number of quantities in L_i^j is at most $k.\log_{(1-\epsilon/2ms)}\text{t}$. This value is polynomial in size of the input (i.e., logt plus some polynomial in m and s) and in $1/\epsilon$.

Since A-ISD produces a solution that is within a factor $(1+\epsilon)$ of optimal and it's running time is polynomial in size of the input, A-ISD is PTAS. □

Example 5.4.2. *Let us consider the dataset in Table 5.4.1.1 of 10 items having 6 attributes and 2 tags. Suppose, we want to identify the top-3 snippets of size (i.e., $s = 3, k = 3$) for item with ID=4. First, we build set X = {1.25, 1.25, 2.5, 1.4286, 1.6667, 1.4286} for attributes $A_1, A_2, \ldots, A_6$, and sort the values in X. Considering the top-3 entries in X, we get $S_p^1 = (A_3, A_4, A_5)$ and the subset sum determines target t = 5.5953. We call A-ISD with ϵ = 0.01, which would return S_p^2 and S_p^3. In the first iteration, we have $L_1^1 = \{1.25^1\}$; L_1^2 and L_1^3 are empty. We trim L_1^1 and it remains the same. In the second iteration, we have $L_2^1 = \{1.25, 1.25\}$, $L_2^2 = \{2.5\}$; L_2^3 is empty. We trim L_2^1 to obtain $L_2^1 = \{1.25\}$; L_2^2 remains the same after trimming. We iterate four more times so that in the final(sixth) iteration, we have: $L_6^1 = \{1.25, 1.4286, 1.6667, \ldots\}$, $L_6^2 = \{2.5, 2.6786, 2.8572, \ldots\}$, $L_6^3 = \{4.1072, \ldots, 5.1786, 5.4167\}$. We look up L_6^3 and identify the top-2 subset sums: 5.1786 and 5.4167. The snippets corresponding to these two scores are: $S_p^2 = (A_1, A_3, A_5)$, $S_p^3 = (A_3, A_4, A_6)$. Thus, we determine the top-3 snippets using A-ISD which lie within 0.01 factor of the optimal; we observe that two of the best three snippets match the results returned by E-ISD.* □

5.4.2 DISD Algorithms

In this section, we propose efficient algorithms for solving the Diversified Informative Snippet Design (DISD) problem.

The objective of DISD is to diversify the snippets of items returned as a result of search query in order to maximize the chances of a user liking at least one of the top items. Similar to related research on diversity aware search, we intend to determine the item snippets based on both their relevance to the search query as well as their

dissimilarity to the other selected snippets. We emphasize word sense diversification in the snippets for diversity and measure diversity as categorical distance, based on the Hamming metric. The relevance score of a snippet is computed by Equation 5.6.

Diversity: Given attribute set $A = \{A_1, A_2, \ldots, A_m\}$ where each attribute A_i can take one of several values a_i from a multi-valued categorical domain D_i, or one of two values {0, 1} if a boolean dataset is considered, we build feature (i.e., description) vectors $\vec{d}$ of length $n_d = \Sigma_{i=1}^{m} D_i$, where values in $\vec{d}$ are set to 1 or 0 depending on the snippet under consideration. The diversity between snippets S_{p_x} and S_{p_y} of products p_x and p_y having description vectors d_{p_x} and d_{p_y} is:

$$diversity(S_{p_x}, S_{p_y}) = \Sigma_{j=1}^{n_d}(d_{p_x}[j] \neq d_{p_y}[j]) \tag{5.8}$$

where j is the vector index and $d_{p_x}[j] \neq d_{p_y}[j]$ is measured as 1, $\forall j$. For example, $\{A_1 = \mathsf{make}\}$, which takes values from $\{\mathsf{Nikon}, \mathsf{Canon}\}$, $\{A_2 = \mathsf{color}\}$, which takes values from $\{\mathsf{black}, \mathsf{white}, \mathsf{red}\}$, and $\{A_3 = \mathtt{type}\}$, which takes values from $\{\mathsf{compact}, \mathsf{SLR}\}$, the feature vectors would be of length $n_d = 7$. If item p_1 has the best snippet $S^1_{p_1} = \langle \mathsf{Nikon},\ \mathsf{SLR} \rangle$ and item p_2 has the second best snippet $S^2_{p_2} = \langle \mathsf{Canon},\ \mathsf{red} \rangle$, $d_{p_1} = 100\vec{0}001$, $d_{p_2} = 010\vec{0}100$; $diversity(S^1_{p_1}, S^2_{p_2}) = 4$. The consideration of such a Hamming distance like metric, satisfying triangular inequality, facilitates the development of an efficient algorithm as shown in Section 5.4.2.2. In this study, it is not our goal to advocate one particular diversity measure over another. Rather, we focus on formalizing the problem and developing efficient solutions to it.

Our DISD problem is conspicuously different from the body of work in diversification of search results [41][40]. Diversity aware search aims to find the top-k relevant items from the set of all n relevant items returned as a result of search query. On

the other hand, the DISD problem aims to find the top-1 result (i.e., snippet) for n relevant items, where each relevant item has a set of top-k results (i.e., combination of snippets) to choose from. A brute-force exhaustive approach (henceforth, referred to as Naive-DISD) to solve the problem requires us to explore k^n combinations, and compute sum $\Sigma_{i=1}^{n}(f(S_{p_i}, T)$,subject to diversity$(S_{p_x}, S_{p_y}) \geq \tau$, for all pairs of p_x and p_y. Thus, we develop efficient and practical algorithms.

5.4.2.1 Exact-DISD : E-DISD

We introduce an exact algorithm for any k (though, DISD problem objective is determining the $top-1$) based on interesting and non-trivial adaptations top-k querying techniques in [40][42]. The main idea of our Exact-DISD (E-DISD) is:

We create n lists $\mathcal{L} = \{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_n\}$ corresponding to the n items returned by search engine for a user query. Each list L_i contains the top-k snippets $\{S_{p_i}^1, S_{p_i}^2, \ldots, S_{p_i}^k\}$ for item p_i having scores (given by Equation 5.6) sorted in decreasing order of score. The lists are accessed in round robin fashion and for every join, we check if it is a valid join. A join is considered to be *valid* if the diversity constraint is satisfied, i.e., any two snippets S_{p_x} and S_{p_y} for items p_x and p_y ($S_{p_x} \in \{S_{p_x}^1, S_{p_x}^2, \ldots, S_{p_x}^k\}$, similarly for S_{p_y}) are dissimilar and have $diversity(S_{p_x}, S_{p_y})$ exceeding a user-provided threshold, τ. The complete score of the join is resolved by summing over the snippet scores, i.e., $\Sigma_{i=1}^{n} f(S_{p_i}^j, T)$, $j \in \{1, k\}$. Our objective is to identify the top-1 combination which would return the n snippets S_{p_1}, S_{p_2}, ..., S_{p_n} for the n items, where $S_{p_i} \epsilon$ $\{S_{p_i}^1, S_{p_i}^2, \ldots, S_{p_1}^k\}$, $\forall i = 1(1)n$. The top-1 result is returned if its score is higher than the MPFS (Maximum Possible Future Score), which is the upper bound on the score of an unseen combination. To compute MPFS, we assume that the current entry from a list is joined with the top entries from all other lists, as given below:

$$MPFS = \frac{1}{1 + max((c_1.h_2 \ldots .h_n), (h_1.c_2 \ldots .h_n), \ldots, (h_1.h_2 \ldots .c_n))} \tag{5.9}$$

where, where c_i and h_i are the last seen and top entries from list $\mathcal{L}_i$ respectively. Algorithm 12 is the pseudo-code of our E-DISD algorithm.

Algorithm 12 E-DISD ({p_1, ..., p_n}, n, s, k, {{$S^1_{p_1}, \ldots, S^k_{p_1}$}, {$S^1_{p_2}, \ldots, S^k_{p_2}$}, ..., {$S^1_{p_n}, \ldots, S^k_{p_n}$}}, τ): S_{p_1}, S_{p_2}, ..., S_{p_n}

Create lists $\mathcal{L} = \{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_n\}$ for n items where each list contains the top-k snippets for the item (corresponding to the list) sorted in decreasing value of score (given by Equation 5.6).

//Main Algorithm

1: Top-k-Buffer $\leftarrow \emptyset$
2: **while** Top-k-Buffer.size() $\neq k$ **do**
3: Retrieve the next combination C (i.e., set of n snippets) in round robin manner by Join($\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_s$)
4: **if** VALID JOIN: diversity(S_{p_x}, S_{p_y}) $\geq \tau$, $\forall$ (p_x, p_y) $\in \{p_1, \ldots, p_n\}$ **then**
5: Add entry {C, TotalScore (C)}to CandidateList // TotalScore = $\Sigma_{i=1}^{n} f(S^j_{p_i}, T)$, $j \in \{1, k\}$
6: Calculate MPFS //Equation 5.9
7: **if** CandidateList has entry with TotalScore$\geq$MPFS **then**
8: **return** entry // entry = $\{S_{p_1}, S_{p_2}, \ldots, S_{p_n}\}$
9: **end if**
10: **end if**
11: **end while**

Example 5.4.3. *Let us consider the dataset in Table 5.4.1.1 of 10 items having 6 attributes and 2 tags. Suppose, a user search query returns item with ID=*1, 2, 3, *and* 4. *The top-*4 *snippets of size* 3 *(i.e., $k = 4, s = 3$) of each item is known by employing E-ISD. Our objective is to find the top-*1 *combination (i.e., snippets for the 4 items). Let the diversity threshold be set at $\tau = 2$. We create 4 lists $\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$, and $\mathcal{L}_4$ containing the top-*4 *size-*3 *snippets corresponding to the 4 items, as shown in*

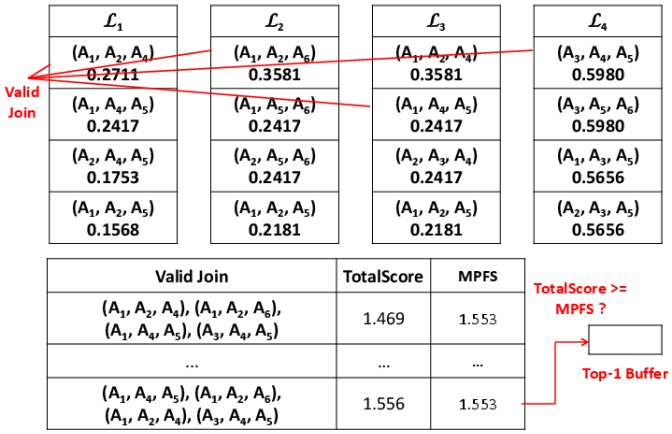


Figure 5.4. Exact-DISD Algorithm for Item IDs -1, 2, 3, 4 in Table 5.4.1.1 .

Figure 5.4. The first join is: $S_1^1 = (A_1, A_2, A_4)$ *from* $\mathcal{L}_1$ *with* $S_2^1 = (A_1, A_2, A_6)$ *from* $\mathcal{L}_2$ *and* $S_3^2 = (A_1, A_2, A_4)$ *from* $\mathcal{L}_3$ *and* $S_4^1 = (A_3, A_4, A_5)$ *from* $\mathcal{L}_4$*, which is invalid since the diversity constraint is not satisfied. We access the lists in a round-robin fashion to get the first valid join, as shown in Figure 5.4. The total score of the join is 1.469, while MPFS is 1.553. Since 1.469¡1.553, we move to the next valid join. In this way, we finally obtain* $S_1^2, S_2^1, S_3^1, S_4^1$ *as the top-*1 *combination by looking up only* 8 *valid combinations, without having to explore all* $4^4 = 256$ *combinations.* □

5.4.2.2 Approximation-DISD: A-DISD

The exact algorithm of Section 5.4.2.1 has exponential time complexity in the worst case. Therefore, we propose an approximation algorithm A-DISD which borrows idea from methods to handle the facility dispersion problem, that deals with location of facilities on a network in order to maximize distances between facilities, minimize transportation costs, avoid placing hazardous materials near housing, etc.

We consider the facility dispersion problem variant in Ravi et al.'s paper [27] that maximizes some function of the distances between facilities to select p out of n facilities. The optimality criteria considered in the work are MAX-MIN (i.e., maximize

the minimum distance between a pair of facilities) and MAX-AVG (i.e., maximize the average distance between a pair of facilities). Under either criterion, the problem is known to be NP-hard by reduction from the Set Cover problem, when the distances satisfy the triangle inequality. The approximation algorithm for the MAX-AVG dispersion problem is as follows: it initializes a pair of nodes (i.e., facilities) which are joined by an edge of maximum weight and adds a node in each subsequent iteration which has the maximum distance to the nodes already selected. Recall that, our DISD problem wants to maximize both relevance and diversity (i.e., distance between facilities), and hence we develop the following distance function, which is shown to be metric in [46]:

$$dist(S_{p_x}, S_{p_y}) = f(S_{p_x}, T) + f(S_{p_y}, T) + 2\lambda.diversity(S_{p_x}, S_{p_y}) \tag{5.10}$$

Also, recall that the DISD objective is to select n entries (one snippet for each item) from nk entries such that no two of the n entries belong to the same item. We adapt Ravi et al.'s MAX-AVG algorithm to solve our DISD problem having this additional parameter - k - in the following way:

- In order to ensure that the selected n snippets belong to n different items, we set $dist(S^i_{p_x}, S^j_{p_x}) = 0, \forall(i,j) \in \{1,k\}$. This ensures that the top-k snippets belonging to the same snippets do not join with each other.

Algorithm 13 is the pseudo-code of our A-DISD algorithm; and Theorem 5.4.2 discusses its theoretical properties, which follows from [27].

Theorem 5.4.2. *Let OPT and APP denote respectively the optimal set and approximate set of n snippets selected out of nk snippets, such that no two of the n snippets belong to the same item. If the distance between the snippets (dist) satisfies triangular inequality,* $OPT/APP \leq 4$.

Algorithm 13 A-DISD ($\{p_1, \ldots, p_n\}$, n, s, k, $\{\{S^1_{p_1}, \ldots, S^k_{p_1}\}, \{S^1_{p_2}, \ldots, S^k_{p_2}\}, \ldots, \{S^1_{p_n}, \ldots, S^k_{p_n}\}\}$): $S_{p_1}, S_{p_2}, \ldots, S_{p_n}$

//Main Algorithm

1: $M^I \leftarrow$ Compute $nk \times nk$ distance matrix using $dist$ in Equation 5.10 //Set $dist(S^i_{p_x}, S^j_{p_x})$=0, $\forall\{i,j\}$=1(1)k in M^I; I is input set of all nk snippets
2: $\{S_{p_x}, S_{p_y}\} \leftarrow \text{MAX}(M^I)$
3: $J \leftarrow \{S_{p_x}, S_{p_y}\}$
4: **while** $J \neq n$ **do**
5: $\quad S_{p_z} \leftarrow \Sigma_{\{S_{p_z} \in [I-J]\}} \text{MAX}(M^{I-J})$
6: $\quad J \leftarrow [J, S_{p_z}]$
7: **end while**
8: $G^{app} \leftarrow J$
9: return G^{app} // $G^{app} = \{S_{p_1}, S_{p_2}, \ldots, S_{p_n}\}$

Example 5.4.4. *Let us again consider the dataset in Table 5.4.1.1 of 10 items having 6 attributes and 2 tags. Let the user search query returns item with ID = 1,2,3 and 4. The top-4 snippet of size 3 (i.e., $k = 4, s = 3$) of each item is known by employing E-ISD. The objective is to find the top-1 diversied combination (i.e, snippets for the 4 items). Let, S_{ij} denote the jth snippet returned by E-ISD for the ith product. List of snippets for ID = 1,2,3 and 4 are shown in upper part of Figure 5.4 as $\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$, and $\mathcal{L}_4$ respectively. The distance between two snippets are caluclated using Equation 5.10. Here is the distance from S_{11} to all the other snippets {S_{11}: 0 , S_{12}: 0 , S_{13} : 0 , S_{14}: 0, S_{21}: 3.62, S_{22}: 4.53, S_{23}: 5.51, S_{24}: 3.48, S_{31}: 1.63, S_{32}: 2.51, S_{33}: 3.51,S_{34}: 3.48, S_{41}: 4.86, S_{42}: 6.86, S_{43}: 4.83, S_{44}: 5.83}. For a particular snippet of an item, all other snippets of that item will have a distance of zero. Next, as depicted in Algorithm 13, we choose the snippets with maximum distance. Here, S_{21} and S_{41} has the maximum distance(6.95) among all pair of snippets. So we choose S_{21} and S_{41} and put into the result set J. Then, we contiue the iteration until we find snippet for each individual item. Inside the while loop, in line 4 of Algorithm 13, we choose a snippet and from either item 1 or item 3. For each snippet which is not already*

selected, we take the distance from each selected snippet and take the sum over them. The snippet which have the highest sum is added to the result. Snippet S_{13} *has the highest sum of 9.30(5.53 + 3.77) considering the distances from* S_{21} *and* S_{41}*. So we add* S_{13} *to* J *and* J *is now* $\{S_{21}, S_{41}, S_{13}\}$*. We will contiune to iterate in similar manner until the result set has the size of total number of items (here, number of items,* $n = 4$*). In the next iteration, we find a snippet from item 3 which has the maximum sum of distance from the already selected snippets.* S_{34} *has the maximum sum of 11.786(2.57 + 5.81 +3.39) considering the distance from the snippets in* J*. We append* S_{34} *to* J *and* $J = \{S_{11}, S_{42}, S_{32}, S_{34}\}$*. Then the algorithm terminates and return* J *as the size of* J *is equal to the number of items returned as the result of the query.* □

5.5 Experiments

We conduct a set of comprehensive experiments using both synthetic and real datasets for quantitative (Section 5.5.1) and qualitative analysis (Section 5.5.2) of our proposed algorithms. Our quantitative performance indicators are (a) *efficiency* of the proposed algorithms measured by running time and the number of combinations that are considered from the pool of all possible combinations; (b) *approximation factor* of results produced by the approximation algorithm measured as the ratio of the acquired approximate result score to the actual optimal result score. We also conduct a detailed use-case evaluation, where we show how our snippets are helpful to draw user attention.

Real Car Dataset: We crawl a real dataset of 606 cars[5] spanning 34 brands from Yahoo! Autos(http://autos.yahoo.com/) for the year 2010. The products contain

[5]Recall that, the number of items in the dataset is not important for the execution cost.

technical specifications as well as ratings and reviews, which include pros and cons. We parse a total of 60 attributes: 25 numeric, and 35 boolean and categorical (which we generalize to boolean). The total number of reviews we extract is 2,180. We process the text listed under pros in each review to identify a set of 15 desirable tags such as `fuel economy`, `stylish exterior`, etc, using the keyword extraction toolkit AlchemyAPI (http://www.alchemyapi.com/).

Synthetic Dataset: We generate a large boolean matrix of dimension 10,000 (items)$\times$100 (50 attributes + 50 tags) and randomly choose submatrices of varying sizes, based on our experimental setting. We split the 50 independent and identically distributed attributes into four groups, where the value is set to 1 with probabilities of 0.75, 0.15, 0.10 and 0.05 respectively. For each of the 50 tags, we pre-define relations by randomly picking a set of attributes that are correlated to it. A tag is set to 1 with a probability p if majority of the attributes in its pre-defined relation have boolean value 1.

We use the synthetic datasets for quantitative experiments, while the real dataset is used in the qualitative study.

System configuration: Our prototype system is implemented using C#. All experiments were conducted on an Windows 7 machine with 2.30Ghz Intel i5 processor, 64 bit Operating System and 6GB RAM.

5.5.1 Quantitative Evaluation

We first compare the performance behavior of the ISD algorithms - Naive-ISD, E-ISD, and A-ISD in Figures 5.5, 5.6, and 5.7. Since Naive-ISD can only word for small problem instances, we pick a subset from the synthetic dataset containing 1000 items, 50 attributes, 10 tags. Figure 5.5 compares the execution time of the ISD

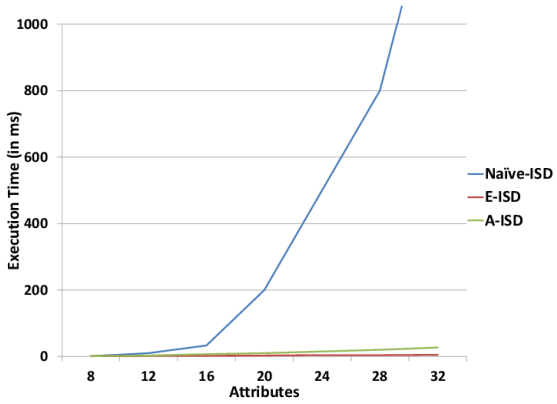


Figure 5.5. ISD: Execution time for varying m (Synthetic data).

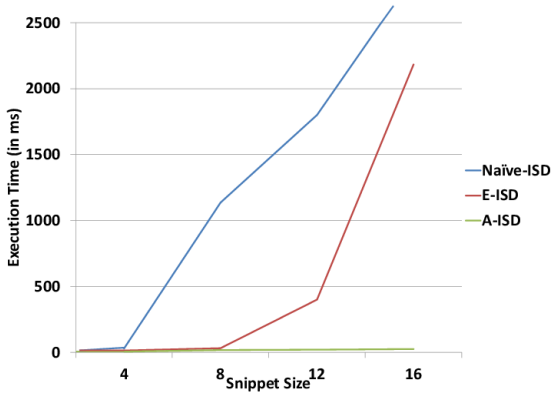


Figure 5.6. ISD: Execution time for varying s (Synthetic data).

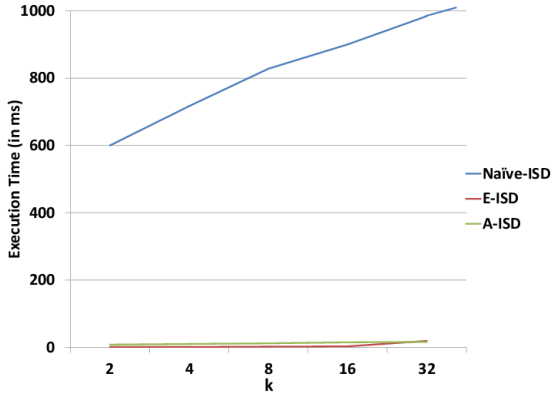


Figure 5.7. ISD: Execution time for varying k (Synthetic data).

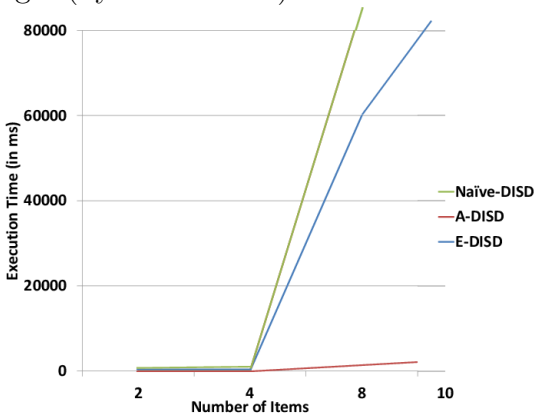


Figure 5.8. DISD: Execution time for varying n (Synthetic data).

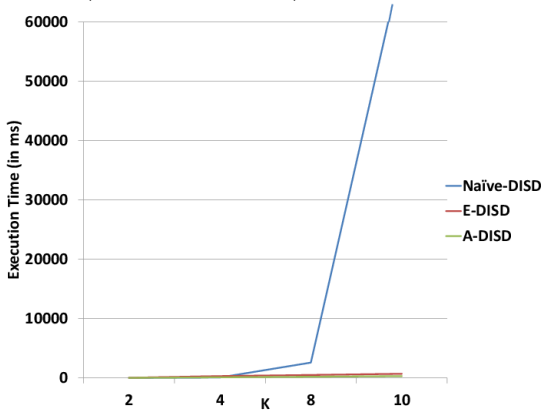


Figure 5.9. DISD: Execution time for varying k (Synthetic data).

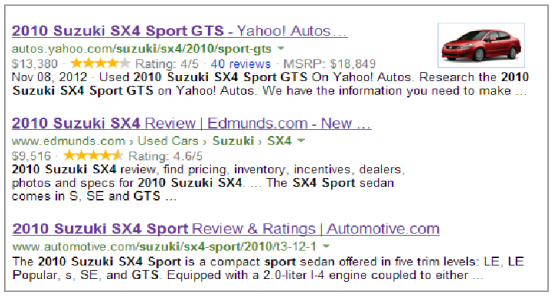


Figure 5.10. Currently Available Snippets for 2010 Suzuki SX4 Sport GTS.

algorithms for 1000 items, 10 tags, having snippet size $s = 5$ and top-k's $k = 5$, with varying number of attributes, m. We observe that our E-ISD and A-ISD (with an approximation factor $\epsilon = 0.1$) outperforms the Naive-ISD method as the number of attributes increases. Table 5.2 also validates that our algorithms are superior to Naive-ISD for the same synthetic data instance and varying m. The number of snippets looked up by E-ISD is orders of magnitude less than that by Naive-ISD.

Table 5.2. ISD - Number of snippets looked up (Synthetic data of 1000 items, 10 tags, snippet size s = 5, k = 5).

Number of attributes	Naive-ISD	E-ISD
8	56	84
12	792	210
16	4368	252
32	201376	462

Next, we analyze the time taken by the three algorithms when the snippet size, s varies in Figure 5.6. We consider a synthetic data containing 1000 items, 20 attributes, 10 tags, $k = 10$, and vary s from 4 to 16. We observe that the time taken by Naive-ISD is again much more than that taken by our methods - E-ISD and A-ISD. We also observe that the time taken by E-ISD is affected by s, since the number of lists considered is equal to the snippet size. However, A-ISD is fairly stable with increase in s. Figure 5.7 compares the execution time of Naive-ISD, E-ISD and A-ISD for varying k. We consider a synthetic data containing 1000 items, 30 attributes, 10 tags, $s = 10$, and vary k from 2 to 32. As expected, the running time of all three algorithms increases with increase in k. We do not evaluate the ISD algorithms for varying number of items (since number of items in dataset is not important for execution cost), and number of tags (since we consider composite tag).

Figures 5.8 and 5.9 compares the execution time of our DISD algorithms - Naive-DISD, E-DISD, and A-DISD. Recall that, the computational complexity of the DISD problem is dependent on the number of relevant items n and the k of $top-k$. Therefore, we evaluate the performance behavior of our DISD algorithms by varying n and d. Figures 5.8 shows how the execution time of Naive-DISD and E-DISD rises sharply with increase in the number of relevant items, when synthetic data of 1000 items, 30 attributes, 10 tags, snippet size, k of top-k 10, and $\tau = 2$. Our A-DISD performs extremely well with increase in the number of relevant items, while Theorem 5.4.2 guarantees result quality. Figure 5.9 is similar in behavior to Figures 5.8 for a similar synthetic data instance, and clearly shows how our A-ISD algorithm is effective for large real datasets.

We empirically demonstrate the quality of results generated by our approximation algorithms A-ISD and A-DISD, as shown in Table 5.3. For A-ISD on a synthetic data having 1000 items, 40 attributes, 10 tags, snippet size 5, and $\tau = 2$, the obtained approx factor is the ratio of the score for the top-1 snippet to the score of the optimal snippet returned by E-ISD. For A-DISD on a synthetic data having 8 items with each item having top-4 snippets, the obtained approx factor is the ratio of the relevance score ($\Sigma_{i=1}^{n} f(S_{p_i}, T)$) for the best snippet combination to the score of the optimum returned by E-DISD. Note that, A-DISD does not have an user-submitted ϵ as input. The diversity threshold in the results obtained by E-DISD and A-DISD is 2.

Table 5.3. A-ISD and A-DISD: Quality(Synthetic data).

Algorithm	User-Defined Approx factor ϵ	Approx factor Obtained
A-ISD	0.5	0.97
	0.2	0.98
	0.1	0.99
A-DISD	-	0.98

5.5.2 Qualitative Evaluation

We use the real cars dataset to confirm that our algorithms draw interesting snippets highlighting the desirability of certain car specifications (attribute values), as opposed to the snippets that are currently returned by the search engines. For a user looking for a used japanese sports car, one of the top cars returned by the search engine is "Suzuki SX4 Sport". For the 2010 Suzuki SX4 Sport GTS, the usual snippets displayed by the search engine and/or the retail sites are shown in Figure 5.10. As we see, the snippet compositions are not striking and discuss the usual high-level car specifications, that other cars returned by the search query mostly display. Our E-ISD algorithm returns the following snippet, that highlights salient and query-relevant attributes like `mileage`, `horsepower`, `safety features`, etc.

> **2010 Suzuki SX4 GTS:** $13,380 based on 24,000 driven miles; 23 mpg city / 30 mpg hwy; 148 hp; MPFI Engine; KYB(R) Shock Absorbers and Sport Ride Type

Next, we study how diverse snippets are returned by our A-DISD algorithm. For a user looking for a used audi a4, suppose the top cars returned by the search engine include "FrontTrak Multitronic", "Quattro Manual", "Quattro Tiptronic", and "Avant Quattro Tiptronic". The cars share several attributes in common, and are hence likely to generate similar snippets. However, our A-DISD algorithm identifies several unique features that these used cars have such as `front fog-light` in FrontTrak Multitronic, `first-aid kit` in Quattro Manual, `garage door-opener` in Quattro Tiptronic, and `delayed courtesy light` in Avant Quattro Tiptronic. Thus, the diverse snippets returned to an user looking for an used Audi A4 are likely to increase the chances of the user clicking on one of them.

CHAPTER 6

Technical Item Recommendation

6.1 Introduction

When shopping for a new laptop in the traditional way, customers would walk into a shop and rely on the experience of a shopping assistant to help them select the best laptop for their needs. A good shopping assistant would ask questions intelligible to non-expert users, for instance, "*Do you intend to use the laptop for playing modern computer games?*", while mapping the answers to technical product specifications, such as, "*The customer will need at least 4* GB *of* RAM." E-commerce has disrupted this custom in two ways. First, customers shop online, from their homes, without any human interaction involved. Second, catalogs of online shops are so big and with so many continuous updates that no human, however expert, can effectively comprehend the space of available products. As a consequence, the customers are left without any guidance to understand their *needs* and map them to a product.

In this chapter, we propose a system that addresses this need. Our system, called ShoppingAdvisor, draws inspiration from a recent marketing strategy called *"Which product should I buy?"* flowchart. Each box in these flowcharts asks the prospective shopper a question, and the sequence of answers leads the shopper to the suggested shopping option. Figure 6.1 shows an example flowchart to choose the right version of the Amazon Kindle e-book reader[1]. At each level the user should look at a question on the left side (e.g., "Input Method"), choose one of the available options (e.g., 5-way controller, Touch or Keyboard) and note down the "orange word"

[1] http://theunderstatement.com/$pick_y our_k indle/$

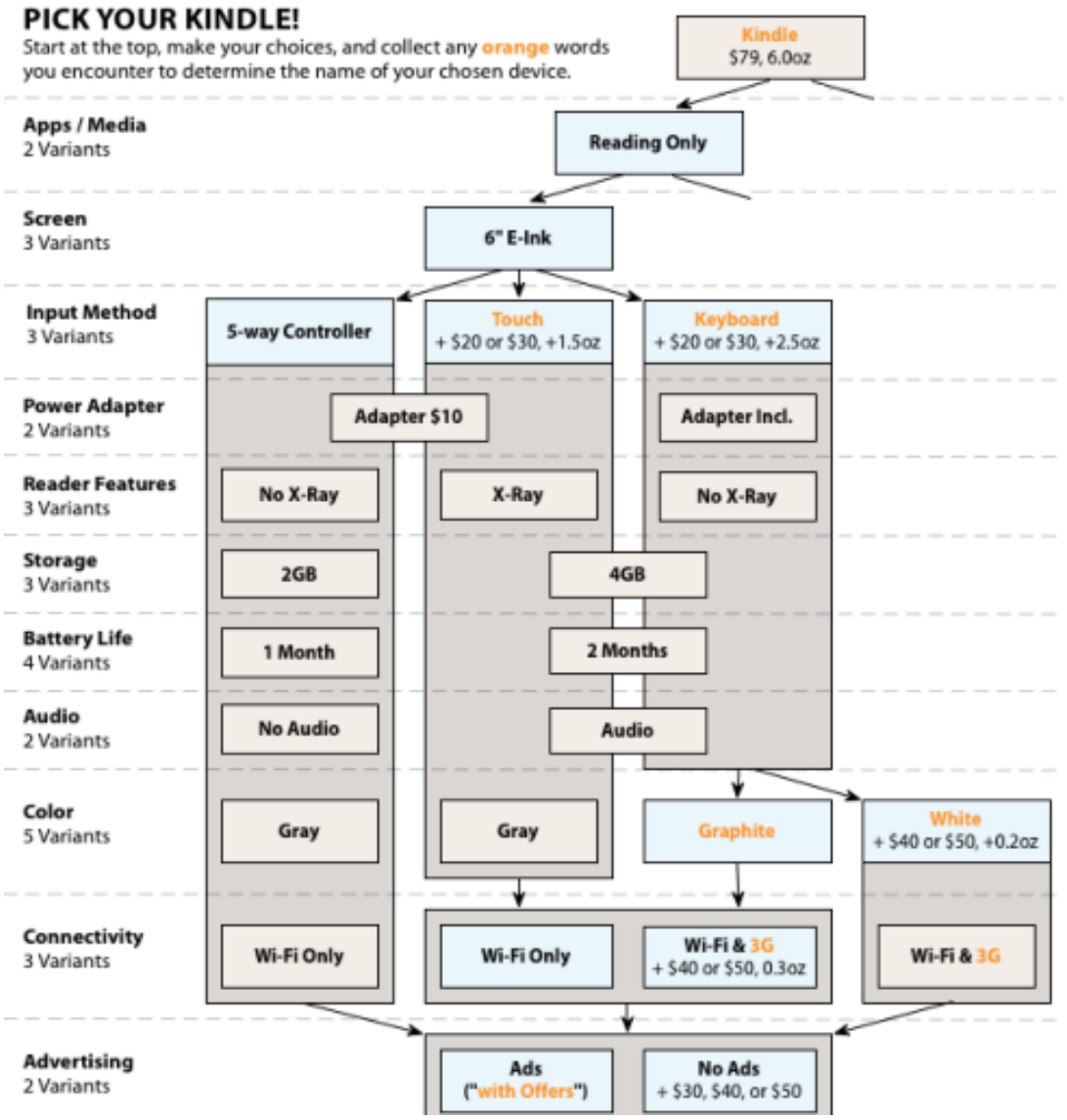


Figure 6.1. Example of "Which Kindle should I buy?" flowchart.

in the box. At the end of the flowchart, the correct version of Kindle will be made by the concatenation of "orange words" along the path chosen by the user. The kind of questions asked by the flowchart requires a high level of technical knowledge, to make an informed decision. For example, to understand the "Reader Features" question the user should know that X-Ray is a feature to allow to view a list of all passages from the book that mention an idea, person, or topic.

"*Which product should I buy?*" flowcharts present two problems. First, designing such flowcharts manually is time consuming. Second, they rely on questions about technical attributes, which an average shopper might not understand. For example, the flowchart guide to buying a laptop might ask the question "*Do you need more than 4GB of RAM?*", and expect the shopper to understand the purpose of RAM. We address both problems by automatically designing flowcharts that help shoppers select the best product for their needs. An important design principle of our approach is that we distinguish between two types of features: (*i*) features in a *user space*, which contain user information such as demographics, life-style preferences, and interests; and (*ii*) features in a *product space*, which contain technical information of the products, e.g., the CPU speed of a laptop.

ShoppingAdvisor generates a tree-shaped flowchart, in which the internal nodes of the tree contain questions addressed to the users. These questions involve only attributes from the user space, that non-expert users can understand easily. For instance, "*Are you a student?*" or "*Would you be storing videos in your laptop?*". A potential shopper, starts from the root of the ShoppingAdvisor tree, answers questions, follows the control flow, and descends towards a leaf of the ShoppingAdvisor tree, where they find a ranked list of product recommendations. In fact, a user can stop answering at any level in the tree as a ranking is available at each node, not only at leafs.

At a high-level, the ShoppingAdvisor tree resembles a *decision-tree*, and our method for learning its structure resembles decision-tree learning algorithms. However, there are important differences. First, given a set of users with similar features, our system induces a ranking on *all* the existing products, based on the attributes of the products. Thus, unlike traditional decision trees in which each node is a weak decision maker, each tree node in our model is a strong classifier and

it outputs a ranking on products instead of class labels. Thus, unlike traditional decision trees in which each node outputs a class label, we learn trees in which each node outputs a ranking of products. Second, unlike other tree-based methods, in our model the splitting and ranking domains are different. Indeed, the ShoppingAdvisor tree partitions the users on the basis of user attributes, so that similar users will end up in the same nodes, under the assumption that they will prefer similar products. However, the ranking of products at each node depends on their technical attributes. In this way, if the system learns that, say, the storage capacity of a laptop is an important feature for a particular user segment, the system will weigh this feature appropriately and will rank high other laptops with large hard disks. Thus, the system learns the implicit relationship between user attributes and product attributes, and identifies the best recommendations for a particular set of users. This identification of implicit relationships between user and product attributes lets the system deal with both new users and with new items, alleviating cold-start problems.

Figure 6.2 shows an example of a "*Which camera should I buy?*" flowchart with technical questions (left) and an equivalent example with non-technical questions produced by ShoppingAdvisor (right). A shopper answering `yes` to both questions "*Do you want a camera for traveling?*" and "*Do you want a camera for backpacking trips?*"f is more likely to be recommended a ultra-compact camera[2].

We demonstrate the effectiveness of our interactive recommendation system for two categories of products: *cars* and *cameras*. In addition to the conceptual contributions – introducing a new problem definition and developing novel methods to solve this problem – we also show how to mine publicly available data to create datasets required by our problem definition, namely, integrating information regarding users,

[2]Our ShoppingAdvisor recommendations are not influenced by, or biased towards, any particular brand.

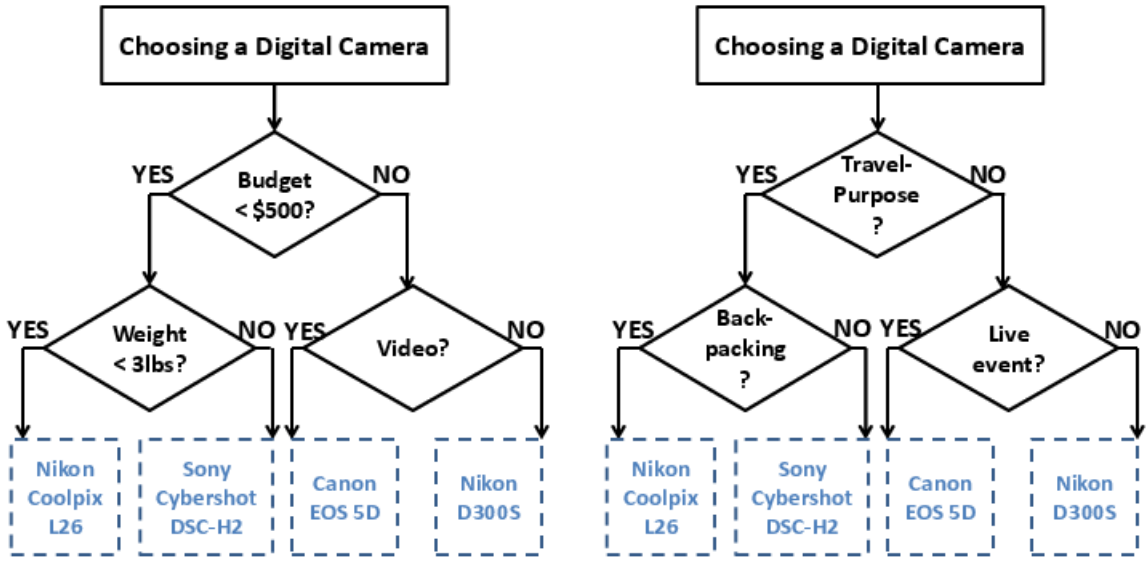


Figure 6.2. Example of "Which camera should I buy?" flowchart (left) and its equivalent non-technical ShoppingAdvisor flowchart (right).

products, and reviews. For our first use case, cars, we extract information from Yahoo! Autos.[3] We collect car specifications (numerical, boolean and categorical attributes), as well as ratings and reviews submitted by users. Each user review includes pros and cons, short free-text snippets that highlight positive and negative aspects of the car. We employ standard text-mining techniques to extract tags such as `fuel economy`, `stylish` and `performance` from the reviews. We use those tags as user attributes: we represent each user by the set of tags that the user has used in the pros section of his reviews. We take a different route for cameras and use data from flickr.[4] In this case the tags used by each user to describe their photos are used as a proxy to user interests. We extract the cameras used to take the photos with such tags from the metadata, and we retrieve their technical specifications from CNET[5]. Rather than

[3]http://autos.yahoo.com
[4]http://www.flickr.com
[5]http://www.cnet.com

using review scores from CNET, we choose to test our framework in a different setting, where review scores might not be available, and use popularity of a camera in flickr as a proxy. For each tag, we count how many pictures were taken with a given camera, and use this number to rank the cameras. We use mean reciprocal rank (MRR) to evaluate ShoppingAdvisor, and we show how the performance increases by more than 50% along the path from root to leaf. We also show how collaborative filtering algorithms such as k-nearest neighbor benefits from *feature selection* done by ShoppingAdvisor. We obtain results of comparable quality by using only a subset of user attributes, i.e., features for k-NN, while producing more interpretable recommendations.

6.2 Problem Framework

Once again, we model a collaborative content site $\mathcal{D}$ as a triple $\langle \mathcal{U}, \mathcal{I}, \mathcal{R} \rangle$, representing the set of reviewers (i.e., users), the set of items (i.e., products) and the feedback, respectively. Each feedback r can be considered as a triple itself, represented as $\langle u, p, s \rangle$, where $u \in \mathcal{U}$, $i \in \mathcal{I}$, $(r, T) \subset \mathcal{R}$, respectively. We define a user schema, $\mathcal{U}_\mathcal{A} = \langle ua_1, ua_2, \ldots \rangle$, to represent each user as a set of attribute values conforming to the user schema: $u = \langle uv_1, uv_2, \ldots \rangle \in \mathcal{U}$, where each uv_x is a value for the attribute $ua_x \in \mathcal{U}_\mathcal{A}$. The corresponding user table $\mathbf{U}$ contains information about the users and it has dimensions $n_U \times m_U$. That is, we consider information for n_U users from $\mathcal{D}$, and each user is described with m_U attributes. We do not make any explicit assumption regarding what type of information we have, it may be any kind information, for example, registration information such as demographics, or implicitly provided information, such as tags used on social-media sites, or browsing behavior. Such information expresses the interests and lifestyle preferences of users. We use the notation u_i to refer to the i-th user. Similarly, we define an item schema, $\mathcal{I}_\mathcal{A} = \langle pa_1, pa_2, \ldots \rangle$, to

represent each item as a set of attribute values, $i = \langle pv_1, pv_2, \ldots \rangle$, where each pv_y is a value for the attribute $pa_y \in \mathcal{I}_{\mathcal{A}}$. The corresponding product table $\mathbf{P}$, is of dimensions $n_P \times m_P$ and contains product information. We assume that we have information for n_P products, and each product is described with m_P technical attributes in $\mathcal{I}_{\mathcal{A}}$, typically provided by the manufacturer. For instance, for cameras the attributes contain information such as resolution, zoom, aperture, and weight. We use the notation p_j to refer to the j-th product. The third table created from $\mathcal{D}$ is the review table $\mathbf{R}$ with dimensions $n_C \times 3$, and each row describing a reviewing action (u_i, p_j, s_{ij}). This row contains data about user u_i evaluating product p_j with score s_{ij}. We may think of s_{ij} as a numeric rating, say, $s_{ij} \in [1, 5]$, where $s_{ij} = 1$ reflects a negative opinion and $s_{ij} = 5$ is positive. Obviously, many other scoring schemes are possible.

Table 6.1. An example user table $\mathbf{U}$.

uid	Gender	Age	Travel	Family	Live	Video
1	female	young	1	1	0	0
2	male	old	0	1	1	0
3	female	young	0	0	1	1
4	male	teen	1	0	0	1

Table 6.2. An example product table $\mathbf{P}$.

pid	Weight	Resolution
1	5.0 lbs	36.3 MP
2	1.7 lbs	10.1 MP
3	2.0 lbs	14.0 MP
4	4.5 lbs	28.1 MP

Table 6.3. An example review table $\mathbf{R}$.

uid	pid	Rating
1	3	4.5
2	2	3.5
3	1	4
3	3	3

Examples of a user table, a product table, and review table are shown in Tables 6.1, 6.2, and 6.3, respectively. Note that a user may review more than one product, and a product may be reviewed by more than one user. For ease of reference, we denote by $\mathcal{P}$ the set of all products and by $\mathcal{U}$ the set of all users in our

system. The set of all user attributes is denoted by $\mathcal{A}$. Note that $|\mathcal{P}| = n_P$, $|\mathcal{U}| = n_U$, and $|\mathcal{A}| = m_U$.

The system we design and build here, ShoppingAdvisor, is a *decision tree* that guides the users in making their shopping decisions. We denote this tree by $\mathcal{T}$. The internal nodes of the ShoppingAdvisor tree $\mathcal{T}$ correspond to user attributes in $\mathcal{A}$. A potential shopper is supposed to use the tree $\mathcal{T}$ as a *flowchart* for receiving product recommendations. A user starts from the root of the tree. The root, as any internal node, contains a user attribute $a \in \mathcal{A}$. The attribute a is perceived as a question by the user. For example, if a is a demographics attribute, the question can be *"Are you in the a demographics group?"*, while if a is a tag, the question can be *"Are you interested in tag a?"* The user, by answering this question, follows the left or right subtree, and continues recursively answering questions in internal nodes of $\mathcal{T}$until they reach a leaf node or they decide to stop.[6] The leaf nodes of $\mathcal{T}$ correspond to an *ordering* of the products in $\mathcal{P}$. Once a shopper reaches a leaf node $\ell \in \mathcal{T}$, by following the internal nodes of $\mathcal{T}$ and answering questions, ShoppingAdvisor recommends to the shopper products in the order specified by the leaf node ℓ. In practice, we assume that ShoppingAdvisor recommends the top-k products in the ordering, although the user may select a "`more` k `products`" button. We leave such system details outside our discussion and focus on the algorithmic abstractions.

We now define the main problem we address in this chapter.

Problem 6.2.1 (ShoppingAdvisor)**.** *We are given as input a product table* $\mathbf{P}$*, a review table* $\mathbf{R}$*, a user table* $\mathbf{U}$*, and integers* h *and* k*. The task is to learn a* ShoppingAdvisor *tree* $\mathcal{T}$*. Each internal node of the tree contains a question formed by*

[6] In this chapter, we focus on identifying the attribute of interest, and not on the task of formulating the question in a human-interpretable way. We assume that this problem can be solved independently, even with human supervision of the designers of the ShoppingAdvisor application.

a user attribute $a \in \mathcal{A}$. Each tree node contains a top-k ranked list of the products $\mathcal{P}$. The height of the tree is restricted to be h.

The objective of the tree $\mathcal{T}$ is to provide relevant recommendations on products $\mathcal{P}$. A new shopper, starting from the root, traverses down the tree $\mathcal{T}$, answering at most h questions, until reaching a leaf node, where the user receives the top-k recommendations contained at that leaf node. The quality of the learned tree $\mathcal{T}$ reflects the quality of such recommendations made to a new user/shopper.

To make the definition of our problem precise, we need to quantify how we evaluate recommendations made by the ShoppingAdvisor tree $\mathcal{T}$. For this task, we use *ten-fold cross-validation*. Users in the evaluation fold have reviewed products – their scores are contained in the review table $\mathbf{R}$ – and thus we can evaluate the quality of $\mathcal{T}$'s recommendations. As we need to evaluate a ranked list we can employ standard measures from information retrieval.

As already mentioned, we model our ShoppingAdvisor system as a decision tree. We learn the structure of the tree by partitioning the set of users in the training set recursively on the basis of available user attributes – such as, demographics, tagging behavior, and so on – and then match a test user, to the leaf node of users with whom the user shares their preferences. For each group of users, we also learn the product attribute weights and generate a ranking of top-k products in the training set, in order to identify the top products for recommendation.

6.3 Algorithms

6.3.1 General Algorithm

We first introduce a general algorithmic framework for solving the ShoppingAdvisor problem. Our algorithmic framework uses two functions as black boxes.

The first is a payoff function, used to choose the best question to ask at any node of the ShoppingAdvisor tree $\mathcal{T}$. In other words, the function payoff determines the best user attribute $a \in \mathcal{A}$ to partition the set of users at any node of the tree $\mathcal{T}$. The second function is a rank function, used to determine the ranking of the products recommended to users. Learning the rank function can be seen as learning weights on the attributes of products, which in turn can be used to rank all products in $\mathcal{P}$; and subsequently select the top-k products.

Typically, decision trees are constructed in a top-down fashion, where each internal node splits the training instances into two or more subspaces according to the output of a certain function evaluated on the input data. In the ShoppingAdvisor tree $\mathcal{T}$, each internal node of the tree corresponds to the set of users whose attributes match the attributes at all internal nodes on the path from the root to that node. Thus, we recursively partition the set of users at each internal node. The partition criterion is that the users within each side of the split should agree on their ratings on the products. The goal is to select the user attribute, so that when we perform the split based on that attribute, the uniformity criterion on the user ratings is maximized. As an example, if it so happens that avid hikers tend to prefer a certain camera, for its weight, ruggedness, and ability to take high-quality outdoors photos, then the `hiking` tag should be used to split the active users at that step of the construction of the tree.

Each tree node contains a ranking of products, induced from a learning-to-rank model that uses the ratings of the users belonging to that node. Consider a shopper who has cascaded until a node q of the tree $\mathcal{T}$. If q is a leaf node, the shopper is provided with a top-k list of product recommendations. If q is an internal node, the shopper is asked the question that corresponds to q, and depending on the answer cascades to one of the two subtrees of q. The shopper is also given the possibility

not to answer the question and receive the current recommendation at node q. This is possible since a product ranking is available at each node. For simplicity, in the evaluation performed in this chapter, we assume that shoppers navigate until tree leafs.

6.3.1.1 Learning the tree structure

Determining the structure of the tree $\mathcal{T}$ is equivalent to choosing which user attribute $a \in \mathcal{A}$ to use for a question at each node.

Formally, consider a tree node q and the set of users U_q who correspond to q, namely, the set of users whose attributes match the attributes at all internal nodes on the path from the root of the tree until q. Given a candidate splitting user attribute $a \in \mathcal{A}$, two subsets of U_q can be defined: the set of users $U_q(a)$ who match attribute a, and the set of users $U_q(\overline{a})$ who do not match attribute a. For simplicity, we assume that $U_q = U_q(a) \cup U_q(\overline{a})$ and $U_q(a) \cap U_q(\overline{a}) = \emptyset$, although it is not required as our framework can handle overlapping subsets. The root node of the tree comprises all n_U users of the user table $\mathbf{U}$. To determine the best user attribute $a \in \mathcal{A}$ to split U_q at node q we evaluate the pay-off function associated with the sets of users resulting from the split. In particular, we consider a $\mathsf{combine}$ function that maps the pay-off of the two subsets to a single-valued measure.

$$\mathsf{payoff}(q, a) = \mathsf{combine}(\mathsf{payoff}(U_q(a)), \mathsf{payoff}(U_q(\overline{a})), |U_q(a)|, |U_q(\overline{a})|, |U_q|),$$

where the function $\mathsf{payoff}(U)$ evaluates the quality of ranking induced by a set of users U. There are a number of natural options for the $\mathsf{combine}$ function, such as *sum*, *arithmetic mean*, *geometric mean* and *harmonic mean*.

One has to consider all possible user attributes $a \in \mathcal{A}$, and choose as splitter the one that maximizes the pay-off.

$$\mathsf{splitter}(q) = \arg\max_{a \in \mathcal{A}} \mathsf{payoff}(q, a). \tag{6.1}$$

The idea behind such a posterior goal of maximizing pay-off is to partition the set of users into two groups, which have similar preferences, and which rank the products in $\mathcal{P}$ in a similar way. Furthermore, we aim at leveraging hidden correlations among the set of user attributes and the set of product attributes. For instance, in the previous example with hikers preferring certain lightweight cameras, our tree should learn the fact that the weight of a camera is an important feature for that specific subset of the population, and thus, it should tend to rank lightweight cameras higher, even if they have not been explicitly rated.

Note that our recursive algorithm is an instantiation of a greedy heuristic. In principle, selecting the best splitter at a certain node according to Equation (6.1) may lead to globally suboptimal solutions. However, this is a cost we have to pay due to the **NP**-hardness of the ShoppingAdvisor problem.

6.3.1.2 Learning product rankings

We next consider the problem of learning to rank the products in $\mathcal{P}$ at a given tree node q. The input consists of users U_q belonging to node q. We also have access to the product table $\mathbf{P}$ and the review table $\mathbf{R}$. In fact, we only need the rows of the review table $\mathbf{R}$ corresponding to U_q, and denote the sub-table as $\mathbf{R}_q$.

The objective is to learn a function $\mathsf{rank} : \mathcal{P} \to \mathbb{R}$ which, given a product $p \in \mathcal{P}$ specified by its technical attributes (i.e., by a row in the product table $\mathbf{P}$), returns a value $\mathsf{rank}(p)$, which in turn can be used to induce a ranking on the set $\mathcal{P}$. We opt for a linear function, and thus the task is to learn weight coefficients for the product attributes. Note that in order to handle categorical attributes of products with our linear ranking function we convert such categorical attributes into a set of boolean attributes, and then treat them as numerical 0–1 values.

To learn the function rank we can use any learning-to-rank algorithm, such as REGRESSION, RANKSVM, or GRADIENTDESCENT. To employ such a learning-to-rank algorithm we need to assign a score to each product p in $\mathbf{P}$. Such a score is computed as the average rating of p in the review table $\mathbf{R}_q$, that is, the average rating of p over the set of users who correspond to the node q under consideration.

6.3.1.3 Putting the learning components together

We now discuss the interaction of the two learning ingredients: learning the tree structure and learning the product ranking. We first observe that the ranking function we learn at a tree node q depends on the set of users (and their ratings on products) who correspond q. We write rank_U to emphasize the dependance of the ranking function from a set of users U. The quality of ranking can be evaluated using a quality measure eval. Functions such as *precision*, *recall*, *normalized discounted cumulative gain* (NDCG), and *mean reciprocal rank* (MRR) can be used as eval functions. We write $\mathsf{eval}(\mathsf{rank}_U)$ to denote the quality of a rank_U ranking measured by such an eval measure. Finally, we recall that a good user attribute to split a tree node q is a user attribute that induces sub-populations with good rankings in each one. The quality of the ranking should be reflected in the payoff function used. Thus we set

$$\mathsf{payoff}(U) = \mathsf{eval}(\mathsf{rank}_U). \tag{6.2}$$

In the next section we discuss our considerations for the functions payoff, rank, and eval.

6.3.2 The LEARNSATREE algorithm

Our proposed algorithm, LEARNSATREE, for learning SHOPPINGADVISOR tree is an instance of the general algorithm presented in the previous section, with judicious choices for the functions payoff, rank, combine, and eval. We now discuss more in

detail those choices. We first introduce our learning-to-rank model and we define the function rank that learns weights of product attributes.

6.3.2.1 Learning to rank

The goal is to learn a weight vector $\mathbf{w} = \{w_1, \ldots, w_{m_P}\}$ for the m_P technical attributes of the products $\mathcal{P}$. As discussed in the previous section, the learn-to-rank algorithm is applied to each tree node q. Assume that for a node q we have a set $\mathbf{P}_q$ of training instances. Those training instances are a subset of products of $\mathcal{P}$ accompanied with review scores from the users associated with node q. Following the notation of the previous section, the training instances are the result of joining the product table $\mathbf{P}$ with the review sub-table $\mathbf{R}_q$. For learning the ranking function rank we employ a pairwise RankSVM method [47]. In this approach, the training instances under consideration are expanded into a set of preference pairs. Namely, we create ordered pairs of products $(\mathbf{p}_i, \mathbf{p}_j)$ where the product $\mathbf{p}_i$ has higher score than the product $\mathbf{p}_j$ from the users in the tree node q. Let us denote the set of such ordered pairs by $\mathbf{P}_q^2$. We then find the weight vector $\mathbf{w} = \{w_1, \ldots, w_{m_P}\}$ by optimizing a pairwise objective function:

$$\min_{\mathbf{w} \in \mathbb{R}^{m_P}} \left\{ \frac{\lambda}{2} \|\mathbf{w}\|^2 + \sum_{(\mathbf{p}_i, \mathbf{p}_j) \in \mathbf{P}_q^2} \mathsf{loss}\left(\mathbf{w}^T \cdot (\mathbf{p}_i - \mathbf{p}_j)\right) \right\},$$

where loss is a suitably-defined loss function, such as *hinge loss*, i.e., $\mathsf{loss}(y) = \max(0, 1 - y)$. For the class of linear ranking functions, the objective of attaining an optimal ranking function rank^*, i.e., finding the weight vector $\mathbf{w}$ so that the number of inversions is minimized, is **NP**-hard [48], and the RankSVM algorithm provides an approximate solution. Once the weight vector $\mathbf{w}$ is learned, the rank function is defined as $\mathsf{rank}(\mathbf{p}) = \mathbf{w}^T \cdot \mathbf{p}$, and it induces a ranking on the *whole* set of products $\mathcal{P}$ by $\mathsf{rank}(\mathbf{p}_1) \geq \ldots \geq \mathsf{rank}(\mathbf{p}_{n_P})$.

Besides promoting the popular products belonging to the training instances for the users of node q, our method ensures that all products will be ranked, even products that have not been reviewed by the set of users in node q. This property helps us handle the cold-start problem, where a new product arrives in the system and initially there is very little feedback available for that product.

Next we evaluate the quality of the ranking generated by our method, which according to Equation (6.2) defines the payoff function.

6.3.2.2 Evaluating the ranking

The learning-to-rank model induces a ranking $\mathsf{rank}(\mathbf{p}_1) \geq \ldots \geq \mathsf{rank}(\mathbf{p}_{n_P})$ on the products in $\mathcal{P}$. The quality of this ranking is measured by the eval function. Our eval function measures the number of correctly-ranked pairs in the ranking generated for the products in $\mathbf{P}_q$, that is, the products in our training set in the node q. So, assuming that the set $\mathbf{P}_q$ contains n products, we have

$$\mathsf{eval}(\mathsf{rank}) = \frac{2\,|\{(\mathbf{p}_i, \mathbf{p}_j) \in \mathbf{P}_q^2 \mid \mathsf{rank}(\mathbf{p}_i) > \mathsf{rank}(\mathbf{p}_j)\}|}{n(n-1)}.$$

Note that the reason why we use this evaluation function, rather than other measures mentioned before such as precision, recall, and NDCG, is that minimizing the number of inversions is the most common way to optimize a pairwise learning-to-rank function. In other words, our choice of the eval function stems from the RankSVM approach.

Likewise, we choose to use *sum* as the combine function. Indeed, by summing the number of correctly ranked pairs we are guaranteed by RankSVM that the eval function is monotonically increasing with the height of the tree. In analogy with decision trees, this property allows for effective pruning strategies while building the tree.

6.3.2.3 Stopping criterion

The construction of a decision tree has another critical element: deciding when to stop growing the tree. Ideally, the algorithm will stop its recursive partitioning of the subspaces along one (or both) direction(s) if the *perfect* ranking is achieved in the left and right children nodes, i.e., the payoff associated with splitting by attribute $a \in \mathcal{A}$ for node q is 0. In reality, we first grow the tree to its entirety, and then employ post-pruning with the aim of removing sections of the tree that provide little power to capture user preferences. For example, for a node q split by tag `travel`, if its child node is split by the near-synonomous tag `vacation`, our post-pruning rules should trim the child node (or, the associated set of nodes). We employ pruning rules on the validation set. Our most significant stopping condition, in addition to the regular rules, follows from the observation that – the number of inversions due to our ranking model on a set of training instances belonging to a node montonically decreases with the decrease in size of the training set along a root-to-node path.

6.4 Experiments

We evaluate our ShoppingAdvisor system with both real and synthetic data. Our primary objective is to demonstrate the effectiveness of our system by comparing the quality of recommendations returned by our system with recommendations made by baseline system(s) not leveraging user attributes. We highlight how popular collaborative recommendation technique(s) benefit from our system. Besides a quantitative comparison of the recommendation quality, we also conduct a detailed use-case evaluation, where we show that recommendations with consideration of shoppers' personal preferences are superior to those by traditional state-of-art systems. We also show the scalability of our algorithm by studying the running time under varying parameters.

Real Car Dataset: Our first dataset is extracted from Yahoo! Autos. We focus on new cars listed for the year 2010 spanning 34 different brands. There are several models for each brand, and each model offers several trims.[7] Since each trim defines a unique attribute-value specification, the total number of trims that we crawl are the 606 products in our dataset. The products contain technical specifications as well as ratings and reviews, which include pros and cons. We parse a total of 60 attributes: 25 numeric, and 35 boolean and categorical (which we generalize to boolean). The total number of reviews we extract is 2 180. In this dataset we do not have user information, and thus we consider that each user has reviewed only one car. Thus, the total number of users is also 2 180. Since the ShoppingAdvisor system considers tags as form of user feedback, we extract tags from the reviews using the keyword extraction toolkit AlchemyAPI [8]. We process the text listed under "pros" in each review to identify a set of 15 desirable tags such as `fuel economy`, `comfortable interior` and `stylish exterior`.

Real Camera Dataset: Our second dataset is on cameras. For extracting user preferences for this dataset we take a different route. Rather than using an e-commerce site such as Yahoo! Shopping or Amazon, we use a social-content site such as flickr. Our intention is to capture user preferences by the tags that people use to describe their photos. So we assume that flickr tags such as `food`, `nature`, `animal` and `landscape` are meaningful representations of user preferences. Furthermore, we intend to leverage the hidden associations between photo tags and cameras. As an example, of the 3 000 photos in our flickr dataset tagged as `food`, Canon EOS 20D, Canon EOS 350D, Canon EOS 400D and Nikon D80 have been used 1 200 times while 550 other cameras have been used the rest of the times. Therefore, a shopper looking for a camera for

[7] Trims denote different configurations of standard equipment.
[8] http://www.alchemyapi.com/api/keyword/

taking food photos will be recommended a camera from among Canon EOS 20D, Canon EOS 350D, Canon EOS 400D and Nikon D80. The technical specifications of the cameras are retrieved from CNET. Our flickr dataset sample has 135 025 photos uploaded by 37 064 users using 9 365 cameras; the tag vocabulary size is 422 240. We clean the dataset to consider only those instances in which (*i*) the cameras have well-defined set of technical specifications in CNET and are not phone cameras or digital camcorders; (*ii*) the tags are valid English words; and (*iii*) a user has used at least two cameras. Our final reduced flickr dataset has 11 468 photos and 10 845 tags from 5 647 users and 654 cameras. Since the number of tags is huge, we use latent Dirichlet allocation (LDA) technique and aggregate the tags into 25 topics based on their co-occurrence. We then express user preferences at the level of LDA topics. Thus, our camera dataset has 11 468 instances in the review table **R**. It has 5 647 users described by 25 attributes (tag topics) in the table **U**. And it has 654 cameras in the table **P**. Some indicative user attributes, i.e., tag topics, are the following: `wildlife` (tags: `birds`, `zoo`, etc.), `food` (tags: `fruit`, `market`, etc.), `sports` (tags: `car`, `tennis`, etc.), and so on. Those tag topics correspond to the questions required to build the ShoppingAdvisor tree.

Synthetic Dataset: We generate a large matrix of dimension 4 000 × 12 corresponding to the review table **R**. There are 200 products in the product table **P**, each having 20 boolean attributes and 1 000 users in the user table **U**, and 20 tags in the review table **R**. We randomly assign the products to the users for the 4 000 comments. We split the 20 independent and identically distributed attributes into four groups, where the value is set to 1 with probabilities of 0.75, 0.15, 0.10 and 0.05, respectively. For each of the 10 tags, we pre-define relations by randomly picking a set of attributes that are correlated to it. A tag is set to 1 with a certain probability if the majority of the attributes in its pre-defined relation have boolean value 1. In order to generate rating

scores for each of the 4 000 user-item interactions, we randomly distribute the 1 000 users into ten categories. For each user category, we pre-define relations by randomly picking a set of tags that are preferred by the users in that category. A user rates a product high, medium or low based on the proportion of his preferred tags in the tag vector corresponding to the product she has reviewed.

In order to evaluate our recommender system, we partition each of our datasets into training and test sets via ten-fold cross-validation. We use the training set for building our ShoppingAdvisor system, and measure its performance on the test set. We also consider a part of the training set as a validation set in order to optimize the model parameters, i.e., in order to post-prune the decision tree. The main evaluation metric that we use in our experiments to measure the quality of recommendations is mean reciprocal rank (MRR), which is a meaningful measure for single-item retrieval.

Mean reciprocal rank (MRR): In information retrieval MRR measures how far away from the first retrieved document the first relevant one is. In our datasets, with one relevant item per test user, we measure the recommendation quality by finding out how far from the top of the list the relevant item is. The reciprocal rank of a test user is the multiplicative inverse of the rank of the relevant item for that test user. The mean reciprocal rank is the average of the reciprocal ranks of results for all test cases

$$\mathsf{MRR} = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\mathsf{rank}_i},$$

where k is the number of instances in the test set and rank_i is the position of the i-th test user's relevant item in the ranked list of items returned. Note that, we partition our dataset into training and testing in such a way that the test set consists of users who have rated items high. In this way, for each user in the test set there is a highly relevant item and thus the MRR measure is meaningful.

Our second quantitative indicator is efficiency, i.e., running time for training; testing is fast as it only involves descending down the ShoppingAdvisor tree and receiving a recommendation at a leaf. The training time involves building the decision tree, as well as learning the ranking model for each possible split in a node. Clearly, the learning to rank module, i.e., RankSVM is an expensive operation especially since it requires to build the preference matrix each time. We employ techniques to pre-materialize part of the preference matrix, and thus reduce training time. We also present a detailed report of anecdotal evidence for real data in Section 6.4.2 as a qualitative validation of the effectiveness of the system.

Parameter Settings: In our experiments, we use Quinlan's C4.5 algorithm to build the decision tree. For RankSVM, we use Olivier Chappelle's RankSVM implementation, and we set the training error penalization parameter C to 0.001 for synthetic and car datasets, and 0.0001 for the camera dataset. The total number of instances in the car dataset available for partitioning into train and test set is around 10 000. We select a subset of the total data for training and testing (maintaining ten-fold cross-validation requirements) in order to avoid running out of memory. For the camera and the synthetic dataset, we work with the complete data.

To validate the effectiveness of ShoppingAdvisor, we compare its performance with baseline RankSVM.

RankSVM: This is a pairwise learning-to-rank algorithm that generates recommendations by learning item-feature weights. Our LearnSATree algorithm employs RankSVM for learning to rank, therefore we can consider this technique equivalent to the ranked list returned by ShoppingAdvisor at the root, before a potential shopper answers any question. We also compare the performance of k-NN with a variant, described below in order to demonstrate how ShoppingAdvisor is useful to existing recommendation techniques:

k-NN: This is standard collaborative-filtering algorithm that matches a test user to a set of users in the training set, and returns a ranked list of items by aggregating the item lists for the top neighbor users in the training set.

SA.k-NN: Feature selection and weighting has an important role in improving the effectiveness of a k-NN learner. ShoppingAdvisor allows k-NN to select a subset of features, i.e., tags by traversing down the tree, and then perform k-NN (user-based or item-based) on the reduced feature space. It is particularly useful since collaborative recommendation algorithms assume the availability of user preferences for matching similar users with similar interests, which may not be the case in reality. Moreover, user preferences drift rapidly over time and it is better to elicit user responses before making recommendations.

System Configuration: Our prototype system is implemented in Java and Matlab. All experiments were conducted on an Windows 7 machine with 2.30Ghz Intel i5 processor, 64 bit Operating System and 6GB RAM.

6.4.1 Quantitative Evaluation

Table 6.4 presents a comparison of recommendation quality, measured by average MRR over 10 folds, of ShoppingAdvisor with RankSVM, and k-NN with SA.k-NN for both the synthetic and the real datasets. The average height of the trees for the car, camera, and synthetic datasets are 15, 19 and 7, respectively. The average number of questions users in test set answer to receive their recommendation in the leaf nodes of ShoppingAdvisor are 12, 12 and 6, respectively.

We observe that leveraging shopper preferences clearly yields better quality recommendations (i.e., at the leaf nodes of ShoppingAdvisor) than those returned when the user does not answer any questions (i.e., RankSVM at the root node). The increment in quality is around 50% for both real datasets from root to leaf (see last two

Table 6.4. MRR comparisons of ShoppingAdvisor with RankSVM and k-NN with SA.k-NN for Car, Camera, and Synthetic datasets.

Dataset	RankSVM	ShoppingAdvisor	k-NN	SA.k-NN
Car	0.013	0.019	0.020	0.022
Camera	0.012	0.019	0.029	0.029
Synthetic	0.060	0.231	0.604	0.580

Table 6.5. Training time of ShoppingAdvisor for Car, Camera, and Synthetic datasets.

Dataset	Training Set Size	Features (tags)	Materialized Preference Pairs	Time (in sec)
Car	1900	15	36000	256
Camera	5000	25	50000	2168
Synthetic	3500	10	40000	1950

Table 6.6. Training time increases rapidly when exact preference matrix is considered (Synthetic dataset).

Training Set Size	Exhaustive Preference Pairs	Train Time (in sec)	MRR	Materialized Preference Pairs	Train Time (in sec)	MRR
50	965	0.948	0.045	483	0.726	0.045
100	3766	6.029	0.097	1883	1.472	0.093
150	8382	46.366	0.131	4191	2.549	0.128
200	14831	209.462	0.199	7416	3.375	0.196
250	23089	649.030	0.204	11545	5.927	0.203
300	33204	1526.300	0.200	16602	9.935	0.200

columns in Table 6.4). The average MRR score for k-NN and SA.k-NN are comparable for all three datasets. This indicates that SA.k-NN returns recommendations of very high quality to the users by asking a smaller number of questions than what the k-NN method would be asking. For the camera dataset, the k-NN method reaches a quality score of 0.029 by asking 25 questions (since the number of tag topics in the dataset is 25) while SA.k-NN achieves the same accuracy by asking only 12 questions, on average. Furthermore, an additional benefit of SA.k-NN compared to k-NN is that SA.k-NN does not require the recommender to be aware of the shopper preferences, and is therefore useful in handling new users.

Note that our ShoppingAdvisor system is also capable of handling new products, without existing reviews, by using their technical attributes, while the recommendations of k-NN are restricted to products with reviews in the training set. Finally, the tree structure in ShoppingAdvisor and SA.k-NN provides a logical explanation of the recommendations being returned, while the recommendations of k-NN are not easily interpretable.

Next, we present the time taken for training our system for synthetic and real datasets. The training time is the time taken to build the tree, which includes the time to execute RankSVM for all possible user attributes in a node. For example, in order to decide the splitting question for the root node from the pool of 15 questions in the car dataset, our algorithm has to perform RankSVM 30 times (15 for each children node). RankSVM is an expensive operation since it builds the preference matrix from the set of training instances belonging to a node.

The first section of Table 6.6 shows the increase in training time, averaged over 10 folds, with increase in number of training instances for synthetic data. MRR is obtained on a set of 20 test instances sampled from the synthetic data. Employing this method for the camera and car datasets, which have a few thousand instances in the training set, would be very expensive. Therefore, we materialize a preference matrix for all training instances in a dataset, and use that to acquire the preference pairs to be considered for a RankSVM operation. However, such materialized preference matrices for the camera and car datasets would take several days to build and would be several gigabytes in size. Therefore, for each item in training set, we make a random selection of preference pairs from the space of all training instances, instead of opting for all possible pairs. The second section of Table 6.6 shows the train time and the recommendation quality when 50% of the preference pairs are considered from the pool of all possible preference pairs, under the same framework settings.

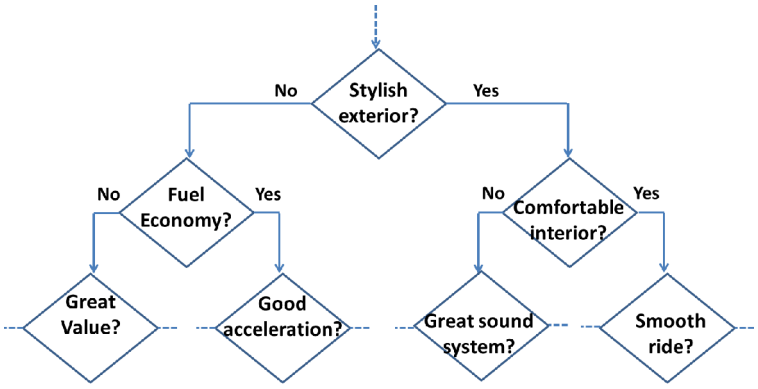


Figure 6.3. ShoppingAdvisor for the Car dataset.

Table 6.7. Example of top car recommendations at three nodes of tree in Figure 6.3

ShoppingAdvisor	Top Cars For Recommendation
`Stylish Exterior = NO`	Jeep Grand Cherokee SRT-8 Dodge Challenger SRT-8 Volvo XC60 AWD
`Stylish Exterior = NO` `Fuel Economy = NO`	Jeep Grand Cherokee SRT-8 Dodge Challenger SRT-8 Audi Q5 Premium quattro Tiptronic
`Stylish Exterior = NO` `Fuel Economy = YES`	Lincoln MKS 3.5L EcoBoost AWD Jeep Grand Cherokee SRT-8 Ford Escape Hybrid

We observe that while training time drops sharply, there is hardly any decrease in MRR. This indicates that we achieve faster training time without compromising RankSVM performance quality. When the number of training instances drops in the deeper level nodes, the number of preference pairs that can be retrieved from the materialized preference matrix may fall too, thereby requiring to generate the preference pairs at run time. On the other hand, classifying a test user is extremely fast. Table 6.5 shows the training time for the different datasets, along with the number of materialized preference pairs in input.

6.4.2 Qualitative Evaluation

Figures 6.3 and 6.4 present snapshots of ShoppingAdvisor trees built for the datasets car and camera, respectively. In Figure 6.3, a user is asked if they would

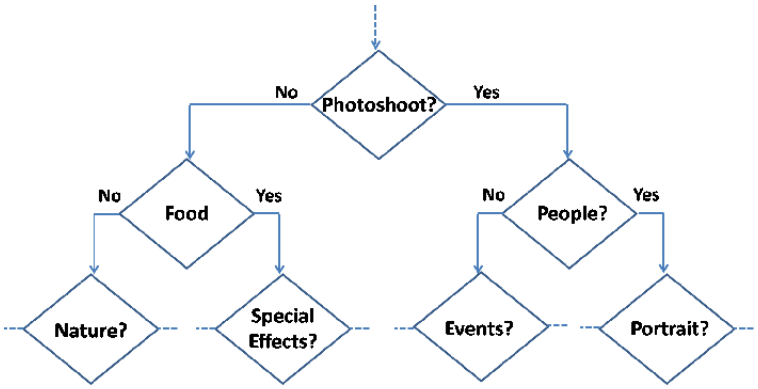


Figure 6.4. ShoppingAdvisor for the Camera dataset.

Table 6.8. Example of top camera recommendations at three nodes of tree in Figure 6.4.

ShoppingAdvisor	`Top Cameras For Recommendation`
`Photoshoot = YES`	Canon EOS Digital Rebel XS Nikon D80 Fujifilm FinePix S3
`Photoshoot = YES` `People = NO`	Canon EOS Digital Rebel XS Olympus E-3 Nikon D50
`Photoshoot = YES` `People = YES`	Canon EOS Digital Rebel XS Canon EOS 30D Nikon D80

like to buy a stylish car. A `yes` takes her to the next question about the interior of the car. If the user does not express any requirement for a comfortable and roomy interior, ShoppingAdvisor specifically asks if they are interested in a great audio system inside the car. On the other hand, a user who wants a car that has good fuel economy is immediately asked if acceleration is important, since fuel-efficient cars have slower acceleration.

Table 6.7 presents the top recommendations at three nodes of the tree shown in Figure 6.3. We observe that inclusion of the fuel economy condition brings in a hybrid car and an ecoboost car in the top recommendations, while exclusion of requirements of a fuel-efficient car makes ShoppingAdvisor recommend a Audi, which is known to compromise mileage for performance. Again, the presence of Jeep Grand Cherokee SRT-8 in all three nodes reflects its popularity in 2010.

Similarly in Figure 6.4, a user is asked if they are interested to buy a camera for photoshoot purposes, and if the photoshoot to be conducted is for people, and if it is focused on face shots. Thus, we see that ShoppingAdvisor narrows down the preferences of the shopper, and helps recommend cameras tailored to her needs. The camera recommendations at three nodes of the tree are shown in Table 6.8. For example, one of the top cameras recommended to her, Canon EOS 30D, introduced an auto image rotation feature in order to make better use of the LCD display especially during portrait-orientated shots. Again, for the shopper who is looking for a camera for shooting events and not people, ShoppingAdvisor recommends Olympus E-3 which happens to be a lightweight digital SLR camera. The presence of the Canon EOS Digital Rebel XS in all three nodes reflects its popularity among flickr members.

CHAPTER 7

Related Work

We now present related work in the literature that focuses on (i) sub-areas reasonably aligned to our research of exploratory mining of collaborative content; and (ii) techniques which we have adapted to solve some of our problems.

User feedback mining: In this dissertation, we have primarily focused on two forms of user feedback namely ratings and tags. Recall that, we extract short keywords, i.e., tags when user feedback is available in the form of reviews. Tag mining has been used in multiple applications including tag recommendations [49], item recommendations [50, 51], document navigation [52], and tagging motivation [53] However, most of these works are tailored to specific datasets and none of them defines a general mining problem, studies its complexity and develops efficient generic algorithms. There is a large body of work related to processing the text of reviews that focused on identifying sentiment and product features [54, 55, 56] or a combination thereof [57, 58]. In [58, 59], the authors extract individual product features and users' sentiments towards those features. Our problem of collaborative content analysis in Chapters 2 and 3 is clearly different from review mining and summarization [58, 60], which aims to produce a sentiment summary (i.e., positive or negative) using NLP and data mining techniques. Those studies are therefore tangential to our study and do not address the question of aggregating and interpreting ratings. The lone exception is [61], where the authors develop a method to label individual item reviews with the bias of a reviewer. Unlike our work, this study adopts a priori rules for comparing groups and does not provide an object measure based framework on

which formal optimization problems can be defined. The dynamics of social tagging has been an active research area in recent years, with several papers focusing on the tag prediction problem. A recent work [62] proposes a probabilistic model for personalized tag prediction and employs the Naive Bayes classifier. Related research in text mining [31] found that the Naive Bayes classifier performs better than SVM and CRF in classifying blog sentiments. Another study that indirectly supports the use of Naive Bayes for tag prediction is done by Heymann et al. [63], who found that tag-based association rules can produce very high-precision predictions. The process of collaborative tagging has been studied in [64]. While [65] discusses how to improve tag selection for tag clouds, we argue that tag clouds are inefficient methods to summarize the preferences of a set of user sub-population. Other related work investigates tag suggestion, usually from a collaborative filtering and UI perspective; for example with URLs [67] and blog posts [66]. Due to the popular adoption of recommendation systems by online sites such as Amazon and Netflix, explaining recommendations has also received significant attention. Herlocker et. al [68] provides a systematic study of explanations for recommendation systems, while [69] describes how explanations can be leveraged for recommendation diversification. Bilgic and Mooney [70] convincingly argues that the goal of a good explanation is not necessarily promotion, but to enable users to make well-informed decisions. Our study of user feedback interpretation in Chapters 2 and 3 is one step towards the ultimate goal of providing users with meaningful explanations to make informed decisions, and manufacturers with automatically mined knowledge to conduct better business.

Item designing and positioning: The problem of item design has been studied by many disciplines including economics, industrial engineering and computer science [71]. Optimal item design or positioning is a well studied problem in Operations

Research and Marketing. Shocker *et al.* [72] first represented products and consumer preferences as points in a joint attribute space. Later, several techniques [73, 74] were developed to design/position a new item. Work in this domain requires direct involvement of consumers, who choose preferences from a set of existing alternative products. Miah et al. [75] study the problem of selecting product snippets given a user query log, in order for the designed snippet to be returned by the maximum number of queries. However, none of these work has studied the problem of item design in relation to social rating and tagging, as we do in Chapter 4.

WEB ADVERTISEMENT AND SNIPPET: There has been a lot of work on web advertisement and snippet construction [38], most of which leverage text mining and natural language processing techniques to identify the top sentences to display [76] or in response to user search query [77], and evaluated by click-based metrics [82]. There are several research challenges associated with finding the best ad, such as designing the appropriate mathematical optimization model to maximize value for users, advertisers, publishers, etc. [78][79]; integrating user's past browsing history and behavioral attributes [36]; revamping ad selection technology to bring relevance and appropriateness of ads closer to the other content served on the web [80]; improving advertiser's experience by increasing the value that the advertiser gets from online advertising [81]; minimizing expected user effort to comprehend snippets [37], etc. We consider the novel task of snippet generation in Chapter 5, i.e., identifying the most informative item features to highlight in snippet, by leveraging collaborative tagging feedback. While faceted search employed by sites like Amazon, eBay, etc. performs pre-defined top-down navigation on the concept hierarchy, where all features of the currently selected concept are displayed, our objective in Chapter 5 is to highlight the important features.

Recommendation in e-commerce: Recommendation is traditionally formulated as the problem of estimating ratings for items that have not been seen by a user [83]. Once these ratings are estimated, a recommendation is built by picking the items with highest rating. A more recent formulation makes this assumption explicit, and casts the recommendation task as a ranking problem [84]. Given this formulation, techniques from the learning-to-rank literature can be applied to learn personalized ranking functions. Application of recommender systems to e-commerce dates back to the '90s [85]. While early works often mined transaction logs, more recent works focus on user ratings, especially for digital media like movies and news. The explosion of the size of digital catalogues has made recommendation a necessity. [86] provide an overview of recommender systems and their categorization into content-based, collaborative filtering and hybrid. Amazon's "customers who bought" feature is based on co-buying information [87]. Similarly, [88] create a recommender system for e-commerce based on mining transaction logs that uses a probabilistic graphical model to handle skew and sparseness in data. Our work in Chapter 6 does not rely on transaction logs and only takes into account implicit or explicit user feedback. Our system bears some resemblance to Bayesian networks, where each node is a decision tree and each edge represents consumer information [89]. In our case the network is a decision tree with sharp constraints, consumer information is elicited explicitly via questions and nodes are instances of a learning-to-rank algorithm. As the recommendations provided by our system derive from eliciting user preferences, they are also easy to explain to the user. Explanation of recommendation has been shown to be effective in increasing the acceptance of recommendations [68]. Our recommendation system naturally lends itself to a keyword-style explanation, which has been found to be the most effective kind of explanation [70]. [90] present an interactive system that is similar to ours in spirit. The system recommends technical products by a asking

a series of questions that start from high-level needs of the user and transition to technical features of the item. Users are clustered into "target groups" according to their similarity in evaluation structure. While the idea is similar to ours, the authors only describe the interaction between the system and the user. The paper does not address how such a system would be built and which algorithm would power the recommender system. Indeed, our system is able to automatically build a question tree that could drive the interaction described by the authors.

OLAP AND DATA ANALYTICS OVER DATABASES: Our idea of using structurally meaningful cuboids as the basis for user feedback interpretation is inspired by studies in data cube mining, first proposed in Gray et. al [8] and Ramakrishnan et. al [91]. Most of those studies, however, focus on how to efficiently compute aggregate measures for all cuboids and are therefore orthogonal to our approach in Chapters 2 and 3. Keyword-Driven Analytical Processing (KDAP) [92] combines intuitive keyword-based search with the power of aggregation in OLAP, without having to spend considerable effort in organizing the data. This technique focuses on ranking results in dynamic facets to allow users to explore the aggregation space efficiently. Our work differs from KDAP in that it aims to retrieve the global top-k item rating interpretations covering all dimensions, without giving precedence to one dimension over another. Sathe and Sarawagi's work [93] on intelligent roll-ups in multidimensional OLAP data proposes a cost-based operator that automatically returns easy-to-interpret summaries of all possible maximal generalizations along various roll-up paths around a specific problem. Quotient Cube [94] provides a succinct summary of the data cube, preserving roll-up/dill-down semantics and grouping similar cuboids/scenarios (identical aggregate values) into classes. We, in turn, seek to find the top-k interpretations that best describe an item rating set. Moreover, the consideration that adjacent

cuboids share common aggregate values frequently [94] is not always true in our context. However, none of these adopts formal objective measures based on user ratings like in our work. To the best of our knowledge, our work in Chapter 2 is the first to leverage structurally meaningful descriptions for collaborative rating analysis. A recent work [95] introduces the problem of promotion analysis in a multi-dimensional space. The authors propose a compact cube structure that quickly locates promotive subspaces and effectively prunes out un-interesting subspaces for a target object pertaining to the promotion query. We focus on the problem of item rating interpretation in a multi-dimensional cube space.

TOP-K ALGORITHMS: Our top-k pipelined algorithm is inspired by the rich work on top-k algorithms ([32, 34]). A recent survey by Ilyas et al. [96] covers many of the important results in this area. The classic setting of these works is that each list contains an attribute of an object and a monotone aggregate function is used for ranking. The top tier of our pipelined top-k algorithm in Chapter 4 is adapted from this setting, where each list has the probability of a tag for each assignment of attribute values. In contrast, in the bottom tier of our algorithm, an entry from one list can match with any entry from the other lists. This setting is adapted from the problem of top-k join [34]. The top-k techniques in Chapter 5 are also inspired by [34].

LEARNING TO RANK: Our framework in Chapter 6 uses a learning-to-rank algorithm as a basic building block to generate recommendations. We chose to use SVM-rank [47], but other algorithms like RankBoost [97] would be equally viable. Probabilistic Boost Tree (PBT) [98] works in a way similar to our system. PBT builds a tree in which each node is an instance of a boosting algorithm. The tree is expanded by splitting the training instances based on the classification performed at each node.

However, the goal of the two systems is different: PBT performs classification while our algorithm performs ranking. TreeRank [99] uses a tree-based algorithm to solve the bipartite ranking problem, where each element has a binary relevance label and the goal is to rank all relevant items on top. However, in our case binary relevance is not adequate to express complex user preferences.

Locality sensitive hashing: Locality Sensitive Hashing (LSH) was first introduced in [17, 20] for nearest-neighbor and approximate nearest-neighbor search in a high-dimensional vector space. In [18], Charikar constructed the LSH function for cosine similarity, which supports fast similarity between two high-dimensional vectors by reducing them to bit-arrays of much smaller dimensions. This result has been used in several problems, including efficient noun clustering and duplicate detection. In our work in Chapter 3, we show how we adapt LSH to rank and choose the best bucket containing our collaborative content analysis results.

Facility dispersion problem:: Facility Dispersion Problem (FDP) was first introduced in [100]. It deals with the location of facilities on a network in order to maximize distances between facilities, minimize transportation costs, avoid placing hazardous materials near housing, outperform competitors facilities, etc. One of the foremost work to map the objective of diversification to those used in facility dispersion is [101]. In our work in Chapter 3 as well as in Chapter 5, we use the problem variant in [27] to serve the tag diversity problem instantiations, and also explain how the same may be extended to solve similarity problems.

CHAPTER 8

Conclusions

8.1 Summary of Contributions

We develop a framework for collaborative social content mining and identify a family of novel exploratory mining problems that benefit experience and decision-making of content producers and consumers. To the best of our knowledge, how user and item attributes influence social rating and tagging behavior have not been studied in the context of collaborative user-generated content before. We formalize the problems, majority of which are proved to be NP-Complete, and design a suite of algorithms - exact, approximation with theoretical properties, and efficient heuristics - that account for the technical challenges associated with the tasks and solve them. We perform detailed evaluation over synthetic data and real data crawled from the web order to validate the effectiveness of our algorithms and utility of our framework.

8.2 Research Frontiers

Exploratory mining of collaborative social content is a reasonably young and promising research field, and offers several exciting future research directions. We now illustrate few of them:

8.2.1 Medium Term Research Agenda

We first present medium term goals for future work that builds on the work presented in the dissertation.

(i) Beyond Traditional Marketplace: One immediate research goal is pushing the boundaries of information management in social media beyond traditional marketplaces to foster social betterment , as discussed below:

- Healthcare: A survey of U.S. consumers by PwC in 2012 reveal that a third of U.S. consumers use social media sites such as Facebook and Twitter to seek medical information and track and share symptoms[1]. Another report by Pew Internet & American Life Project in 2012 reveals that 72% of internet users look online for health information[2]. The number of niche social networking sites dedicated to healthcare professionals and consumers has proliferated too. Consumers engaging in health information seeking via the Internet want (i) to find information quickly, and (ii) to have information streamlined. Therefore, a collaborative content mining framework capable of providing the right abstraction of pertinent information is becoming increasingly important, especially since more than 70,000 websites providing health care information are available today. It will facilitate consumers evaluate and leverage other users knowledge, experience and feedback before making health care decision. Collaborative content, which largely consists of user feedback on items (i.e., health services) can also help healthcare providers, practitioners and pharmaceutical companies review and improve their quality of service.

- Recreation: The rise of Web 2.0 has led to a worldwide increase in socially connected people, along with increase in interconnectivity and interactivity of web-delivered content such as blogs, videos, photos, etc. Significant innovations emerge from collaborations and creative pursuit cannot be an exception. Therefore, the vast amount of user-generated opinion available in collaborative content sites can be readily explored to enable the discovery of new work. For example, an artist creating a new musical piece can leverage the ratings and tags that users have selected for existing popular songs, in order to identify the pieces attributes (e.g. acoustic, audio features, theme, etc.) that will increase its chances of becoming popular. Similarly, a blogger or a

[1]http://www.pwc.com/us/en/health-industries/publications/health-care-social-media.jhtml
[2]http://pewinternet.org/Commentary/2011/November/Pew-Internet-Health.aspx

photo-journalist can select a topic based on the tags that other popular topics have received. Besides new item (i.e., new music, new blog, new photo, etc.) creation, user feedback behavior mining can also help viral marketing of new content across the web, positioning of news articles in sites in order to draw maximum traffic to the news sites, etc.

8.2.2 Long Term Research Agenda

Last, we present the long-term goals of our research.

(i) Leveraging user-user interaction: In this dissertation, we have primarily focused on user-item interaction in the form of short user feedback for web items. However, the rise of social media and social networking sites has triggered the evolution of user-user interaction, along with user-item interaction. User-user interaction exists in the form of friendship links in social networking sites, responses to content shared by other people in the friendship network, etc. For example, an average Facebook user has 130 friends in her network, and clicks the like button 9 times (which is an user-user interaction) per month, and writes 25 comments per month (which is another user-user interaction)[3]. Recent research has seen the emergence of applications that leverages such user-user interactions in social networks, since such meaningful interactive relationships are critical to improving trust and reliability in the content. Our long term goal includes leveraging user-user interactions, in conjunction with user-item interactions for effective knowledge-rich mining of collaborative social content.

(ii) Towards a general framework: Our ultimate goal is to develop a general collaborative content mining system that caters to any community featuring two-way communication between producers and consumers (e.g., manufacturer and buyer, healthcare

[3]www.facebook.com/press/info.php?statistics

service provider and patient, musical artist and listener, blogger and reader, etc.) and handles all kinds of mining problems in the context of collaborative social content sites. The mining problems and the solutions presented in this dissertation can be extended to handle most other related problems, since they provide the basis for modeling correlation between user feedback and item attributes and user attributes. However, we intend to investigate the scope of additional exploratory mining problems in order to broaden the scope of the framework. It would also be interesting to collaborate the different types of collaborative content for mining interesting insight from social media (e.g., use demographics driven healthcare information to help recommend vacation packages). In addition to the original research challenges (i.e., information explosion, information overload, and user-item interaction intractability), building such a generic framework would introduce additional challenges such as data cleansing to handle the huge volume of inaccurate information the internet contains (especially, for healthcare social content mining), data integration from multiple heterogeneous sources, etc.

REFERENCES

[1] M. Das, S. Amer-Yahia, G. Das, and C. Yu, "Mri : Meaningful interpretations of collaborative ratings," *PVLDB*, vol. 4, no. 10, pp. 1063–1074, 2011.

[2] S. Thirumuruganathan, M. Das, S. Desai, S. Amer-Yahia, G. Das, and C. Yu, "Maprat: Meaningful explanation, interactive exploration and geo-visualization of collaborative ratings," *PVLDB*, vol. 5, no. 12, pp. 1986–1989, 2012.

[3] M. Das, S. Thirumuruganathan, S. Amer-Yahia, G. Das, and C. Yu, "Who tags what? an analysis framework," *PVLDB*, vol. 5, no. 11, pp. 1567–1578, 2012.

[4] M. Das, S. Thirumuruganathan, S. Amer-Yahia, G. Das, and C. Yu, "An expressive framework and efficient algorithms for the analysis of collaborative tagging," *The VLDB Journal*, pp. 1–26, 2013.

[5] M. Das, G. Das, and V. Hristidis, "Leveraging collaborative tagging for web item design," in *KDD*, 2011, pp. 538–546.

[6] M. Das, H. Rahman, G. Das, and V. Hristidis, "Generating informative snippet to maximize item visibility," in *CIKM*, 2013, pp. 1721–1726.

[7] M. Das, G. D. F. Morales, A. Gionis, and I. Weber, "Learning to question: leveraging user preferences for shopping advice," in *KDD*, 2013, pp. 203–211.

[8] J. Gray, A. Bosworth, A. Layman, D. Reichart, and H. Pirahesh, "Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals," in *ICDE*, 1996, pp. 152–159.

[9] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is np-complete," *Information Processing Letters*, vol. 5, no. 1, pp. 15 – 17, 1976.

[10] D. E. Vaughan, S. H. Jacobson, and H. Kaul, "Analyzing the performance of simultaneous generalized hill climbing algorithms," *Computational Optimization and Applications*, vol. 37, pp. 103–119, 2007.

[11] S. Acharya, P. B. Gibbons, and V. Poosala, "Congressional samples for approximate answering of group-by queries," in *SIGMOD*, 2000, pp. 487–498.

[12] T. A. Slocum, R. B. MacMaster, F. C. Kessler, and H. H. Howard, *Thematic cartography and geovisualization, 3rd Edition*. Pearson Education, 2009.

[13] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[14] S. Amer-Yahia, J. Huang, and C. Yu, "Jelly: A language for building community-centric information exploration applications," in *ICDE*, 2009, pp. 1588–1594.

[15] Y. Chen, F. M. Harper, J. A. Konstan, and S. X. Li, "Social comparisons and contributions to online communities: A field experiment on movielens," in *Computational Social Systems and the Internet*, 2007.

[16] D. S. Johnson, "The np-completeness column: An ongoing guide," *Journal of Algorithms*, vol. 8, no. 3, pp. 438–448, 1987.

[17] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *STOC*, 1998, pp. 604–613.

[18] M. Charikar, "Similarity estimation techniques from rounding algorithms," in *STOC*, 2002, pp. 380–388.

[19] M. X. Goemans and D. P. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *Journal of the ACM*, vol. 42, no. 6, pp. 1115–1145, 1995.

[20] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *VLDB*, 1999, pp. 518–529.

[21] M. Slaney, Y. Lifshits, and J. He, "Optimal parameters for locality-sensitive hashing," *Proceedings of the IEEE*, vol. 100, no. 9, pp. 2604–2623, 2012.

[22] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li, "Modeling lsh for performance tuning," in *CIKM*, 2008, pp. 669–678.

[23] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *VISAPP (1)*, 2009, pp. 331–340.

[24] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[25] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Communications of the ACM*, vol. 51, no. 1, pp. 117–122, 2008.

[26] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: Efficient indexing for high-dimensional similarity search," in *VLDB*, 2007, pp. 950–961.

[27] S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi, "Facility dispersion problems: Heuristics and special cases (extended abstract)," in *WADS*, 1991, pp. 355–366.

[28] E. Erkut, T. Baptie, and B. V. Hohenbalken, "The discrete p-maxian location problem," *Computers & Operations Research*, vol. 17, no. 1, pp. 51–61, 1990.

[29] F. Murtagh, "A Survey of Recent Advances in Hierarchical Clustering Algorithms," *The Computer Journal*, vol. 26, no. 4, pp. 354–359, 1983.

[30] C. D. Manning, P. Raghavan, and H. Schtze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[31] A. Pak and P. Paroubek, "Twitter as a corpus for sentiment analysis and opinion mining," in *LREC*, 2010.

[32] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," in *PODS*, 2001.

[33] B. Cestnik, "Estimating probabilities: A crucial task in machine learning," in *ECAI*, 1990, pp. 147–149.

[34] I. F. Ilyas, D. Martinenghi, and M. Tagliasacchi, "Rank-join algorithms for search computing," in *SeCO Workshop*, 2009, pp. 211–224.

[35] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

[36] J. Yan, N. Liu, G. Wang, W. Zhang, Y. Jiang, and Z. Chen, "How much can behavioral targeting help online advertising?" in *WWW*, 2009, pp. 261–270.

[37] A. Kashyap and V. Hristidis, "Comprehension-based result snippets," in *CIKM*, 2012, pp. 1075–1084.

[38] R. White, I. Ruthven, and J. M. Jose, "Finding relevant documents using top ranking sentences: an evaluation of two alternative schemes," in *SIGIR*, 2002, pp. 57–64.

[39] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[40] L. Qin, J. X. Yu, and L. Chang, "Diversifying top-k results," *PVLDB*, vol. 5, no. 11, pp. 1124–1135, 2012.

[41] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong, "Diversifying search results," in *WSDM*, 2009, pp. 5–14.

[42] I. F. Ilyas, D. Martinenghi, and M. Tagliasacchi, "Rank-join algorithms for search computing," in *SeCO Workshop*, 2009, pp. 211–224.

[43] S. Ravi, D. Rosenkrantz, and G. Tayi, "Facility dispersion problems: Heuristics and special cases," in *WADS*, 1991, pp. 355–366.

[44] R. G. Downey and M. R. Fellows, "Fixed-parameter tractability and completeness ii: On completeness for w[1]," *Theoretical Computer Science*, vol. 141, no. 1&2, pp. 109–131, 1995.

[45] G. Zuccon, L. Azzopardi, D. Zhang, and J. Wang, "Top-k retrieval using facility location analysis," in *ECIR*, 2012, pp. 305–316.

[46] S. Gollapudi and A. Sharma, "An axiomatic framework for result diversification," *IEEE Data Engineering Bulletin*, vol. 32, no. 4, 2009.

[47] T. Joachims, "Training linear svms in linear time," in *KDD*, 2006, pp. 217–226.

[48] W. W. Cohen, R. E. Schapire, and Y. Singer, "Learning to order things," *Journal of Artificial Intelligence Research*, vol. 10, pp. 243–270, 1999.

[49] K. Liu, B. Fang, and W. Zhang, "Speak the same language with your friends: augmenting tag recommenders with social relations," in *HT*, 2010, pp. 45–50.

[50] H. Liang, Y. Xu, Y. Li, R. Nayak, and X. Tao, "Connecting users and items with weighted tags for personalized item recommendations," in *HT*, 2010, pp. 51–60.

[51] Y. Guo and J. B. D. Joshi, "Topic-based personalized recommendation for collaborative tagging system," in *HT*, 2010, pp. 61–66.

[52] J. Gwizdka, "Of kings, traffic signs and flowers: exploring navigation of tagged documents," in *HT*, 2010, pp. 167–172.

[53] C. Körner, R. Kern, H.-P. Grahsl, and M. Strohmaier, "Of categorizers and describers: an evaluation of quantitative measures for tagging motivation," in *HT*, 2010, pp. 157–166.

[54] K. Dave, S. Lawrence, and D. M. Pennock, "Mining the peanut gallery: opinion extraction and semantic classification of product reviews," in *WWW*, 2003, pp. 519–528.

[55] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up sentiment classification using machine learning techniques," *CoRR*, vol. cs.CL/0205070, 2002.

[56] B. Snyder and R. Barzilay, "Multiple aspect ranking using the good grief algorithm," in *HLT-NAACL*, 2007, pp. 300–307.

[57] N. Archak, A. Ghose, and P. G. Ipeirotis, "Show me the money!: deriving the pricing power of product features by mining consumer reviews," in *KDD*, 2007, pp. 56–65.

[58] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *KDD*, 2004, pp. 168–177.

[59] A.-M. Popescu and O. Etzioni, "Extracting product features and opinions from reviews," in *HLT/EMNLP*, 2005.

[60] L. Zhuang, F. Jing, and X.-Y. Zhu, "Movie review mining and summarization," in *CIKM*, 2006, pp. 43–50.

[61] S. Amer-Yahia, A. Z. Broder, and A. Galland, "Reviewing the reviewers: Characterizing biases and competencies using socially meaningful attributes," in *AAAI Workshop on Social Information Processing (SIP)*, 2008.

[62] D. Yin, Z. Xue, L. Hong, and B. D. Davison, "A probabilistic model for personalized tag prediction," in *KDD*, 2010, pp. 959–968.

[63] P. Heymann, D. Ramage, and H. Garcia-Molina, "Social tag prediction," in *SIGIR*, 2008, pp. 531–538.

[64] S. A. Golder and B. A. Huberman, "Usage patterns of collaborative tagging systems," *Journal of Information Science*, vol. 32, no. 2, pp. 198–208, 2006.

[65] P. Venetis, G. Koutrika, and H. Garcia-Molina, "On the selection of tags for tag clouds," in *WSDM*, 2011, pp. 835–844.

[66] G. Mishne, "Autotag: a collaborative approach to automated tag assignment for weblog posts," in *WWW*, 2006, pp. 953–954.

[67] Z. Xu, Y. Fu, J. Mao, and D. Su, "Towards the semantic web: Collaborative tag suggestions," in *Proceedings of the Collaborative Web Tagging Workshop at the WWW 2006*, 2006.

[68] J. L. Herlocker, J. A. Konstan, and J. Riedl, "Explaining collaborative filtering recommendations," in *CSCW*, 2000, pp. 241–250.

[69] C. Yu, L. V. S. Lakshmanan, and S. Amer-Yahia, "It takes variety to make a world: diversification in recommender systems," in *EDBT*, 2009, pp. 368-378.

[70] M. Bilgic and R. J. Mooney, "Explaining recommendations: Satisfaction vs. promotion," in *Beyond Personalization (workshop at IUI)*, 2005.

[71] T. Selker and W. Burleson, "Context-aware design and interaction in computer systems," *IBM Systems Journal*, vol. 39, no. 3&4, pp. 880–891, 2000.

[72] A. D. Shocker and V. Srinivasan, "A consumer-based methodology for the identification of new product ideas," *Management Science*, vol. 20, no. 6, pp. 921–937, 1974.

[73] S. Albers and K. Brockhoff, "Optimal product attributes in single choice models," *Journal of the Operational Research Society*, vol. 31, pp. 647–655, 1980.

[74] M. D. Albritton and P. R. McMullen, "Optimal product design using a colony of virtual ants," *European Journal of Operational Research*, vol. 176, no. 1, pp. 498–520, 2007.

[75] M. Miah, G. Das, V. Hristidis, and H. Mannila, "Determining attributes to maximize visibility of objects," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, pp. 959–973, 2009.

[76] A. Turpin, Y. Tsegay, D. Hawking, and H. E. Williams, "Fast generation of result snippets in web search," in *SIGIR*, 2007, pp. 127–134.

[77] Y. Huang, Z. Liu, and Y. Chen, "Query biased snippet generation in xml search," in *SIGMOD*, 2008, pp. 315–326.

[78] A. Z. Broder, "Computational advertising," in *SODA*, 2008, p. 992.

[79] A. Z. Broder, M. Fontoura, V. Josifovski, and L. Riedel, "A semantic approach to contextual advertising," in *SIGIR*, 2007, pp. 559–566.

[80] F. Radlinski, A. Z. Broder, P. Ciccolo, E. Gabrilovich, V. Josifovski, and L. Riedel, "Optimizing relevance and revenue in ad search: a query substitution approach," in *SIGIR*, 2008, pp. 403–410.

[81] A. Ghosh and M. Mahdian, "Externalities in online advertising," in *WWW*, 2008, pp. 161–168.

[82] Z. Li, D. Zhou, Y.-F. Juan, and J. Han, "Keyword extraction for social snippets," in *WWW*, 2010, pp. 1143–1144.

[83] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: An open architecture for collaborative filtering of netnews," in *CSCW*, 1994, pp. 175–186.

[84] S. Balakrishnan and S. Chopra, "Collaborative ranking," in *WSDM*, 2012, pp. 143–152.

[85] J. B. Schafer, J. Konstan, and J. Riedi, "Recommender systems in e-commerce," in *EC*, 1999, pp. 158–166.

[86] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[87] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, 2003.

[88] C.-N. Hsu, H.-H. Chung, and H.-S. Huang, "Mining skewed and sparse transaction data for personalized shopping recommendation," *Machine Learning*, vol. 57, no. 1-2, pp. 35–59, 2004.

[89] J. S. Breese, D. Heckerman, and C. M. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," *CoRR*, vol. abs/1301.7363, 2013.

[90] M. Stolze and M. Ströbel, "Recommending as Personalized Teaching," in *Designing Personalized User Experiences in eCommerce*. Springer, 2004, vol. 5, pp. 293–313.

[91] R. Ramakrishnan and B.-C. Chen, "Exploratory mining in cube space," *Data Mining and Knowledge Discovery*, vol. 15, no. 1, pp. 29–54, 2007.

[92] P. Wu, Y. Sismanis, and B. Reinwald, "Towards keyword-driven analytical processing," in *SIGMOD*, 2007, pp. 617–628.

[93] G. Sathe and S. Sarawagi, "Intelligent rollups in multidimensional olap data," in *VLDB*, 2001, pp. 531–540.

[94] L. V. S. Lakshmanan, J. Pei, and J. Han, "Quotient cube: how to summarize the semantics of a data cube," in *VLDB*, 2002, pp. 778–789.

[95] T. Wu, D. Xin, Q. Mei, and J. Han, "Promotion analysis in multi-dimensional space," *PVLDB*, vol. 2, no. 1, pp. 109–120, 2009.

[96] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-k query processing techniques in relational database systems," *ACM Computing Surveys*, vol. 40, no. 4, pp. 1–58, 2008.

[97] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *Journal of Machine Learning Research*, vol. 4, pp. 933–969, 2003.

[98] Z. Tu, "Probabilistic Boosting-Tree: Learning Discriminative Models for Classification, Recognition, and Clustering," in *ICCV*, 2005, pp. 1589–1596.

[99] S. Clémençon and N. Vayatis, "Tree-based ranking methods," *IEEE Transactions on Information Theory*, vol. 55, no. 9, pp. 4316–4336, 2009.

[100] G. Y. Handler and P. B. Mirchandani, *Location on networks: theory and algorithms*, ser. MIT Press series in signal processing, optimization, and control. The MIT Press, 1979.

[101] S. Gollapudi and A. Sharma, "An axiomatic approach for result diversification," in *WWW*, 2009, pp. 381–390.

BIOGRAPHICAL STATEMENT

Mahashweta Das was born in Calcutta, India in 1984. She received her Bachelor's and Master's degree in Computer Science and Engineering from Jadavpur University, India in 2007 and the Ohio State University, USA in 2009 respectively. She received Doctor of Philosophy in Computer Science from The University of Texas at Arlington in 2013. While her MS thesis focuses on spatial and temporal mining, her PhD research under Dr. Gautam Das concerns information management in social and collaborative media. Her research interests include large-scale data mining, databases, machine learning, algorithms and social computing. She visited IBM Research Lab in summer 2009 as a research intern in the Information Management group. She also interned in the Recommendation Group at Technicolor Research Lab during summer 2011 and in the Web Mining Group at Yahoo! Research during summer 2012. She has published refereed articles in SIGKDD, VLDB among others, and her paper titled "Who Tags What? An Analysis Framework" has been selected for the Best Papers of VLDB 2012. She is the winner of the Sensor KDD 2009 challenge for knowledge discovery from climate data. She received the John S. Schuchman Outstanding PhD Student Award 2013 by the College of Engineering, University of Texas at Arlington, as well as the Graduate School Summer 2013 Dean's Dissertation Fellowship. She received the TCS-JU Best Student Award for 2003-2007 by Department of Computer Science and Engineering, Jadavpur University. She is also a finalist of Google 2012 Anita Borg Memorial Scholarship: USA.