Summer 2012

Latent feature networks for statistical relational learning

Mohammad Khoshneshin
University of Iowa

LATENT FEATURE NETWORKS FOR STATISTICAL RELATIONAL LEARNING

by

Mohammad Khoshneshin

An Abstract

Of a thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Business Administration
in the Graduate College of
The University of Iowa

July 2012

Thesis Supervisor: Professor W. Nick Street

ABSTRACT

In this dissertation, I explored relational learning via latent variable models. Traditional machine learning algorithms cannot handle many learning problems where there is a need for modeling both relations and noise. Statistical relational learning approaches emerged to handle these applications by incorporating both relations and uncertainties in these problems. Latent variable models are one of the successful approaches for statistical relational learning. These models assume a latent variable for each entity and then the probability distribution over relationships between entities is modeled via a function over latent variables. One important example of relational learning via latent variables is text data modeling. In text data modeling, we are interested in modeling the relationship between words and documents. Latent variable models learn this data by assuming a latent variable for each word and document. The co-occurrence value is defined as a function of these random variables. For modeling co-occurrence data in general (and text data in particular), we proposed latent logistic allocation (LLA). LLA outperforms the-state-of-the-art model — latent Dirichlet allocation — in text data modeling, document categorization and information retrieval. We also proposed query-based visualization which embeds documents relevant to a query in a 2-dimensional space. Additionally, I used latent variable models for other single-relational problems such as collaborative filtering and educational data mining.

To move towards multi-relational learning via latent variable models, we propose latent feature networks (LFN). Multi-relational learning approaches model multiple relationships simultaneously. LFN assumes a component for each relationship. Each component is a latent variable model where a latent variable is defined for each entity and the relationship is a function of latent variables. However, if an entity participates in more than one relationship, then it will have a separate random variable for each relationship. We used LFN for modeling two different problems: microarray classification and social network analysis with a side network. In the first application, LFN outperforms support vector machines — the best propositional model for that application. In the second application, using the side information via LFN can drastically improve the link prediction task in a social network.

LATENT FEATURE NETWORKS

FOR STATISTICAL RELATIONAL LEARNING

by

Mohammad Khoshneshin

A thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Business Administration
in the Graduate College of
The University of Iowa

July 2012

Thesis Supervisor: Professor W. Nick Street

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Mohammad Khoshneshin

has been approved by the Examining Committee for the thesis requirement for the Doctor of Philosophy degree in Business Administration at the July 2012 graduation.

Thesis committee: ______________________________
W. Nick Street, Thesis Supervisor

____
Samuel Burer

____
Jeffrey Ohlmann

____
Gautam Pant

____
Padmini Srinivasan

To my parents, Sima and Rahim,
and my wife, Mahtab

ACKNOWLEDGEMENTS

I owe my deepest gratitude to my PhD supervisor, Dr. Nick Street, who helped me through all stages of my doctoral studies. Not only he was my advisor in research, but he also was a friend who supported me tremendously during these past years. He provided me with latitude to select the research direction that I was most passionate about, insuring that abundant technical and intuitive guidance was within my reach. I also would like to express my gratitude to my committee members: Drs. Padmini Srinivasan, Samuel Burer, Jeffery Ohlmann, and Guatam Pant. This dissertation would not has been possible without their fruitful ideas, constructive comments, and warm support.

I would like to thank my colleagues, Lian Duan and Michael Rechenthin, for the many helpful discussions and endless laughs that we shared. I have also had the pleasure of working alongside Viet Ha-Thuc, Si-Chi Chin, Dengfeng Zhang, Justin Goodson, Kamal Lamsal, Amit Kumar Verma, Nicholas Leifker, Senay Yasar Saglam, Ray Hylock, Brian Almquist, Chris Harris, Jieqiu Chen, Wenjun Wang, Bob Arens, and Chen Yang, whose camaraderie was a privilege I enjoyed on several occasions. I am grateful to the DMIG group members who challenged me to think deeper and wider about my research. I would like to thank Barbara Carr and Renea Jay for all their assistance during my time at U Iowa. I greatly appreciate the support of my Iowa City friends: Mohammad Ahmadi Basir, Niloufar Salimi, Hamid Fahimrezaei, and Raheleh Mohammad.

I would like to express my sincere appreciation to my parents, Sima Sharifi and Rahim Khoshneshin. Their unconditional love, belief, and support has always given me courage and direction. I am in their debt for all I have or will have in life. I also would like to thank my brothers, Puya and Milad, and my parents-in-law, Mehrnaz Yamini and Ali Akbar Ghazizadeh, for their support and love. Last but not least, I wish to thank my beloved wife, Mahtab, for her love, patience, and support during these years. She encouraged me when I was hopeless and supported me when I was frustrated. I truly believe this dissertation could not have been possible without her by my side.

ABSTRACT

In this dissertation, I explored relational learning via latent variable models. Traditional machine learning algorithms cannot handle many learning problems where there is a need for modeling both relations and noise. Statistical relational learning approaches emerged to handle these applications by incorporating both relations and uncertainties in these problems. Latent variable models are one of the successful approaches for statistical relational learning. These models assume a latent variable for each entity and then the probability distribution over relationships between entities is modeled via a function over latent variables. One important example of relational learning via latent variables is text data modeling. In text data modeling, we are interested in modeling the relationship between words and documents. Latent variable models learn this data by assuming a latent variable for each word and document. The co-occurrence value is defined as a function of these random variables. For modeling co-occurrence data in general (and text data in particular), we proposed latent logistic allocation (LLA). LLA outperforms the-state-of-the-art model — latent Dirichlet allocation — in text data modeling, document categorization and information retrieval. We also proposed query-based visualization which embeds documents relevant to a query in a 2-dimensional space. Additionally, I used latent variable models for other single-relational problems such as collaborative filtering and educational data mining.

To move towards multi-relational learning via latent variable models, we propose latent feature networks (LFN). Multi-relational learning approaches model multiple relationships simultaneously. LFN assumes a component for each relationship. Each component is a latent variable model where a latent variable is defined for each entity and the relationship is a function of latent variables. However, if an entity participates in more than one relationship, then it will have a separate random variable for each relationship. We used LFN for modeling two different problems: microarray classification and social network analysis with a side network. In the first application, LFN outperforms support vector machines — the best propositional model for that application. In the second application, using the side information via LFN can drastically improve the link prediction task in a social network.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

The goal of artificial intelligence and machine learning approaches is accomplishing tasks automatically. Many of these tasks can be done by a human, while the limitation of human computational power limits performing them in large scale. For example, analyzing a picture and distinguishing the objects in a picture might be an easy task for a human and hard for machine, but what if we want to process thousands of pictures? Retrieving relevant documents to a query is still an easy task for a human but when the domain of documents is the web, no one is able to be comprehensive. Therefore, a machine with human intelligence — or at least close — and great computational power — compared to humans — will open an array of wonderful possibilities. Among the different aspects of modern life, many tools such as Internet search engines, recommender systems, and intelligent transportation systems have enjoyed the efforts in this direction — towards intelligent machines.

Traditional machine learning approaches can be divided into two separate schools — logic-based models and propositional models. Logic-based models represent knowledge using logic languages such as first-order logic. Questions or queries can be expressed in the same language and then using logical inference, we can answer these questions. Logic programming approaches use deductive inference: given a knowledge base or a set of logic formulas, how we can decide whether a query formula holds? Inductive logic programming (ILP) approaches use an inductive way to generalize from instances. ILP approaches are more powerful compared to logic programming due to their ability to create knowledge bases automatically.

Propositional models treat the learning problem as points in a high-dimensional space and the goal is predicting a value — continuous or discrete — for each point. That is, there is a group of entities — all from the same type — and based on the attributes that exist for the entities, we want to predict the value of the goal attribute. Algorithms with statistical origin such as linear regression or logistic regression fall in to this group. One important class of these models is classification approaches such as decision trees, artificial neural networks, and support vector machines which are tailored for binary prediction.

There are advantages and disadvantages to both logic-based and propositional models. Propositional models can only handle flat data. For example if we want to predict if a person has a disease using propositional models, we only consider his attributes such as his age, gender, and smoking status. However, if the disease is a contagious one, then whether any person in his social network has the disease is very important information which is ignored by propositional model. While it is possible to incorporate the relational data in the propositional data — propositionalizing — it has some limitations. One is that the result might be a very huge dataset due to the exponential growth per each relationship that cannot be handled efficiently. On the other hand, logic-based models can handle relational data easily as they define some structure over entities — predicates. However, logic-based approaches are very poor in dealing with uncertainty and only can handle deterministic settings. In contrast, propositional models are statistical learning approaches and are robust to uncertainties.

Recently, there has been an emergence in models which can simultaneously handle both uncertainty and relational structure — known as statistical relational learning (SRL). Some SRL models are extensions to logic-based models which are known as rule-based models. Some others are extensions to statistical learning and object-oriented models which are known as model-based approaches. There is another class of SRL models — latent variable models — which assumes a latent variable for each entity and then the probability distribution over relationships between entities is modeled via a function over latent variables.

There are many SRL models proposed in the literature as we will explain in the next section. Each model is tailored for a special type problem. Rule-based models are more successful when logic is an appropriate representation language. Many frame-based models are based on relational database design and therefore are useful if one wants to construct a model over an already existing database. Latent variable models are more studied in the case where inference over only one relation is of interest, where they work better than the two other SRL models.

Here, I explain two examples of relational data for which latent variable models work well and will be presented in the next chapters. The first example is preference data. In preference data, there are two types of entities: users and items. Each user rates a number of items. The goal is usually to recommend items of interest based on the rating history of users. Traditional machine learning algorithms fail in this problem. For using the propositional models, we need a unique goal to predict.

However, for a user, we need to predict the rating for all unrated items. That is, instead of predicting an attribute, we want to predict a relationship. Also, it is hard to model such a problem with a logic-based model as there is a lot of noise in the data and the relationship is not binary.

Another examples is text data. In text data, entities are documents and words. Given the bag of words assumption — ignoring the sequence of words in documents — text data can be expressed as a matrix where each element represents the co-occurrence between a word and a document. One goal might be finding the category of each document such as a news group. The co-occurrence between words and documents can be treated as attributes. Such an approach ignores the relational structure of data as all words are from the same type. Also, the number of unique words is usually huge which makes using the propositional models burdensome.

Latent variable models can deal with these types of problems in a natural way. In preference data, we can assume a latent variable (which is a vector) for each user and each item. Then, the rating of a user for an item is a function of their latent variables. Similarly, given a latent variable for each word and document in a text data, the co-occurrence between a word and a document is a function of their respective latent variables. Note that in the latter example, predicting the co-occurrence value is not of interest per se but the latent variable of words and documents can be used for other purposes such as document categorization using their latent variables.

While latent variable models are very powerful in learning only one relation, extending them to multi-relational learning is not straightforward. For example, in the preference data example, some side information might be available regarding a user or an item. How can we use this information to perform better? One approach is using the same latent variable of one entity for learning all types of relationships. This hasn't been completely successful as different types of features explain different types of relationships. On the other hand, other SRL models — rule-based and model-based — can handle multi-relational data naturally but they need supervision to construct a sound model. Also, they are not as powerful as latent variable models. To date, no other SRL model has been as successful as matrix factorization — a latent variable model — in collaborative filtering problems.

The main goal of this dissertation is to explore learning and modeling relational data via latent variable models. First, I start with single-relational problems with specific characteristics and then I encounter multi-relational problems. In summary, the final destination is a generic model that can address following characteristics:

1. minimal human supervision
2. strong learning power
3. general purpose: covers all types of relationships
4. can be updated by adding new relationships.

To this end, we propose latent feature networks (LFN). LFN assumes a component for each relationship. Each component is a variant of latent variable models in which a latent variable is defined for each entity and the relationship is a function of latent variables. However, if an entity participates in more than one relationship, then it will have a separate random variable per relationship. Therefore, we call these latent variables in the component level local features. The local features for each entity are enforced to be correlated via some transformational functions such as linear functions. As a result, information flows between different relationships via enforcing correlation among local features.

LFN can address our goals as it automatically defines a component for an arbitrary relationship (minimal human supervision). LFN has a good learning power (via latent variable models). It can be general purpose if we can define a component for all types of relationships. Finally, we can add a new component to the system if we want to incorporate a new relationship.

1.1 Dissertation outline

In Chapter 2, background and related work relevant to the dissertation is presented. It starts with graphical models which are a key ingredient in statistical relational learning. Graphical models are used for very complex probabilistic models. Almost all modern statistical relational learning approaches use this framework. Similarly, we use graphical models for representing latent feature networks. Then, statistical relational learning literature is discussed. Chapter 2 is concluded with the main drawbacks in SRL literature. In Chapter 3, my work related to single-relational problems is discussed and presented. In Chapter 4, the latent feature network, a general approach for multi-relational learning via latent variable models, is presented. Finally, the conclusion and future directions are presented in Chapter 5.

1.2 Contributions of the dissertation

In this dissertation, I explored using latent variable models for learning relational problems. First, I modified latent variable models to improve the existing state-of-the-art single-relational models. In one direction, I proposed using Euclidean distance kernel in latent variable models which has several benefits such as visualization, fast information retrieval and fast embedding of new entities in an old model. In another direction, a kernel-based approach —latent logistic allocation — for co-occurrence data was developed. The state-of-the-art models for co-occurrence data use categorical latent space. This limits the learning power of algorithm. We relaxed this constraint and developed a Bayesian inference approach which was very successful in areas of information retrieval and dimensionality reduction. Furthermore, the scalability of collaborative filtering via co-clustering was improved via a memetic algorithm. Finally, a latent variable model — latent Dirichlet allocation — was customized for unsupervised learning in an educational problem.

Most previous studies — including mine — have applied latent variable models to single-relational data. However, learning multiple relations simultaneously can be very useful as one relation can improve learning another. Recently, some models have been proposed for handling multi-relational learning via latent variables [97, 96]. These models assume the same latent variable for each entity in different relationships. However, the essence of relations is different. Furthermore, available data for different relations might be imbalanced. Therefore, one random variable might address some relations well but some others poorly. To remedy this problem Agarwal et al. [1] proposed using a local latent variable for each relation of an entity and a single global latent variable for the entity. They assume that local latent variables are generated from a normal distribution with the mean of a linear transformation of the global random variable. Finally an ad-hoc algorithm based on maximum likelihood estimation is used for learning the latent variables. Although the theoretical model is sound, the impact of the model is not as expected and it mostly performs well in the cold start movie recommendation — when a user has only a few movies rated. Additionally, the mentioned literature is limited to real-valued relationships while many relationships are categorical. Also, they are mainly limited to the matrix factorization framework — using a dot product kernel in the latent variable models.

In the path towards multi-relational learning vial latent variable models, I proposed a general framework — latent feature networks. Latent feature networks assume a component for each relationship which is a single-relational latent variable model. Then it connects these components with the shared entities among relationships. All of the previous models studied in the literature can be derived from this framework. Therefore, latent feature network can be used as a flexible core model for constructing an all-purpose program for multi-relational learning via latent variable models. Furthermore, we derived special cases of latent feature networks for two specific problems: Microarray classification and social network analysis. Compared to models studied in the literature, in these approaches I modeled categorical relationships and developed an efficient Bayesian learning algorithm. In Microarray classification, our model outperforms support vector machines — the state-of-the art algorithm for the studied problem. In social network analysis, we showed that using bookmarking information can improve link prediction in a social network.

CHAPTER 2
BACKGROUND AND RELATED WORK

In this Section, we provide a survey on statistical relational learning approaches as the relevant literature. First, we review graphical models, a probabilistic model that often is used as a learning and inference representation for statistical relational learning.

2.1 Graphical models

The goal of graphical models [55, 89, 76, 68, 103] is presenting a high dimensional distribution over n random variables $x_1, x_2, ..., x_n$ in a compact way. When random variables are binary, then there are 2^n different configurations over the random variables. Graphical models use local dependency between random variables to represent a probability distribution in a structured way. A graph is used to represent the complex probability distribution. Nodes denote random variables and edges denote dependencies between random variables. The key concept to the compact representation is conditional independence. Random variables x and y are conditionally independent given random variable z if

$$P(x, y|z) = P(x|z)P(y|z).$$

Generally speaking, two random variables are conditionally independent given a third one, if the given random variable blocks information flow between them in the graph. That is, the given random variable should block any path between to random variable nodes in the graph. We will detail on conditional independence for different types of graphical models later.

Three important concepts related to graphical models — and generally to probabilistic modeling — are representation, inference, and learning. Representation is about the structure and properties of the probability distributions and mainly the way conditional independence can be derived. Inference concerns evaluating the values of random variables given the dependency and conditional independence structure. Learning is regarding the parameters associated to a probability distribution. In the next two sections we briefly describe representation, inference and learning for graphical models.

2.1.1 Representation

There are three different types of graphical models: Bayesian networks, Makrov networks and factor graphs. Bayesian networks (BNs) are directed acyclic graphs [47, 53, 89, 52, 23]. In BNs — similar to other graphical models — each random variable is represented by a node in the network. The dependency between random variables are defined via directed edges between nodes which can be interpreted as causal relationships. An example of a Bayesian network is shown in Figure 2.1-(a). There are 5 distinct binary random variables in this BN which represents the health condition of a typical person. A person might have Pneumonia, Tuberculosis, none or both. Both of these diseases might cause Lung Infiltrates. XRay can determine if Lung infiltrates exist. Seputum Smear is a test for detecting tuberculosis. This is a toy model but it represents a group of Bayesian network models for disease diagnosis.

A probability distribution represented by a Bayesian network is factorized as follows:

$$P(x_1, ..., x_n) = \prod_i P(x_i|\mathcal{P}_i) \tag{2.1}$$

where $\mathcal{P}_i$ represents the set of parents of random variable x_i. In the example, parents of the “Lung Infiltrates” random variable are “Pneumonia” and “Tuberculosis”. In many applications of Bayesian networks, the conditional probability distribution $P(x_i|\mathcal{P}_i)$ is given by a table-based probability that determines the probability of the child given each possible configuration of the parents. Such a representation is exponential in the number of parents and only applicable to discrete models. Figure 2.1-(b) represents a set of conditional probability distributions in table format for the example.

The conditional independence in Bayesian networks is determined by the concept of d-separation [76]. Two random variables — or generally groups of random variables — are d-separated by a random variable z — or a group of random variables — if there is no active path between them given z. A path between two nodes is active if it is a directed path and no connecting random variable is observed along the path or there is a directed path from both random variables to the observed random variable z. The first type of path — directed path — is active since information can flow from parent to child and vice versa. The second type of path is active since if the child of two parents is observed then it gives information about all parents which makes them dependent.

The second type of graphical models is Markov networks [89, 93, 66, 94] —

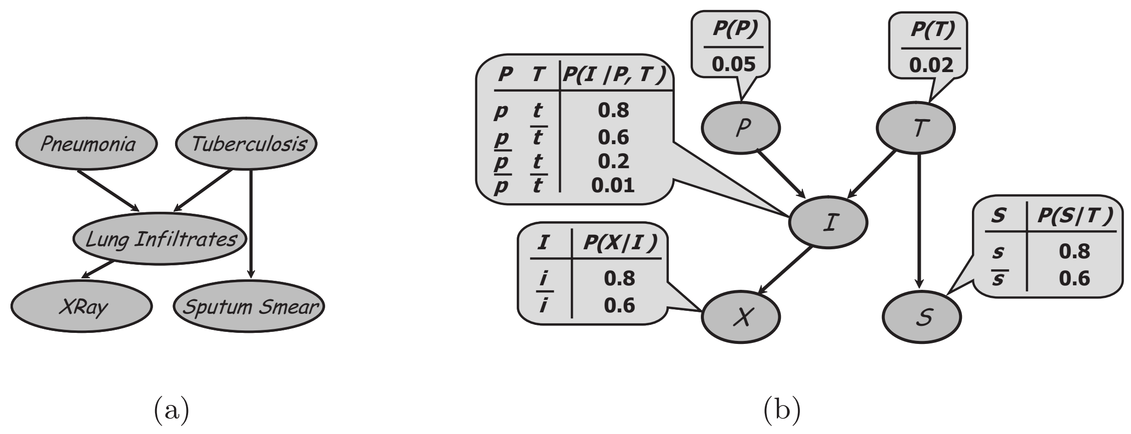
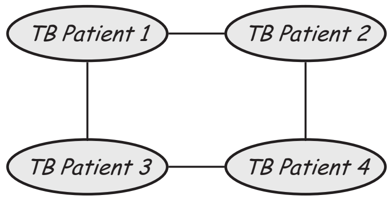


Figure 2.1: Bayesian network, (a) Representation, (b) Conditional probability tables (Picture from [40], chapter 2)

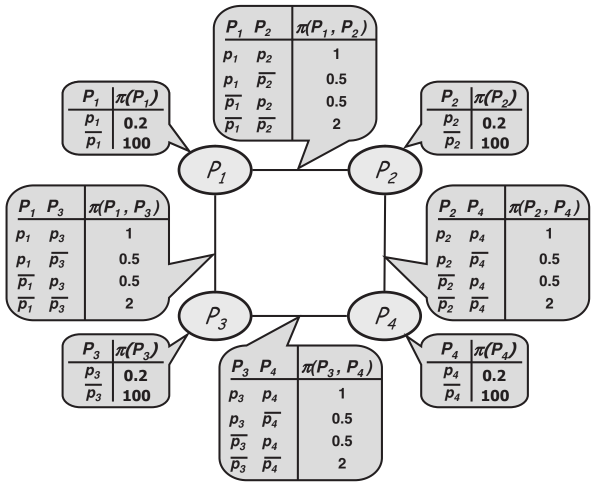
Markov random fields (MRF) — which are undirected graphical models. Markov networks are very flexible in representing probability distributions given a Gibbs distribution. A probability distribution follows a Gibbs distribution if it has the following format:

$$P(x_1, ..., x_n) = \frac{1}{Z} \prod_k \pi_k(C_k) \tag{2.2}$$

where C_k is a set of random variables, π_k is a function over random variables in C_k, and $Z = \sum_X \prod_k \pi_k(C_k)$ is the partition function which enforces the probability distribution adding to one. The function π_k is called the potential function or factor and as the only condition, it has to be positive. A Markov network is defined based on a Gibbs distribution over variables where there is an edge between two nodes if they share a variable set C_k. This definition results in cliques for each C_k in the Markov network.



(a)



P_1	$\pi(P_1)$
p_1	0.2
$\overline{p_1}$	100

P_1	P_2	$\pi(P_1, P_2)$
p_1	p_2	1
p_1	$\overline{p_2}$	0.5
$\overline{p_1}$	p_2	0.5
$\overline{p_1}$	$\overline{p_2}$	2

P_2	$\pi(P_2)$
p_2	0.2
$\overline{p_2}$	100

P_1	P_3	$\pi(P_1, P_3)$
p_1	p_3	1
p_1	$\overline{p_3}$	0.5
$\overline{p_1}$	p_3	0.5
$\overline{p_1}$	$\overline{p_3}$	2

P_2	P_4	$\pi(P_2, P_4)$
p_2	p_4	1
p_2	$\overline{p_4}$	0.5
$\overline{p_2}$	p_4	0.5
$\overline{p_2}$	$\overline{p_4}$	2

P_3	$\pi(P_3)$
p_3	0.2
$\overline{p_3}$	100

P_3	P_4	$\pi(P_3, P_4)$
p_3	p_4	1
p_3	$\overline{p_4}$	0.5
$\overline{p_3}$	p_4	0.5
$\overline{p_3}$	$\overline{p_4}$	2

P_4	$\pi(P_4)$
p_4	0.2
$\overline{p_4}$	100

(b)

Figure 2.2: Markov network, (a) Representation , (b) Node potentials (Picture from [40], chapter 2)

Figure 2.2-(a) shows an example of a Markov network. This Markov network includes four random variables for different people. Each random variable is binary and is true if the person has Tuberculosis. There is an edge between two patients if they have been in contact. Figure 2.2-(b) shows a set of potential functions for each contact.

Factor graphs [78, 73, 37] are very similar to Markov networks as they are used to represent a Gibbs distribution (2.2) over random variables . The only difference is in the graphical representation. Factor graphs are bipartite graphs with two different types of nodes. The first type of nodes are random variables and the second type of nodes represent factors which are a function of random variables. Figure 2.3 represents the factor graph representation of the Markov network example in Figure 2.2.

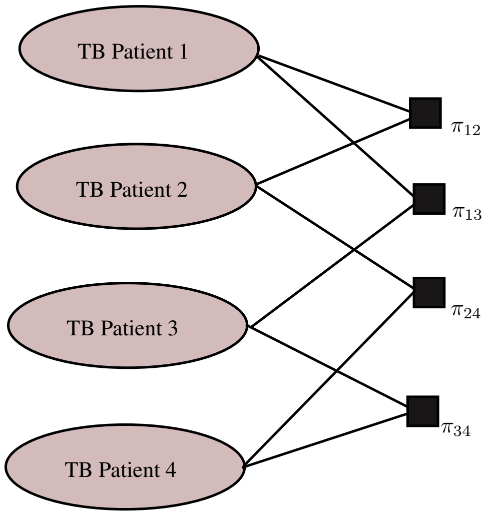


Figure 2.3: A factor graph represents the Markov network in Figure 2.2-(a). Potential functions π are defined similar to Figure 2.2.

Conditional independence is easier in Markov networks and factor graphs compared to Bayesian networks. Two random variables are dependent if there is any path between them in the undirected graph. Therefore, two variables are conditionally independent given a group of observed variables if a part of all paths between them is observed.

There are some dependencies that only can be represented by a directed or undirected graph. However, this only holds at the graphical representation level. Note that the factorized probability distribution for Bayesian networks (2.1) is a special case of Gibbs distribution (2.2). As a result, it is straightforward to convert a Bayesian network to a Markov network while the other direction is not true [68]. This is because of the local constraints posed on conditional probability distributions in the

Bayesian network model. Frey et al. [36] proposed an extended factor graph which includes directed edges as well and has the representational power of both directed and undirected models. However, such an extension does not affect computations for inference.

2.1.2 Inference and Learning

Inference is one of the main goals in probabilistic models where we are interested in computing probability $P(Y|E)$ where Y is a group of unobserved random variables of interest and E — evidence — is the set of observed variables. From probability theory, we have

$$P(Y|E) = \frac{P(Y,E)}{P(E)}.$$

Therefore, inference consists of simply computing marginal distributions. If we define a set of random variables X as the rest of the random variables (other than E and Y), then we need to compute $P(Y,E) = \sum_X P(Y,E,X)$ and $P(E) = \sum_Y P(Y,E)$. These simple looking sums turn out to be intractable in high-dimensional probability distributions. If we ignore the structure posed by the graphical model and want to sum over unwanted random variables one by one, then the number of operations is exponentially increasing with the number of random variables. For continuous random variables, inference is even harder as the analytical integration may not be possible even for one variable.

Learning in graphical models [54] includes inference over random variables, inference over distribution parameters, or constructing a dependency structure that fits the data. There are algorithms for performing exact inference which are limited to special families of distributions. Sum-product algorithms [73] are inference algorithms that give exact results only for factor graphs without loops or trees. As we discussed in the previous section, other graphical models can be converted to factor graphs. This algorithm starts by summing over random variables in leaves and passing the result as a message to other random variables. After all random variables get the message from their connected factors, we have the marginal for all of them. Replacing sum by integration gives the algorithm for continuous variables. Belief prorogation [89] for discrete random variables or Kalman filtering [3] for continuous random variables are special cases of the sum-product algorithm.

If we are only interested in the modes of random variables, the max-sum algorithm using dynamic programming ([8], ch 8) can be used.

If there is a loop in the graph, then the sum-product algorithm does not work. We still can run it, but there is no guarantee that the algorithm converges and even if it converges, the results are not exact. Such a strategy has worked in some applications and is known as loopy belief propagation [89]. Some convergence proofs can be found in [107].

There are two main groups of approximate inference for graphical models which have origins in statistical physics. The first are sampling methods which are known as Markov chain Monte Carlo approaches (MCMC) [83, 41, 4]. MCMC is an iterative Monte Carlo simulation to sample from any probability distribution. First, we initialize random variables. Then in each step, a new configuration is given from a proposal distribution which might depend on the previous configuration. Then using a formula, the proposed sample is accepted or rejected. Such a process is a Markov chain since we move in the space from point to point — a chain — and the place we go only depends on where we are now — Markovian. It can be shown that if an MCMC algorithm converges, then we are sampling from the true probability distribution which means our estimation based on sampling is exact in limit. However, this might not be the case as the needed iterations before converging might be computationally far away.

MCMC can be seen in the framework of stochastic optimization heuristics. Greedy search might get stuck in a local mode — usually complex probability distributions are multi-modal (non-concave). A highly randomized setup might wander aimlessly forever without sampling from the important regions of the distribution. To resolve this trade-off problem, some optimization heuristic algorithms such as simulated annealing/tempering or population based algorithms are used ([77], chapters 4 and 5).

Variational approximations [86, 106, 56, 28] use algorithms similar to methods in variational calculus. In variational approximation, the main inference problem $P(Y|E)$ is replaced by an easier one $Q(Y|E)$ — the variational distribution. Then the divergence between Q and P — a distance function such as KullbackLeibler divergence — is minimized. Q is family of distributions which is similar to the concept of a functional in variational calculus. Since $P(Y|E)$ is not tractable and we cannot compute the distance function directly, the lower bound over the divergence which involves the available $P(Y, E, X)$ is minimized.

The most popular variational approach is the naive mean field method [103] in which the variational distribution Q factorizes over all random variables. That is, we recover independent distributions over random variables while they are correlated. This is very similar to maximum likelihood estimation (MLE) or maximum a posteriori probability (MAP) with the main difference that mean field methods provide distributions over random variables while MLE or MAP only give point estimates. While this might seem a subtle difference, the result can be much better mainly since we use the point estimates or distribution estimates iteratively to re-estimate other random variables and propagating uncertainty via distributions causes robustness to noises. Variational approximations have provided results very close to exact inference — posterior distribution — in many applications.

In average, MCMC algorithms are slower compared to variational methods while they are easier to implement. Also, there are convergence and exactness proofs for MCMC while there is no guarantee that variational approximation gives a good result. Finally, a combination of both is possible where the proposal distribution in MCMC is guided by variational approximation which gives a strong combination [35, 105, 26, 18, 20, 19, 29]

2.2 Statistical relational learning

Traditional machine learning algorithms either address uncertainty — statistical learning approaches such as support vector machines — or relational data

— logic-based approaches such as inductive logic programming. Statistical relation learning (SRL) [12, 40, 64] approaches try to address both uncertainty and relational data.

Early efforts in the machine learning community were based on propositionalizing — transferring relational data into a flat format [71, 67, 70]. Such an approach is ad-hoc and inefficient since transferring relational data to flat data might explode exponentially and some complex relations may be lost. Modern SRL approaches can be divided into 3 categories: rule-based models, frame-based models, and Latent variable models. Rule-based models use logic language — mostly first-order logic — for representing knowledge and via probabilistic models try to infer or validate logic formulas. Frame-based models are object-oriented models that use graphical models for representing objects or entities and the relationships between them. Finally, latent variable models assume random variables for each entity for learning the relations.

2.2.1 Rule-based models

Rule-based models use the language of logic to represent knowledge. An early deterministic rule-based model is logic programming [50] which is a deductive method. The syntax or language in logic programming is first-order logic and inference is based on deducing some formulas given other formulas. Inductive logic programming (ILP) [81] generalizes from instances in data to construct knowledge base models (a set of logic formulas). While the early logic-based models are able to handle relational data via logic formulas, they fail to perform in the presence of uncertainty due to the rigidness of pure logic. Rule-based models are extended to SRL models by combining logic-based methods with probabilistic representations such as Bayesian networks. Bayesian logic programs ([40], chapter 10) combine logic programming and Bayesian networks. Stochastic logic programs ([40], chapter 11) use a combination of logic programs and stochastic grammar (a special case of Markov networks).

One of the successful rule-based relational learning approaches is Markov logic networks (MLN) [90], which combine first-order logic — for representing knowledge — and Markov networks — for probabilistic reasoning. MLN assigns a weight to each first-order logic formula in a knowledge base which implies the likelihood of the formula. Violating a formula in a knowledge base, a world will be less probable — not impossible as is the case in logic programming. For example, consider the following formulas in a knowledge base ([40], chapter 12):

$$\forall x, Smoke(x) \Rightarrow Cancer(x), \tag{2.3}$$

$$\forall x, y, Friend(x, y) \Rightarrow (Smoke(x) \Leftrightarrow Smoke(y)). \tag{2.4}$$

Formula 2.3 means if person x smokes then person x has cancer and formula 2.4 implies friends have similar smoking habits. Using only logic representation, if only one person smokes but does not have cancer, then (2.3) does not hold but in MLN its likelihood decreases.

MLN constructs a Markov network using the formulas in the knowledge base by defining a node for each predicate — such as Cancer or Smoke — in the knowledge base and connecting nodes that share the same formulas. However, variables in the

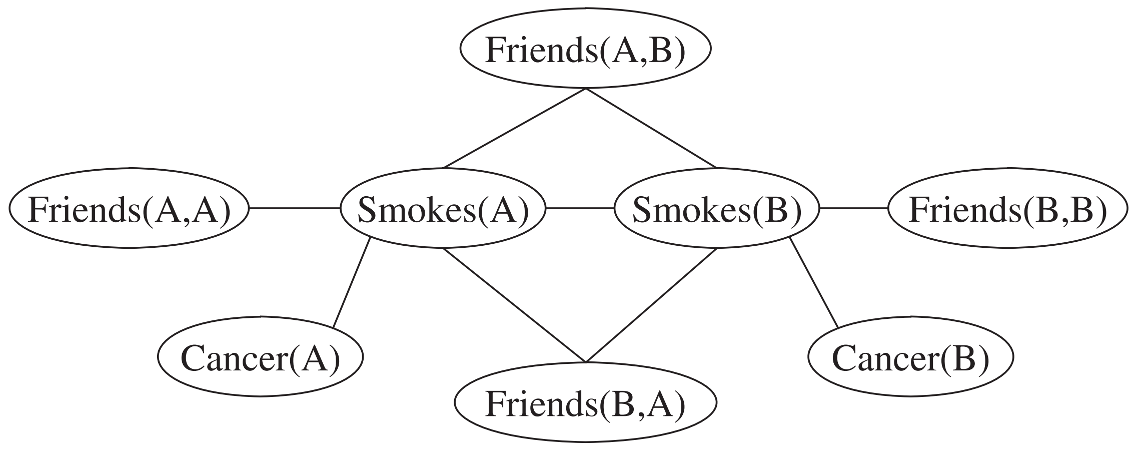


Figure 2.4: An example of Markov logic network (Picture from [40], chapter 12)

predicates are replaced by instances. Figure 2.4 represents a Markov network for two persons A and B using formulas 2.3 and 2.4.

The probability distribution of an assignment to variables is computed from

$$P(x) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right) \tag{2.5}$$

where $n_i(x)$ is the number of times formula i holds given the assignment x and w_i is the weight of formula i.

2.2.2 Frame-based models

Frame-based models are object-oriented models that use graphical models for representing objects or entities and the relationships between them. Many discriminative frame-based algorithms are extensions to propositional machine learning algorithms such as support vector machines. Some others are extensions to object-oriented relational models such as the entity-relationship model.

Conditional random fields (CRF) [75] are undirected graphical models. CRF is a discriminative model which means that the model concentrates on the discrimination task in classification. Let X denote the observed variables and Y denote the hidden random variables that we wish to predict. Using Gibbs distribution 2.2 for Markov random fields, the probability distribution over all variables is

$$P(X,Y) = \frac{1}{Z}\prod_k \pi_k(C_k), \tag{2.6}$$

where each C_k is a subset of X and Y variables and the partition function is $Z = \sum_{X,Y}\prod_k \pi_k(C_k)$. In contrast, CRF is only concerned with predicting Y and uses

$$P(Y|X) = \frac{1}{Z}\prod_k \pi_k(C_k), \tag{2.7}$$

where $Z = \sum_Y \prod_k \pi_k(C_k)$. This makes computations much easier due to reducing inference only over the variables Y.

In practice, the potential function is represented as a log-linear model over features:

$$\pi_k(C_k) = \exp\left(\sum_j \theta_{jk} f_{jk}(C_k)\right) \tag{2.8}$$

where θ_{jk} are real-valued parameters and f_{jk} is a feature function defined over the data. Given this definition, we have an exponential family distribution with natural parameters θ and sufficient statistics f. θ is often learned via maximum likelihood and f is defined by the modeler. Note that if we have only one variable in set Y, then CRF reduces to logistic regression.

Relational Markov networks ([40], chapter 6) are similar to CRFs and are used for collective classification. The specific character of relational Markov networks is

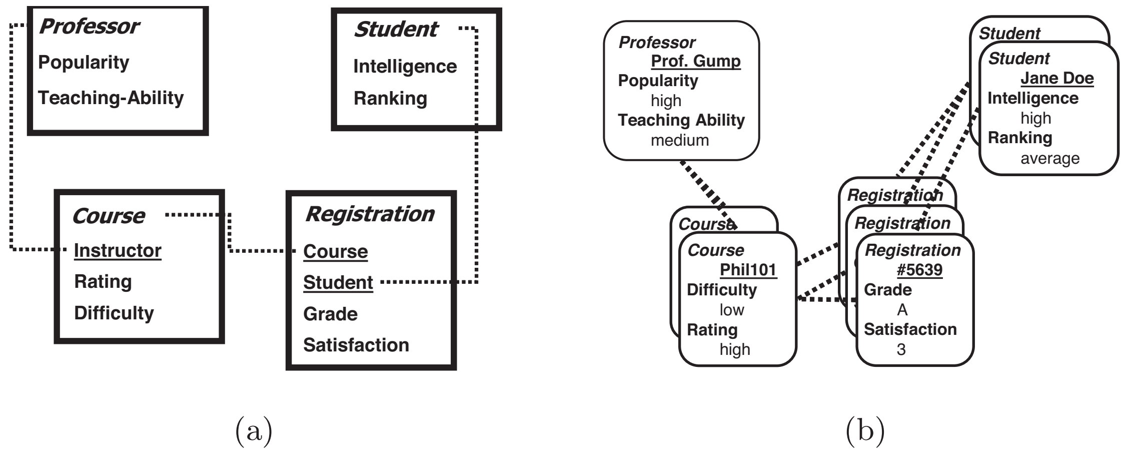


Figure 2.5: An example of a relational model, (a) Representation, (b) Some instances of the relational model (Picture from [40], chapter 5)

that they use SQL queries for defining potential functions. Both CRF and RMN are discriminative models.

Probabilistic relational models (PRM) [39] define a probability model based on a relational database. A relational database consists of tables for different entities. Each table includes a number of attributes associated with the entity. Tables are related via the intrinsic relationships between entities. Figure 2.5-(a) represents a relational database for a university course system. There are four types of entities: professor, student, course, and registration. Each entity has some attributes as shown in the picture. Each course is related to a professor as its instructor. Each registration includes a course and its related student. Figure 2.5-(b) represents some instances in this database.

PRM constructs a Bayesian network over a relational database. This is defined

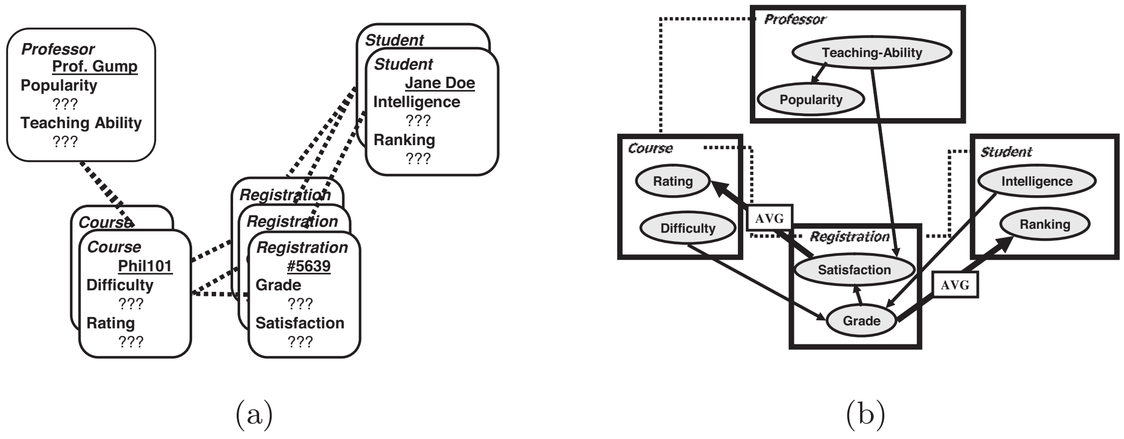
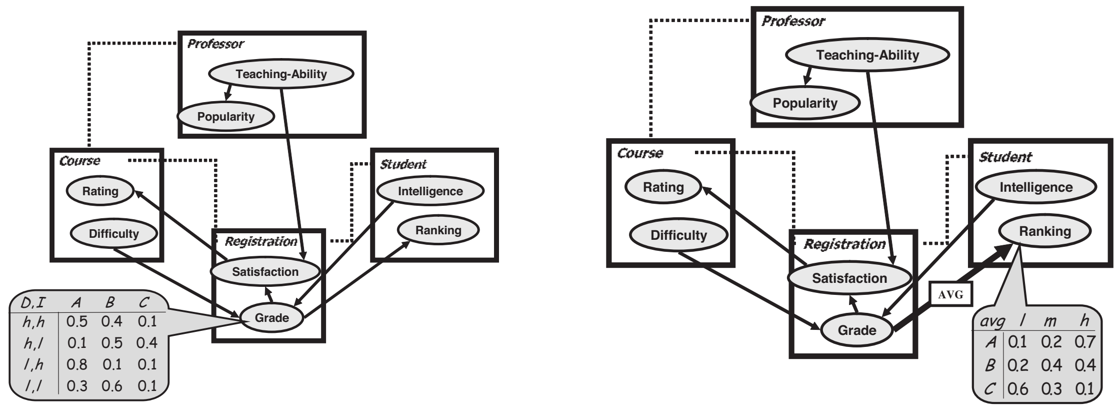


Figure 2.6: Probabilistic relational model structure, (a) Representation, (b) Structure (Picture from [40], chapter 5)

at the attribute level. That is, each attribute affects some other attributes. Connected attributes may be from the same table or different tables. Figure 2.6-(b) represents a Bayesian network constructed over the example in Figure 2.5. Teaching-ability of a professor affects his popularity and the satisfaction from a course — the first one is internal while the second is external. Satisfaction influences the rating of the professor for a course. Other relations can be interpreted similarly.

The parameterization of PRM is based on conditional probability distributions (CPD) as shown in Figure 2.7. These CPDs are learned via maximum likelihood estimation.

Relational dependency networks (RDN) [84] are bi-directional graphical models to represent relational data. In RDB, two entities can be parents of each other — the bi-directional concept. However, the independence view of the graph is similar



(a) (b)

Figure 2.7: Some examples of conditional probability distributions (Picture from [40], chapter 5)

to undirected graphs ignoring the direction. Directions are only used for defining conditional probability distributions. Therefore, the model does not represent a valid joint distribution but using MCMC methods, it can give a sound joint distribution by normalizing over samples. In this sense, it gives a Gibbs distribution having inconsistent conditional distributions as potential function and Neville et al. [84] showed that RDN has an equivalent relational Markov network. Factorie [79] is another frame-based SRL model that uses factor graphs as representation and is a discriminative model similar to conditional random fields.

2.2.3 Latent variable models

Latent variable models capture a relationship by assuming a latent variable for each entity. Contrary to the previously-mentioned SRL models that modeled relations via a log-linear function of some features, in latent variable models we do not have to define features explicitly. Using latent variables we can model different kinds of relationships such as real-valued or categorical easily.

In what follows, we present two latent variable models in the context of two applications. The first application is collaborative filtering which is a real-valued relation prediction problem. The second is text data modeling which is a categorical relation learning problem.

2.2.3.1 Collaborative filtering

Collaborative filtering is used to suggest items of interest to users. Assume that we have a number of users and items, and some users have rated some items (e.g., based on a 1-to-5 scale). The main task is recommending appropriate items to users based on their previous ratings. One natural approach, which is the goal of many collaborative filtering methods, is the prediction of unknown ratings. The user can then be given suggestions based on items with a high expected rating.

One of the most accurate and scalable collaborative filtering algorithms is matrix factorization (MF), which is based on a latent factor model [69]. Singular value decomposition (SVD) and related methods using gradient descent are often used in collaborative filtering [15].

In a collaborative filtering (CF) problem, there are N users and M items. Users have provided a number of explicit ratings for items; r_{ui} is the rating of user u for item i. The goal of collaborative filtering approaches is predicting unknown ratings given known ratings. There are two popular error functions; mean absolute error (MAE) and root mean squared error (RMSE). Since the absolute value function is not differentiable at all points, RMSE is more desirable as an objective function. Therefore, the objective function of a model-based collaborative filtering approach can be defined as follows:

$$\min \sum_{u} \sum_{i} w_{ui}(r_{ui} - \hat{r}_{ui})^2 \tag{2.9}$$

where $\hat{r}_{ui}$ is the prediction of the model for the rating of user u for item i. w_{ui} is 1 if the rating r_{ui} is known and 0 otherwise.

Latent variable models such as SVD transfer users and items to a low-dimensional space to uncover the latent pattern of ratings. To predict a rating via MF, the following formula can be used [69]:

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u q_i' \tag{2.10}$$

where μ is the total average of all ratings, b_u is the deviation of user u from average and b_i is the deviation of item i from average. b_u and b_i model the fact that some users tend to rate higher and some items are more likable. p_u and q_i are the user-factor vector and item-factor vector respectively in a D-dimensional space. $p_u q_i'$ is the dot product between p_u and q_i where a higher value means user u likes item i more than average.

Since in collaborative filtering problems the data matrix is highly sparse, classical SVD approaches have not been successful [69]. Therefore, a gradient descent approach is suggested to solve this problem by minimizing the following objective function [100, 88]:

$$\min_{p,q,b} \sum_{u,i} w_{ui}[(r_{ui} - \mu - b_u - b_i - p_u q_i')^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2)] \tag{2.11}$$

where the $\lambda(\|p_u\|^2+\|q_i\|^2+b_u^2+b_i^2)$ term avoids overfitting by restricting the magnitude of parameters. λ is an algorithmic parameter. Using formula (2.11), the gradient descent updates for each known rating r_{ui} can be as follows [69]:

$$\begin{array}{lcl} b_u & \leftarrow & b_u + \gamma(e_{ui} - \lambda b_u) \\ b_i & \leftarrow & b_i + \gamma(e_{ui} - \lambda b_i) \\ p_u & \leftarrow & p_u + \gamma(e_{ui} q_i - \lambda p_u) \\ q_i & \leftarrow & q_i + \gamma(e_{ui} p_u - \lambda q_i) \end{array}$$

where e_{ui} is the current error for rating r_{ui} and γ is the step size of the algorithm. Therefore, in each iteration of the gradient descent algorithm, there are T (number of known ratings) steps to go through all ratings in the training dataset.

2.2.3.2 Learning co-occurrence data

Co-occurrence data can be represented as a matrix where each element of the matrix — a dyad — depicts the number of times the entity represented by the row and the one represented by the column have co-occurred. It is possible that rows and columns represent the same or different types of objects. We explain the case of the co-occurrence between objects from different types, and then generalizing to the case of same types is straightforward. Most of the related work on applying latent variable models to co-occurrence data (e.g. latent Dirichlet allocation — LDA) is applicable to the different types case.

In co-occurrence data, each dyad can be divided into a number of separable entities — tokens. Each token is associated with two entities from two different types (an entity i and an entity k). Therefore, for each token, the distribution over entities is categorical — multinomial. Representing each token with index u, a token is represented as (i_u, k_u) where $u = 1, ..., N$ (N is the total number of tokens), $i_u = 1, ..., N_I$ (N_I is the total number of entities of the first type), and $k_u = 1, ..., N_K$ (N_K is the total number of entities of the second type). v_{ik} represents the number of tokens with value (i, k). Text data is an example of co-occurrence data, where a token (i_u, k_u) represents the word i_u that occurred in document k_u. That is, for a token u in a document k, there is a multinomial distribution over words indexed by i. In the case of similar entities, each dyad addresses the co-occurrence between the entities from the same type.

A common approach in learning from co-occurrence data is latent variable models. In these models, a latent space is assumed to be responsible for generating the data and the learning task is recovering the latent variables.

The first proposed latent variable model for the co-occurrence data was latent semantic indexing/analysis (LSI) [30]. LSI uses singular value decomposition (SVD) to learn the latent variable space. Although SVD was the first latent variable approach to model the co-occurrence data, it can be applied to any dyadic data with any format. SVD has been used to model preference data [7] and a simpler version of SVD (matrix factorization) has also been used for the same problem [69].

Let X_i (a $1 \times D$ vector) represent the latent variable of entity i and Y_k (a $1 \times D$ vector) represent the latent variable of entity k in a D-dimensional space. In LSI, the value of a dyad is estimated from the weighted dot product function:

$v_{ik} \simeq X_i W Y_k^T$ where W is a $D \times D$ matrix of weights. For learning the latent space, LSI minimizes the squared error function: $\sum_{ik}(v_{ik} - X_i W Y_k^T)^2$. The main intuition behind LSI is that the values v_{ik} are noisy and using SVD, we can retrieve more robust representatives for entities via a low rank matrix.

Probabilistic latent semantic indexing (pLSI) [49] moves towards a probabilistic extension of LSI by constraining the latent variables and employing the maximum likelihood approach. In pLSI, instead of constructing a model based on a dyad value v_{ik}, the joint probability of a token (i, k) occurring is learned: $P(i, k)$. Using the same notation for latent variables, we have:

$$P(i,k) = \sum_{d} P(z)P(i|z)P(k|z) = X_i W Y_k^T$$

where the z^{th} dimension of X_i is interpreted as $P(i|z)$, the z^{th} dimension of Y_k is interpreted as $P(z|d)$, and the diagonal elements of W are interpreted as $P(z)$ and non-diagonal elements are zero. Given that we have N tokens, the probability of the whole dataset follows a multinomial distribution:

$$\mathcal{D} \sim Multinomial(\{P(i,k)\}_{i,k}, N).$$

Then the likelihood function is:

$$L = \sum_{ik} v_{ik} \log P(i,k)$$

which is to be maximized subject to the constraints:

$$\sum_{i} X_{iz} = \sum_{i} P(i|z) = 1$$

$$\sum_k Y_{kz} = \sum_k P(k|z) = 1$$
$$\sum_z W_{zz} = \sum_z P(z) = 1.$$

In pLSI, the expectation maximization algorithm is used to maximize the likelihood function. pLSI is very prone to overfitting even though a tempered heuristic is used in the learning algorithm. Another problem in pLSI is learning the parameters for new documents. As we learn the probability of document k given dimension z, learning the parameters for a new document changes the probability distribution for the training documents. To address these problems, Blei et al. [11] proposed a generative probabilistic algorithm — latent Dirichlet allocation (LDA). In LDA, two types of entities are considered differently. Compared to pLSI which models a token by $P(i, k)$, LDA models it by $P(i|k)$. More accurately, in LDA, we assume that each token belongs to an entity k of the second type and the generative model chooses an entity i from N_I entities of the first type. In the text data example, for each document k, there are N_k tokens and for each token, a word i is chosen.

Using the same notation for latent variables, we have:

$$P(i|k) = \sum_z P(i|z)P(z|k) = X_i Y_k^T \tag{2.12}$$

where the z^{th} dimension of X_i is interpreted as $P(i|z)$ while the z^{th} dimension of Y_k is interpreted as $P(z|k)$. Note that the weight matrix W is the identity matrix in this model. LDA, also known as a topic model, is interpreted based on D topics. For each token in entity k, a topic z is chosen from a multinomial distribution with probabilities $P(z|k)$ and then given the topic z, an entity i is chosen from a multinomial distribution with $P(i|z)$. The concept topic can be interpreted as an auxiliary variable. However, the final probability for a token comes from (2.12), since the topic of the token is an unobserved hidden variable.

The main generative model is usually explained via the language model which generates the text content in a collection of documents. In this model, there is a predefined number D of topics. The language model of each document is defined by the distribution of the document over the D topics: $Y_{kz} = P(z|k)$ where z is the topic and k is the document. Y is generated from a symmetric Dirichlet distribution with parameter α ($P(Y|\alpha) = \frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K} \prod_k \theta_k^{\alpha-1}$). Document k consists of N_k tokens (an instance of a word), and each token comes from a specific topic z. Given that a topic z has generated a token, the probability that word i occurs for that token is $X_{iz} = P(i|z)$. X is generated from a symmetric Dirichlet distribution with parameter β. In summary, the process of generating documents of a corpus is as follows:

1. For each topic z:
    (a) Choose word distribution $X_{.z} \sim Dir(\beta)$
2. For each document k:
    (a) Choose topic distribution $Y_{k.} \sim Dir(\alpha)$
    (b) For each token u
        i. Choose a topic $z \sim Multinomial(Y_{k.})$
        ii. Choose a word $i \sim Multinomial(X_{.z})$

Figure 2.8 shows the graphical model for LDA using the plate notation. The

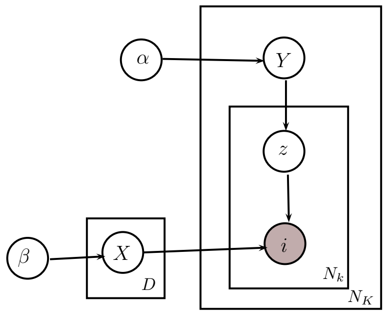


Figure 2.8: The graphical model of the latent Dirichlet allocation model

number in the lower-right corner of each plate depicts the frequency of the contents of that plate. In the corpus, there are N_K documents and for each document a distribution over topics (Y) is generated. There are D topics and for each topic a distribution over words (X) is generated. Finally, there are N_k tokens in document k and for each token a topic d and a word i is chosen.

Due to the intractability of inference in LDA, Blei et al. [11] used variational inference for doing approximate inference. Gibbs sampling, a Markov chain Monte Carlo algorithm, is another strategy to infer the parameters in LDA [44].

Both pLSI and LDA extend LSI towards a probabilistic model on the same direction by defining latent variables X and Y as probability parameters. Therefore, the (weighted) dot product can be interpreted as the probability model on dyads. However, constraining latent variables to create a probabilistic model is not the only way to extend LSI. It is possible to allow latent variables to be free variables and translate the latent space to a probability model via other mathematical manipulations such as logistic function, as will be discussed in the next section. This way, the model can fit the data better due to the flexibility of latent variables.

Another drawback in pLSI and LDA is ignoring the biases in occurrence of different entities. In the text data example, some words tend to occur more often. Both models use the parameter $P(i|d)$ (the probability of word i given topic d) for explaining word distributions. Therefore, the words that occur more often dominate the space. To address this problem, LSI and pLSI use normalized frequencies such as tf-idf. However, due to the learning mechanism of LDA, such an approach is not feasible and LDA alone is not very capable in retrieval [104]. As a result, it will be helpful to model the biases of entities explicitly. Such an approach has been used successfully in the collaborative filtering context [69] to model the biases of items and users. In this model, a dot product of latent variables of a user and an item represents the bias of the user for that specific item and each user and each item have their own specific biases. Since in pLSI and LDA, the dot product itself is interpreted as a probability, it is hard to incorporate the bias in these models. The inclusion of biasness is possible in LLA, as will be explained.

Another relevant work is sufficient dimensionality reduction (SDR) [43], which is a model for performing dimensionality reduction in co-occurrence data. SDR is an information theoretic approach to model co-occurrence data rather than a probabilistic model. Let $\tilde{P}(i,k)$ be the empirical probability distribution over tokens. It is assumed for simplicity that marginal probabilities of the probability model and the empirical probability are equal: $P(i) = \tilde{P}(i)$ and $P(k) = \tilde{P}(k)$. Using a maximum entropy approach, latent variables X provide maximal information about entity k

when they satisfy the following constraint:

$$\sum_i X_i P(i|k) = \sum_i X_i \tilde{P}(i|k). \tag{2.13}$$

More precisely, the expectation of latent variables for the entities of the first type with regard to the probability model and the empirical probability should be the same. A similar constraint is defined for latent variables Y:

$$\sum_k Y_k P(k|i) = \sum_k Y_k \tilde{P}(k|i). \tag{2.14}$$

Given the equality of marginal probabilities and constraints (2.13) and (2.14), there is a unique maximum entropy distribution for the probability model:

$$P(i,k) = \frac{1}{Z} \exp(X_i Y_k^T + b_i + b_k), \tag{2.15}$$

where $Z = \sum_{ik} \exp(X_i Y_k^T + b_i + b_k)$, and b_i and b_k can be interpreted as biases for entities i and k. Finally, the latent variables can be found via minimizing the Kullback-Leibler divergence between empirical and model probabilities:

$$P^* = \arg \min_{P \in P_\Theta} D_{KL}[\tilde{P}|P]$$

where P_Θ is the set of probabilities that satisfy the marginal and expectation equality constraints. Note that in SDR, bias parameters are not learned and are estimated directly from the marginal empirical distributions.

Co-occurrence data embedding (CODE) [42] is a similar approach to SDR with the main difference of using squared Euclidean distance instead of dot product to capture the relationship between latent variables. CODE is mainly designed to visualize co-occurrence data. In CODE, two approaches are used for fitting the model to the data. In the first approach, the joint distribution is modeled via:

$$P(i,k) = \frac{1}{Z}\tilde{P}(i)\tilde{P}(k)\exp(-\delta_{ik}^2), \tag{2.16}$$

where $\delta_{ik}^2 = (X_i - Y_k)(X_i - Y_k)^T$ is the squared Euclidean distance between latent variables of i and k and the normalizing factor is $Z = \sum_{ik}\tilde{P}(i)\tilde{P}(k)\exp(-\delta_{ik}^2)$. Note the similarity between CODE and SDR; replacing the dot product with the negative squared Euclidean distance, $b_i = \log\tilde{P}(i)$ and $b_k = \log\tilde{P}(k)$, (2.15) is the same as (2.16).

The second approach in CODE is modeling the conditional probability instead of the joint probability:

$$P(i|k) = \frac{1}{Z_k}\tilde{P}(i)\exp(-\delta_{ik}^2), \tag{2.17}$$

where $Z = \sum_i \tilde{P}(i)\exp(-\delta_{ik}^2)$, which is useful when the empirical conditional probability is available.

To learn the model parameters of CODE, minimization of the Kullback-Leibler divergence between empirical and model probabilities is used. The main intuition behind CODE is that if two points are related then they should be very close in the Euclidean space. Therefore, the result of embedding can be presented as a visualization of entities. Although the embedding is based on the relationship between entities from different types, we expect the entities from the same type to be close due to the transitivity of distance.

Latent logistic allocation (LLA) [61], discussed in Section 3.3, is very similar to CODE and SDR especially for the basic probability model. The main difference is extending CODE and SDR to a fully generative probabilistic model for learning co-occurrence data. Instead of the maximum likelihood approach which is very vulnerable to overfitting, LLA uses a Bayesian approach which has some classic advantages such as robustness. More specifically, SDR is a mere dimensionality reduction technique and CODE is a mere visualization approach while LLA is a Bayesian generative probabilistic model and as a result can be compared to other probabilistic models such as LDA. In this sense, the relationship between SDR/CODE and LLA is very similar to the relationship between LSI/pLSI and LDA. The LLA algorithm uses a Bayesian approach to learn entity occurrence bias, whereas bias parameters were estimated directly from marginal empirical probabilities in SDR and CODE. Furthermore, CODE and SDR are used for co-occurrence data in its all forms; however, LLA provides four different categories for co-occurrence data and present a specific model for each.

Finally, it is worth mentioning that although some approaches such a unigrams or mixture of unigrams [85] are latent variable models to learn co-occurrence data, they do not embed both types of entities in the space and therefore, are fundamentally different from models studied here.

CHAPTER 3
SINGLE-RELATIONAL LEARNING

In this chapter, the work related to single-relational problems is presented. All the completed work is considered as statistical relational learning via latent variable models. As is explained in Chapter 4, we present a multi-relational latent variable model — latent feature networks (LFN) — in this dissertation. LFN has a component for each relation. Given this picture, the presented models in this chapter are realized as a single component in LFN.

In Section 3.1, we present collaborative filtering via Euclidean embedding. In this model, a squared Euclidean distance kernel over latent variables is used to capture the relations between data. We show that this model can be used for visualization and fast recommendation retrieval. In Section 3.2 a co-clustering algorithm for a scalable online recommendation is presented. Section 3.3 is concerned with modeling and learning co-occurrence data for which it outperforms the-state-of-the-art in modeling, learning and visualization. In Section 3.4, we present an application of latent variable models in an education problem.

3.1 Collaborative Filtering

In [62], we proposed a novel model for collaborative filtering. The most widely used latent variable model for collaborative filtering is matrix factorization which uses a dot product kernel over random variables. In this work, we use Euclidean distance as a substitute for dot product. In this method, users and items are embedded in a unified Euclidean space where the distance between a user and an item is inversely proportional to the rating. This model is comparable to matrix factorization in terms of both scalability and accuracy while providing several advantages. First, the result of Euclidean embedding is more intuitively understandable for humans, allowing useful visualizations. Second, the neighborhood structure of the unified Euclidean space allows very efficient recommendation queries. Finally, the method facilitates online implementation requirements such as mapping new users or items in an existing model.

Living in the information age, people use a large variety of information. Unfortunately, the volume of information is so huge that no one can use all of it, even in a very specialized area. For example, there are tons of movies that you have not seen, but how can you decide which one is best to watch in your limited time? Therefore, customization can play an important role. The idea of a recommender system, an automatic system that can recommend an appropriate item, has emerged in response to this problem. There are two main approaches in recommendation systems: *content-based* and *collaborative filtering* [17]. In a content-based recommendation system, items are recommended based on a user profile and product information. Collaborative filtering uses similarity to recommend items that were liked by similar users.

Our primary approach is based on Euclidean embedding. One popular example of Euclidean embedding is multidimensional scaling (MDS). MDS is a branch of multivariate statistical analysis and often used to give a comprehensible visual representation. MDS has been applied in a variety of disciplines including psychology, marketing, and machine learning [13, 25]. A narrow definition of multidimensional scaling is the search for a low dimensional space, usually Euclidean, in which points in the space represent the objects, one point representing one object, and such that the distances between the points in the space match, as well as possible, the original dissimilarities [25].

In the data mining and machine learning area, Euclidean embedding has been frequently used as a visualization approach [6], a dimensionality reduction technique [34], and in unsupervised learning [31]. Euclidean embedding is rarely used as a core of a supervised learning approach. For example, Trosset et al. [101] used MDS in the first step of a two-step data mining approach where the second step is training a classifier on the result of the MDS model. The MDS step can be considered as an unsupervised learning approach. In this paper, we propose a Euclidean embedding method that can be seen as a supervised version of MDS and uses the modeling results directly in prediction.

The traditional solution approaches for MDS often include eigenvector analysis methods for the matrix of dissimilarity between objects, resulting in a complexity of $O(N^3)$ where N is the number of objects [80]. Besides the cubic complexity, the computation must be repeated if data is slightly changed [80]. As a result, iterative optimization methods have been developed [21]. Numerical optimization techniques like gradient descent have been widely used in MDS [72].

To the best of our knowledge, Euclidean embedding has not been used as a direct optimization method to implement collaborative filtering. There is a literature about visualization for recommender systems. In [51], first a collaborative filtering algorithm such as classical SVD is run and then some recommendations will be proposed to a user in a visual manner. Note that items or users are not mapped on a space. Users are able to use some meta data such as genre or language to filter their result. In other work such as [33], recommended items are visualized via MDS. However, users are not mapped at the same time. Only items are scaled using classical MDS where dissimilarities between them will be computed via their correlation.

In contrast to the literature on collaborative filtering visualization, our method embeds both users and items in a unified space. This will facilitate visualization of a target user and items he likes where distance is strongly correlated with his personal preferences.

3.1.1 Model

In this section we present collaborative filtering via Euclidean embedding. Assume that all items and users are embedded in a unified Euclidean space. Let the location show the characteristics of each person. This is a legitimate assumption since one of the early psychological applications of MDS is related to embedding people on a low-dimensional space based on their preferences [25]. A similar assumption can be set for items where the location of each movie reflects its characteristics, such as genre. Note that this is simply the meaning of latent factor model in the framework of Euclidean embedding. Therefore, if an item is close to the user in the unified space,

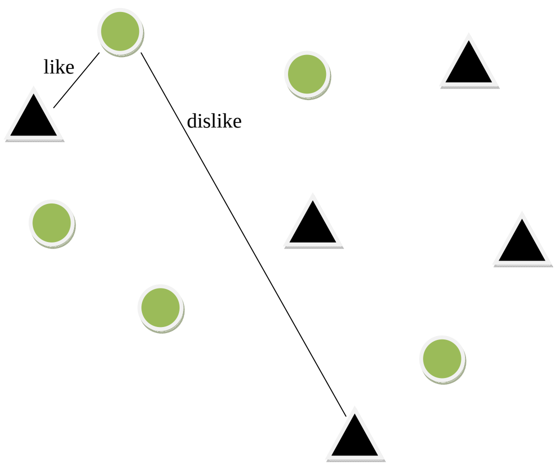


Figure 3.1: Users (circles) and items (triangles) are embedded in a unified Euclidean space

its characteristics are attractive for the user. As a result, there will be a negative correlation between the distance and the likability. Figure 3.1 represents this idea.

In the Euclidean embedding framework, equation (2.10) can be re-written as

$$\hat{r}_{ui} = \mu + b_u + b_i - (x_u - y_i)(x_u - y_i)' \tag{3.1}$$

where x_u and y_i are point vectors of user u and item i in a D-dimensional Euclidean space and $(x_u - y_i)(x_u - y_i)'$ is the squared Euclidean distance. The reason we used squared Euclidean distance instead of Euclidean distance is that the former is computationally cheaper while the accuracy of the method is empirically the same.

The main difference between the matrix factorization model and the Euclidean embedding model lies in the latent factor space characteristics. In the space of Euclidean embedding, the interpretation of user points and item points are the same, since the maximum rating can be reached when they are at the same point. However in matrix factorization the user and item space are not unified. As an example let a user be at $p = (.5, .5)$ in the matrix factorization model and $x = (.5, .5)$ in the Euclidean embedding model. The most similar item to this user in Euclidean embedding is located at $y = (.5, .5)$. However for matrix factorization there is no ideal solution and $q = (.5, .5)$ is worse than, for example, $q = (1, 1)$.

Similar to matrix factorization, the goal of the Euclidean embedding model is embedding users and movies in a low-dimensional Euclidean space based on the known ratings and then predict the unknown ratings based on the trained model. Therefore, Euclidean embedding is a supervised learning approach in which the training phase includes finding the location of each item and user to minimize a loss function. Modifying equation (2.11) to Euclidean embedding, we have:

$$\min_{x,y,b} \sum_{u,i} w_{ui}[(r_{ui} - \mu - b_u - b_i + (x_u - y_i)(x_u - y_i)')^2 + \lambda(\|x_u - y_i\|^2 + b_u^2 + b_i^2)]. \tag{3.2}$$

We control the magnitude of the $(x_u - y_i)$ instead of individual points since the distance does not depend on the absolute value of the points but the relative position of x_u to y_i.

This optimization problem is different from typical MDS problems in several respects. First, here we are embedding two different kinds of objects (items and users) in a low-dimensional space, while in standard MDS there is only one object type. Second, the rating matrices are always extremely sparse, which makes the matrix operations less trustable. Finally, in addition to coordinates of points, the b parameters are also being optimized, while in MDS, only the object locations are of interest. Therefore, conventional MDS techniques cannot be applied directly in our Euclidean embedding problem.

Using gradient descent to minimize the Euclidean embedding objective function (equation 3.2), updates in each step can be defined as

$$\begin{array}{lcl} b_u & \leftarrow & b_u + \gamma(e_{ui} - \lambda b_u) \\ b_i & \leftarrow & b_i + \gamma(e_{ui} - \lambda b_i) \\ x_u & \leftarrow & x_u - \gamma(x_u - y_i)(e_{ui} + \lambda) \\ y_u & \leftarrow & y_u + \gamma(x_u - y_i)(e_{ui} + \lambda) \end{array}$$

where γ is the step size.

3.1.1.1 Time complexity

For comparing the time complexity of Euclidean embedding and matrix factorization models, three tasks are important:

1. Training
2. Prediction

For training, if we pre-compute the $(x - y)$ terms in each step, the other operations will be almost the same as matrix factorization. Therefore, Euclidean embedding needs D (the dimension of space) more operations than matrix factorization. The time complexity of both methods for a step of an iteration is $O(D)$. The total number of iterations to converge depends on the algorithmic parameters.

For the prediction of a rating, with a similar approach as the training task, we need only D more operations and the time complexity of prediction for both models is $O(D)$. However, we can decompose $(x-y)(x-y)'$ to $xx' + yy' - 2xy'$. xx' and yy'

can be pre-computed when the system is off-line and then the number of operations for both models in the online phase will be exactly the same.

3.1.1.2 Visualization

Another advantage of Euclidean embedding is its representation. However, to represent items to users, Euclidean embedding must be implemented only using 2 or 3 dimensions. This can deteriorate the accuracy of recommendations. To avoid this problem, we use the following strategy:

1. Implement CF via Euclidean embedding in a high-dimensional space;
2. Select the top k items for an active user;
3. Embed user, selected items, and some favorite items in a 2-dimensional space via classic MDS, using distances from the high-dimensional space in step 1.

Note that in the third step, MDS can be run very fast because there are only a few objects to embed, all distances are known (from the high-dimensional space solution), and all distances satisfy the triangle inequality (they are real distances in the high-dimensional space).

While classic MDS has been used to visualize items in a CF setting, existing methods use correlation between items as a similarity measure. Using this approach it is not possible to embed a user in the same space because correlation is not defined between an item and a user. Therefore, the picture only shows what items are similar to each other, but it does not indicate how much a user might like each movie. Also sparseness may cause the distance based on correlation to be unknown, and correlation

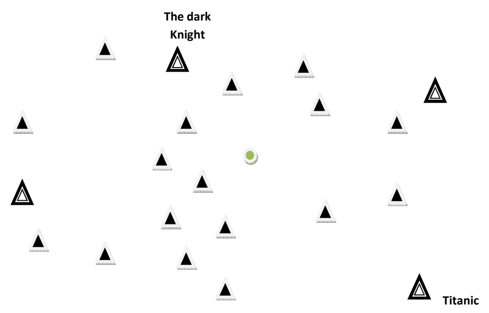


Figure 3.2: Representing close items (triangles) to a user (circle) besides the movies he has already liked (bold triangles) to assist him in selection

does not satisfy the triangle inequality which makes implementing MDS harder.

Using this low-dimensional unified user-item space, we can represent items to users via a graphical interface such as Figure 3.2. Highly-rated items can be presented to give an idea about movies. For example a user might be in a mood to watch a romance movie so he can check the movies near to his favorite romance movies.

3.1.1.3 Fast recommendation generation

Although both matrix factorization and Euclidean embedding algorithms are fast in predicting a rating, the main task of a recommendation system is finding desirable items for a query user. This problem is rarely addressed in the literature. Das et al. [27] proposed some strategies to select candidate items in the context of a news recommendation system in which news items are selected based on a content-based criteria. Also, they proposed to select items that are chosen by a cluster of users. However, these approaches are not applicable to our problem since we are not using any extra information about items, and we choose to use highly accurate latent factor models' features in the selection task.

A key advantage of Euclidean embedding over matrix factorization is that the nature of the mapped space allows candidate retrieval via neighborhood search. Consider an example with user v as the query user, and $D = 2$. Also let $p_v = [.5\ .5]'$ in the matrix factorization model and $x_v = [.5\ .5]'$ in the Euclidean embedding model. In the Euclidean embedding model, the smaller the distance, the more desirable an item will be. Therefore, we need only search for movies near the point $[.5\ .5]'$. On the other hand, for the matrix factorization model, the larger the value of the term $.5(q_{i1}) + .5(q_{i2})$ (where q_{i1} and q_{i2} are movie coordinates), the more desirable the item will be. Therefore, a large area of space may be possible. Figure 3.3 illustrates the difference between matrix factorization and Euclidean embedding search space for a query user.

Therefore, using Euclidean embedding, the top k closest items to an active user can be found via K-nearest neighbors search. Then, ratings for these top k selected items can be estimated and the ranked list can be presented to the user. There is a highly developed literature on spatial indexing and searching for nearest neighbors, using structures such as R-trees [45] which can be used to find recommendation candidates efficiently.

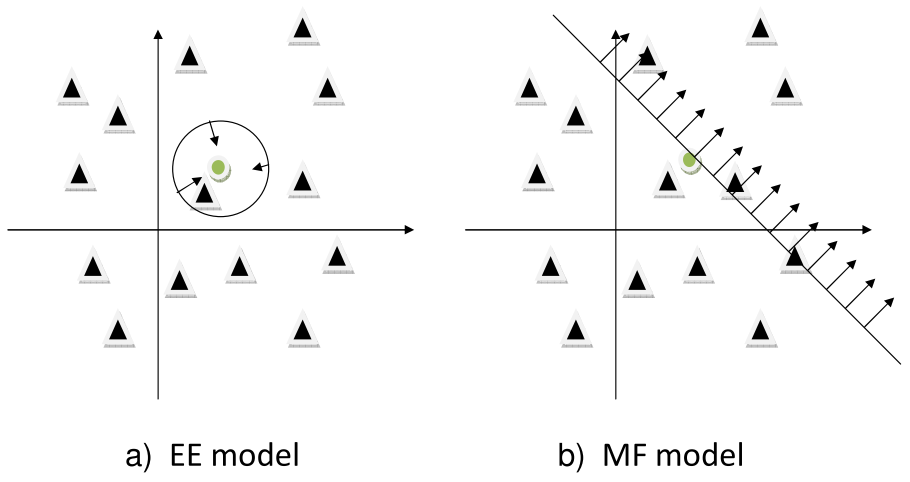


Figure 3.3: The search space for a query user

3.1.1.4 Incorporating new users and items

One drawback of model-based approaches is that if a new user or item arrives, it is hard to incorporate them into the model. However, in practice there are always new users and items which must be considered in the recommendation system. This problem has been addressed as *incremental collaborative filtering* in the literature. George and Merugu [38] used co-clustering as a very scalable incremental collaborative filtering approach. They showed the result of incremental co-clustering is comparable to incremental SVD while co-clustering is more scalable. However, they did not address the problem of incorporating new users and items in their model and only updated relevant parameters for old users and items. Sarwar et al. [95] and Brand [15] proposed using singular value decomposition as an online collaborative filtering strategy. In this work, the original version of SVD is optimized using eigenvaluevector operations. Furthermore, since it is hard to apply classical SVD on sparse data, Sarwar et al. and Brand used imputation to fill in unknown ratings which results in very poor accuracy. Then they used algebraic manipulation to map new users/items in the existing space. However, their approach is not applicable to the MF algorithms which use gradient descent since the user and item factor vectors are not orthogonal.

Generally, for a new user or item, there are $D + 1$ unknown values (D for the vector p or q and one for the scalar b). So if there are K ratings for the new object, then we have a $(D + 1) \times K$ system of equations (for Euclidean embedding these equations are non-linear). To have a unique estimation, at least $D + 1$ ratings are required. However, with fewer than $D + 1$ ratings, it is still possible to have an estimation via a ridge regression approach where the accuracy might compensate. The values used for dimension in the literature are always very high (more than 50 factors). For new items, this number of ratings can be gained easily since the number of users usually is more than the number of items. However, gathering this number of ratings might be very tedious for a new user.

In practice many users tend to rate favorite movies first. Also, a recommender system may employ an active learning strategy by asking new users to provide their favorite items. This way, since we know the point vector of the items in the space, and it is very probable that the new user is very close to her favorite items in the Euclidean embedding space, we can estimate the user vector by

$$x_u = \frac{\sum_j y_j}{n} \tag{3.3}$$

where j indexes the items that the new user u has selected as her favorites and n is the number of selected items.

In the matrix factorization model, knowing the favorite items for a new user may not be as helpful as in Euclidean embedding, since closeness is not related to likability. However, we can argue that the items similar to the favorites are probably interesting as well. As a result, a similar function to equation (3.3) can be used in the MF model. We address the power of both methods in the next section.

3.1.2 Experimental results

In this section, we present the results of experiments performed to evaluate the effectiveness of the presented methods. All of the experiments are implemented via MATLAB on a 3.2 GHz PC with 4 GB RAM.

In these experiments, two datasets were used. The first dataset is the popular Netflix dataset which consists of 17770 movies, around 480,000 users, and around 100,000,000 ratings, which we only used for comparing accuracy (RMSE). We set step size $\gamma = .005$, dimension $D = 50$, and the regularization parameter $\lambda = .005$ for Euclidean embedding and $\lambda = .01$ for matrix factorization (for MF we used the values from literature which gives the best result). The RMSE on the probe dataset for MF is .9097 (after 23 iterations) and for EE is .9124 (after 27 iterations). The results are similar despite the fact that fine-tuning was not performed on the Euclidean embedding model.

For exploring the ideas presented in this paper, we used the Movielens dataset[1] consisting of 100,000 ratings (1-5 scale) by 943 users on 1682 movies. We employed a 5-fold cross-validation strategy, and all of the values presented in this section (including time, accuracy, etc.) are the average values of the five folds. We set step size $\gamma = .005$ and dimension $D = 50$ (unless otherwise is noted), the regularization parameter $\lambda = .03$ for Euclidean embedding and $\lambda = .04$ for matrix factorization. As a stopping criteria for the gradient descent algorithm, we used a tuning set with a size of 5% of the whole training dataset.

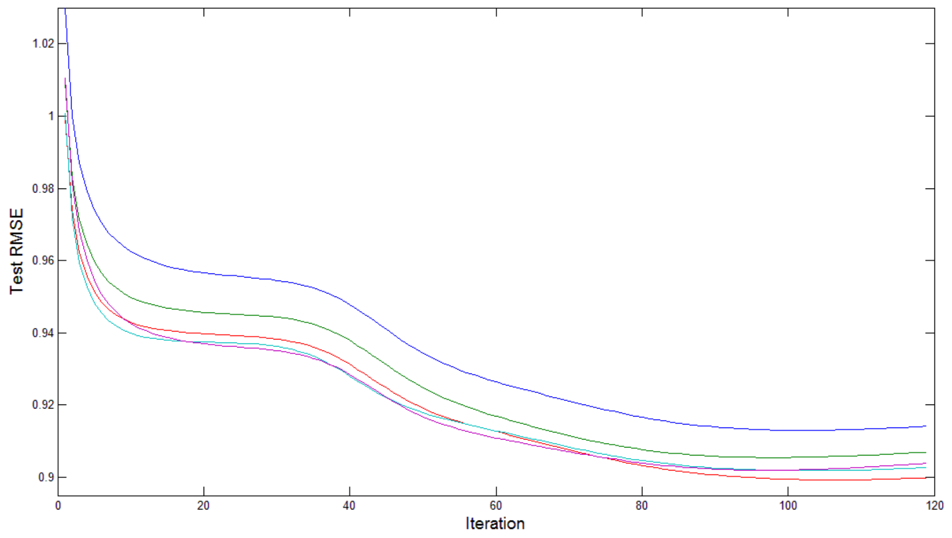
Learning curve: Figure 3.4 shows the values of RMSE in the test dataset as the gradient descent algorithm proceeds. Note that the shapes of these curves are very similar. The only difference is that MF is more prone to overfitting; as it passes the optimal point, the error increases faster.

	MF			EE		
Dimension	5	25	50	5	25	50
RMSE	0.9175	0.9107	0.9097	0.9157	0.9104	0.9086
Iteration	105.6	95.0	96.6	93.2	100.4	98.2
Time (sec)	0.4810	0.5221	0.6068	0.4811	0.5226	0.6074

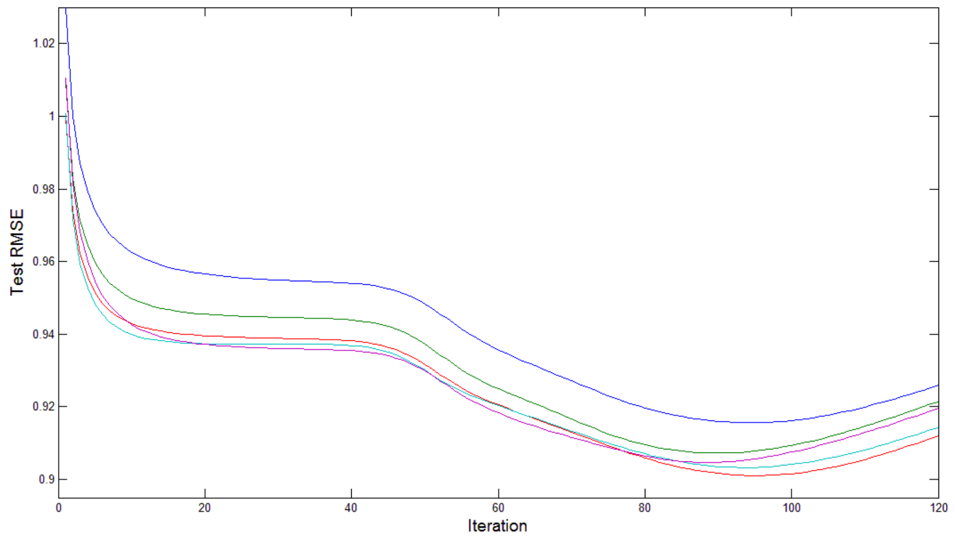
Table 3.1: RMSE, average number of iterations, and average time per iteration for MF and EE in 5, 25, and 50 dimensions

Dimension, accuracy, time: Table 3.1 shows the result of implementing

[1]http://www.grouplens.org/data/



a) EE



b) MF

Figure 3.4: Test RMSE of EE (a) and MF (b) in each iteration of the gradient descent algorithm for five different folds

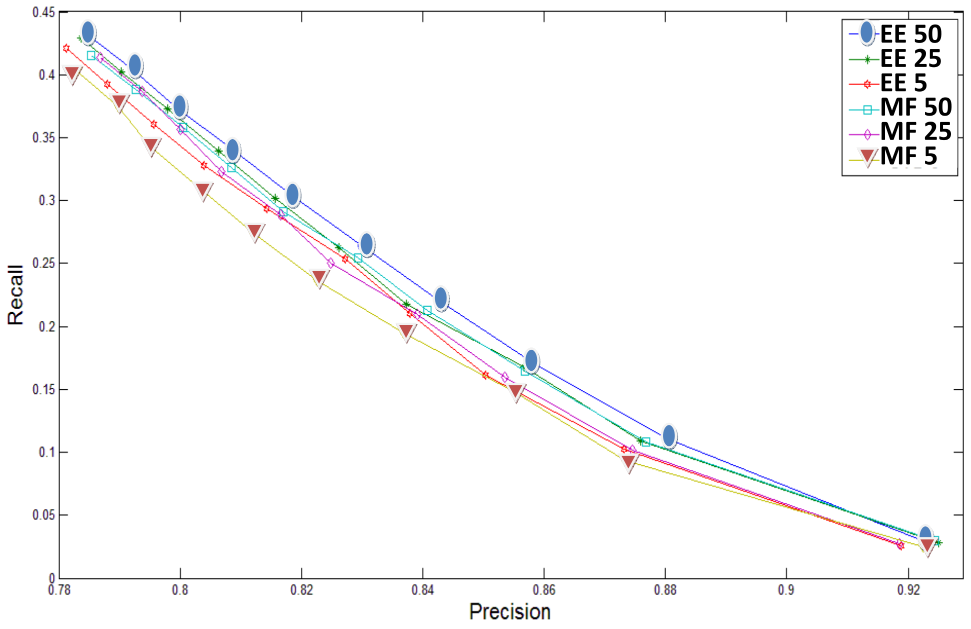


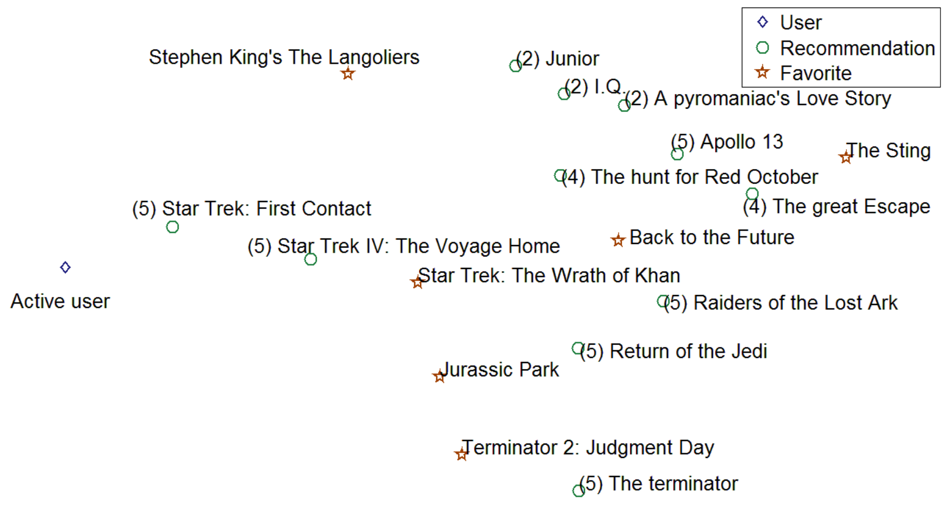
Figure 3.5: Precision/recall curve for different dimension size

EE and MF in 5, 25, and 50 dimensions. Again, both methods give similar results in the sense that increasing the dimension adds little to time requirements, and the accuracy gain from larger dimensionality is small beyond 25-50 dimensions.

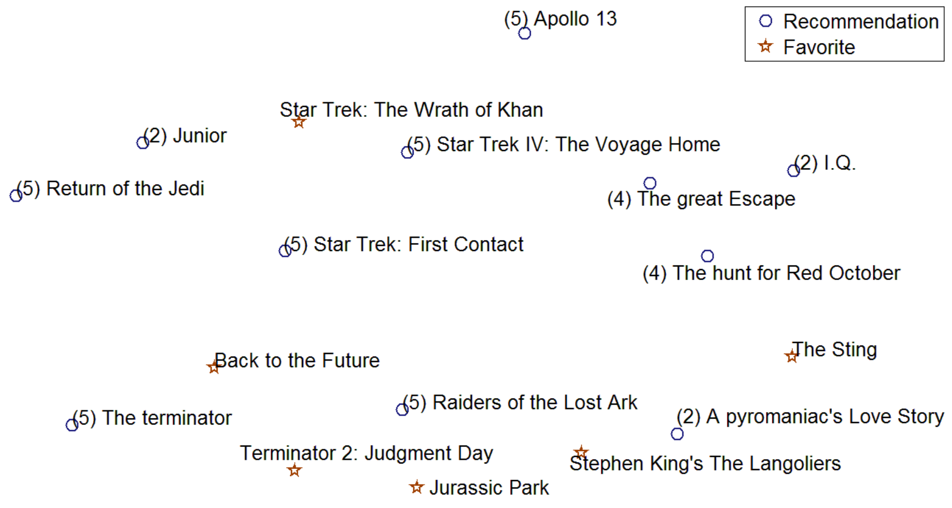
Besides RMSE, another set of measurements that are popular in retrieval problems is precision and recall. Since the main goal of recommender systems is suggesting some items of interest, it is important to measure what percent of recommendations are desirable (precision) and what percent of interesting items are retrieved (recall). Here, ratings of 4 and 5 are considered as desirable. Figure 3.5 shows the precision and recall curve for different dimensions and different methods. The values are the precision and recall for the top-k recommendation scenario for different values of k. As k increases, recall will increase while precision decreases. Again, it seems the overall accuracy of both EE and MF are not very sensitive to the size of dimension. In general, EE performs better than MF.

Visualization: Figure 3.6 shows user visualizations constructed in the following manner. For a typical user, the top 50 movies were selected based on Euclidean embedding with $D = 50$ dimensions. Then, the active user, the top selected items whose ratings were known in the test dataset, and items which were rated as 5 by the active user in the training dataset were embedded in a 2-dimensional space. As a distance we used the Euclidean distance from the primary high-dimensional space. For the sake of comparison, we scaled the same items via MDS using $1-$correlation as the distance. Note that in the comparison method, the active user cannot be embedded. In visualization based on Euclidean embedding, movies that are closer to the active user are more probable to be liked. However, using classic MDS, the picture is harder to interpret since movies can only be compared to each other. For example, disliked movies “Junior,” “IQ,” and “A Pyromaniac’s Love Story” are close to each other and far away from the active user in the first picture, but scattered in the second. The main reason is that, in the first picture items are embedded based on the taste of the active user while in the second picture it is based on the taste of all users.

Generating fast recommendations: As mentioned earlier, finding new recommendations for a user in the EE problem can be treated as a k-nearest neighbors search problem in a Euclidean space. Table 3.2 summarizes the result of top-10 recommendation to all users. In MF and EE we simply used exhaustive search to find best recommendations. For EE-KNN, first 100 movies for each user were selected as



a) Euclidean embedding



B) Classical MDS using correlation as a similarity measurement

Figure 3.6: Visualization of movies using Euclidean embedding and classic MDS

candidates using a brute search K-nearest neighbor search algorithm. Also we found it useful to filter out movies with parameter b_i less than the average of parameter b_i for all items. Then ratings were estimated for all candidates and the top 10 were selected.

In these experiments we used dimension $D = 50$. Note that since we used exhaustive search, the time complexity is a linear function of the number of movies for each user.

	Prec.	Recall	Time (Sec)
MF	0.9065	0.0521	18.2455
EE	0.9063	0.0525	19.5189
EE-KNN	0.9012	0.0423	0.5396

Table 3.2: Precision and recall for simple MF, EE, and EE with candidate generation using KNN-search ($D = 50$ for all runs)

As Table 3.2 shows, the search time can be decreased drastically using KNN-search, while the accuracy is competitive.

New users: As discussed in earlier, if we ask a new user to provide a list of favorite movies, we could quickly map the user in the existing space. To simulate this setting, we selected 224 of the 943 users in the Movielens dataset, and 3 movies with ratings of 5 for each user were randomly selected to simulate the list of favorite movies. Then the space was learned via the ratings of the rest of the users. Afterwards, we estimated the point vector of a user by simple averaging for both EE and MF. The

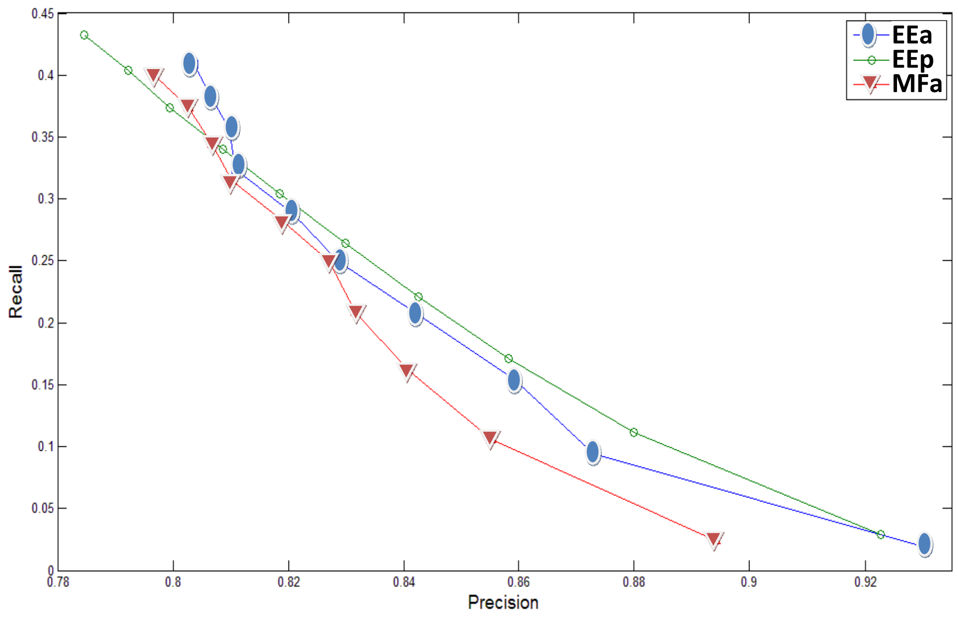


Figure 3.7: Precision and recall for EEa and MFa

result is shown in Figure 3.7. MFa and EEa (a for mapping points via averaging) implement averaging for new users. EEp represents the precision/recall values for the regular settings when users are not new to the system and we included it for the sake of comparison.

As Figure 3.7 shows, EEa performs extremely well. Especially its top-5 recommendation precision (on the high-precision end of the plot) is remarkable. When retrieving a small number of recommendations, the precision of Euclidean embedding is much larger than MF, while the recall is about the same.

3.1.3 Conclusion

In this work, we proposed a novel collaborative filtering algorithm based on multidimensional scaling, a latent variable model. We showed that CF via Euclidean embedding is comparable to one of the most accurate MF-based algorithm in both accuracy and scalability, while it provides some advantages. First, using Euclidean embedding, finding recommendations is equivalent to a k-nearest neighbor search in a metric space which can be performed very fast. Second, using the fact that closeness in the transformed space corresponds to higher ratings, we proposed a very simple and fast approach to incorporate new users by asking them to provide their favorite items.

3.2 Incremental Collaborative Filtering

In this work we proposed an online algorithm for collaborative filtering[59] using the algorithm we presented in [58]. Most collaborative filtering algorithms are accurate but also computationally expensive, and so are best in static off-line settings. It is desirable to include the new data in a collaborative filtering model in an online manner, requiring a model that can be incrementally updated efficiently. Incremental collaborative filtering via co-clustering has been shown to be a very scalable approach for this purpose. However, locally optimized co-clustering solutions via current fast iterative algorithms give poor accuracy. Here, we present an evolutionary co-clustering method that improves predictive performance while maintaining the scalability of co-clustering in the online phase.

A few published approaches have addressed the incremental CF problem. Sarwar et al. [95] and Brand [15] proposed using singular value decomposition as an online CF strategy. Das et al. [27] proposed a scalable online CF approach using MinHash clustering, Probabilistic Latent Semantic Indexing (PLSI), and co-visitation counts. However in this work, only binary ratings (such as implicit user feedback) were considered and therefore the proposed approaches are not applicable to prediction problems. In K-nearest neighbors collaborative filtering approaches, similarity parameters such as correlation can be updated incrementally during the online phase [87].

George and Merugu [38] used co-clustering as a scalable incremental CF approach for dynamic settings. They showed the performance of incremental co-clustering is comparable to incremental SVD but much more scalable. For implementing co-clustering they used the approach of Bregman co-clustering [5], a very fast iterative local search which can result in very poor local optima.

In this paper we propose an incremental CF method that is both scalable and accurate. We use an evolutionary co-clustering algorithm that finds better solutions than Bregman co-clustering at the cost of offline training time, which is relatively unimportant. Also, we revise the incremental algorithm suggested in [38] and introduce an ensemble strategy to give better predictions.

In a collaborative filtering problem, there are U users and V items. Users have provided a number of explicit ratings for items; r_{ui} is the rating of user u for item i. There are two phases in a CF algorithm: an offline phase in which training based on known ratings is performed, and an online phase in which unknown ratings are estimated using the output of the offline phase. Most CF approaches use only the data available offline to predict ratings. In incremental CF, the data available during online phase is incorporated into future predictions, potentially improving the predictive accuracy.

3.2.1 Baseline algorithm

The simplest way to predict a rating is the global average of all ratings. However, some users tend to rate higher and some items are more popular. Including user bias and item bias in rating, we can predict user ratings by

$$\hat{r}_{ui} = (1 - S_{n_u,\omega} - S_{n_i,\omega})\bar{r} + S_{n_u,\omega}\bar{r}_u + S_{n_i,\omega}\bar{r}_i \tag{3.4}$$

where $\bar{r}$ is the global average, $\bar{r}_u$ is the average of ratings by user u, $\bar{r}_i$ is the average of ratings for item i, n_u is the number of ratings by user u, and n_i is the number of ratings for item i. $S_{n_u,\omega}$ and $S_{n_i,\omega}$ are the support function for user u and item i which we define as

$$S_{v,\omega} = \begin{cases} \frac{v}{\omega} & if \quad v < \omega, \\ 1 & otherwise. \end{cases} \tag{3.5}$$

If fewer ratings are available for a user or item, the support will be smaller. The parameter ω determines the necessary support. Empirically we found three to be a robust choice for ω. When the support of a user and an item is zero, then $\hat{r}_{ui} = \bar{r}$; when the support is one, $\hat{r}_{ui} = \bar{r}_u + \bar{r}_i - \bar{r}$; and when the support of a user u is zero and an item i is one, then $\hat{r}_{ui} = \bar{r}_i$, and vice versa for the reverse.

3.2.2 Incremental CF via co-clustering

Clustering refers to partitioning similar objects into groups [5]. Co-clustering partitions two different kinds of objects simultaneously. If one views the clustering problem as grouping rows of a matrix together, then co-clustering is the simultaneous grouping of rows and columns.

In collaborative filtering via co-clustering as suggested in [38], each user u is assigned to a user cluster (represented by $\rho(u)$) and each item i is assigned to an item cluster (represented by $\gamma(i)$) and the prediction is as follows:

$$\hat{r}_{ui} = \bar{r}_{kl} + (\bar{r}_u - \bar{r}_k) + (\bar{r}_i - \bar{r}_l), \tag{3.6}$$

where $k = \rho(u)$ is the user cluster assigned to user u, $l = \gamma(i)$ is the item cluster assigned to item i, $\bar{r}_{kl}$ is the average of ratings belonging to users in user cluster k and items in item cluster l, $\bar{r}_k$ is the average of ratings belonging to users in user cluster k, and $\bar{r}_l$ is the average of ratings belonging to items in item cluster l. The term $(\bar{r}_u - \bar{r}_k)$ tries to remove the bias of user u (some users tend to rate higher) and $(\bar{r}_i - \bar{r}_l)$ is the same for item i (some items are more likable).

George and Merugu [38] used a fast iterative heuristic proposed by Banerjee et al. [5]. This algorithm has two phases: updating user clusters and updating item clusters. When updating user clusters, we assume all co-cluster means are constant and all items are assigned to item clusters, then we assign each user so that the sum of squared errors is minimized. Item cluster updating follows the same approach. Details can be found in [5, 38].

In the online phase, the prediction is as follows:

$$\hat{r}_{ui} = \begin{cases} \bar{r}_{kl} + (\bar{r}_u - \bar{r}_k) + (\bar{r}_i - \bar{r}_l) & \textit{if (oldUser-oldItem)} \\ \bar{r}_i & \textit{if (newUser-oldItem)} \\ \bar{r}_u & \textit{if (oldUser-newItem)} \\ \bar{r} & \textit{if (newUser-newItem)} \end{cases} \tag{3.7}$$

In [38], incremental training is achieved by using new ratings to update the average parameters ($\bar{r}_{kl}$, $\bar{r}_u$, $\bar{r}_k$, $\bar{r}_i$, $\bar{r}_l$) in equation (3.6). However, new users or items are not assigned to clusters during the online phase.

3.2.3 Model

As our experimental results will show, incremental collaborative filtering via co-clustering [38] discussed in Section 3.2.2 may be even worse than the baseline. Therefore, we propose a number of revisions to improve this method.

First, if the support (number of available ratings) for a user or item is low, the co-clustering approach will not provide good predictions for them, and the training phase will be affected by these noisy inputs. As a strategy, users and items with low support are eliminated from the training phase so that training is both more effective and efficient.

The prediction equation (3.6) is not appropriate for the incremental scenario, because it incorporates three average parameters ($\bar{r}_{kl}$, $\bar{r}_k$, $\bar{r}_l$) from a co-clustering solution that is not necessarily reliable. This is due to the fact that $\bar{r}_k$ and $\bar{r}_l$ tend to be very close to the global average empirically. Therefore, this model can be equivalent to $\hat{r}_{ui} = \bar{r}_{kl} + \bar{r}_u + \bar{r}_i - 2\bar{r}$ during the online phase. On the other hand, using only the block average $\bar{r}_{kl}$ for prediction ignores user and item bias which results in poor accuracy as well. To overcome this problem, we begin by co-clustering residuals, rather than ratings. We model a rating prediction as

$$\hat{r}_{ui} = \bar{r}_u + \bar{r}_i - \bar{r} + \varepsilon_{ui}. \tag{3.8}$$

Note that $\bar{r}_u + \bar{r}_i - \bar{r}$ is the same as the prediction equation (3.4) when the support function is 1. As a result, ε_{ui} is the correction parameter for (3.4). For a known rating, (3.8) can be rewritten as

$$\varepsilon_{ui} = r_{ui} - (\bar{r}_u + \bar{r}_i - \bar{r}) \tag{3.9}$$

where ε_{ui} can be interpreted as the residual of the prediction via (3.4). For implementing co-clustering, it is enough to work with the following objective function:

$$\min_{\rho,\gamma} \sum_u \sum_i w_{ui}(\varepsilon_{ui} - \bar{\varepsilon}_{\rho(u)\gamma(i)})^2 \tag{3.10}$$

where w_{ui} is one if rating r_{ui} exists in training dataset and otherwise is zero. $\bar{\varepsilon}_{\rho(u)\gamma(i)}$ is the block average of residuals for user cluster $\rho(u)$ and item cluster $\gamma(i)$.

Now we can define the prediction strategy based on (3.4) and (3.8). For old user - old item,

$$\hat{r}_{ui} = \bar{r}_u + \bar{r}_i - \bar{r} + \bar{\varepsilon}_{\rho(u)\gamma(i)} \tag{3.11}$$

and otherwise

$$\hat{r}_{ui} = (1 - S_{n_u,\omega} - S_{n_i,\omega})\bar{r} + S_{n_u,\omega}\bar{r}_u + S_{n_i,\omega}\bar{r}_i. \tag{3.12}$$

As mentioned, one advantage of co-clustering is scalability. Therefore creating an ensemble of different co-clusterings is desirable. Ensembles are used to improve the accuracy of a method using a group of predictors, while increasing the running time linearly with the number of ensemble elements. Let p denote a co-clustering solution and P be the number of co-clusterings we use in the model. We can predict with

$$\hat{r}_{ui} = \bar{r}_u + \bar{r}_i - \bar{r} + \frac{\sum_p \exp(-z_{ulp} - z_{ikp})\bar{\varepsilon}_{klp}}{\sum_p \exp(-z_{ulp} - z_{ikp})} \tag{3.13}$$

where z_{ulp} is the average error of prediction for user u and item cluster l in co-clustering solution p and similarly z_{ikp} is the average error of prediction for item i

and user cluster k in co-clustering solution p. Intuitively if previous predictions of a co-clustering solution are better, its weight is higher.

In the algorithm proposed in [38], new users and items are not included in the co-clustering during the online phase. As a result, the model is unable to provide legitimate predictions for them. However, in the revised version, it is trivial to find an appropriate cluster for a user or item. Let u be a new user who has provided some ratings. If a sufficient number of rated items exist in the current co-clustering solution, then the new user's cluster can be found using [5]:

$$\rho(u) = \arg\min_{g} \sum_{h} n_{uh}(\bar{\varepsilon}_{uh} - \bar{\varepsilon}_{gh})^2 \tag{3.14}$$

where n_{uh} is the number of times user u has rated the items belonging to item cluster h during the online phase, and $\bar{\varepsilon}_{uh}$ is the average of residuals for those ratings. A similar procedure finds the cluster of a new item. Figure 3.8 shows the incremental training algorithm. The function $numberIn()$ is the number of ratings a user (item) has in the co-clustering solution which is defined by $\sum_h n_{uh}$ ($\sum_g n_{ig}$) for user u (item i). If this value is bigger than a threshold τ, then we can trust the information to incorporate the new user or item. Otherwise, the risk of misclustering will be high. Subsequently, new users and items will not receive prediction from co-clustering at the very beginning. These ratings will be estimated using (3.4), which is more accurate experimentally. As a rule of thumb, τ can be set to 3 since we wish to incorporate new users or items as soon as possible but not at the cost of bad predictions.

We now turn to the construction of co-clusterings via evolutionary algorithms, a population-based search approach that explores a solution space by evolving a group

```
Input:  new rating, co-clustering ($\rho$,$\gamma$)
Update user average $\bar{r}_u$
Update item average $\bar{r}_i$
IF {old user - old item}
   Update co-cluster mean $\bar{\varepsilon}$ and weights $z$
ELSEIF {old user - new item} AND {numberIn(item)>$\tau$}
   Assign the item to an item cluster
   Update co-cluster mean $\bar{\varepsilon}$ and weights $z$
ELSEIF {new user - old item} AND {numberIn(user)>$\tau$}
   Assign the user to a user cluster
   Update co-cluster mean $\bar{\varepsilon}$ and weights $z$
ENDIF
```

Figure 3.8: Incremental training in evolutionary co-clustering

of individuals to find good solutions. In our context, let P co-clustering solutions exist. The goal is to find better solutions by combining the current solutions. Every evolutionary algorithm has three main steps: selection, in which two or more individuals are chosen to create offspring; crossover, in which the selected items are combined to create new solutions; and replacement, in which the new solutions replace existing solutions if they satisfy some criteria.

Our evolutionary co-clustering algorithm is shown in Figure 3.9. A group of co-clustering solutions is randomly generated and locally optimized via iterative Bregman co-clustering. The numbers of user and item clusters are randomly selected from a specific range. In the evolutionary iteration phase, two co-clustering solutions are randomly selected for crossover. In the context of optimization, better individuals are

```
Input:  Population size $P$, Rating matrix


Initialization:
  FOR $p = 1$ to $P$
      $K_p \leftarrow RandomInteger(1, \beta_1)$
      $L_p \leftarrow RandomInteger(1, \beta_2)$
      $\forall u:\ \rho_0(u) \leftarrow RandomInteger(1, K_p)$
      $\forall i:\ \gamma_0(i) \leftarrow RandomInteger(1, L_p)$
      $(\rho_p, \gamma_p) \leftarrow LocallyOptimizationCooclustering(\rho_0, \gamma_0)$
  ENDFOR
Evolutionary iteration:
Repeat
  Select 2 co-clusterings $(\rho_q, \gamma_q)$ and $(\rho_r, \gamma_r)$ randomly
  $K_o \leftarrow \frac{K_q+K_r}{2} + RandomInteger(-\beta_3, \beta_3)$
  $L_o \leftarrow \frac{L_q+L_r}{2} + RandomInteger(-\beta_4, \beta_4)$
  $\rho_o \leftarrow crossover(\rho_q, \rho_r)$
  $\gamma_o \leftarrow crossover(\gamma_q, \gamma_r)$
  $(\rho_o, \gamma_o) \leftarrow LocallyOptimizationCooclustering(\rho_o, \gamma_o)$
  DiscardWorst $\{(\rho_p, \gamma_p)_{p=1}^{P}, (\rho_o, \gamma_o)\}$
Until{convergence OR iter>maxIter}
```

Figure 3.9: Evolutionary algorithm

chosen with higher probability. However, in machine learning, generalization is more important than optimization and biased selection may result in premature convergence. Our initial studies indicated that randomly selecting co-clusterings improved generalization. After selecting two co-clusterings, a new solution is generated via the crossover function presented in Figure 3.10.

The crossover operation is between two clusterings ϕ_1 and ϕ_2. If the required

```
Input:  clusters $\phi_1$ and $\phi_2$,required cluster size $K$
Output:  offspring cluster $\phi_o$

$\forall x, q, r : \zeta_{qr} \leftarrow x | \phi_1(x) = q \ \& \ \phi_2(x) = r$
FOR $k = 1$ to $K - 1$
    $\forall x \in \zeta^{(k)} : \ \phi_o(x) \leftarrow k$
ENDFOR
$\forall x \notin \cup_{k=1}^{K-1} \zeta^{(k)} : \ \phi_o(x) \leftarrow K$
```

Figure 3.10: Crossover algorithm

number of clusters is K, the $K-1$ largest intersections between ϕ_1 and ϕ_2 are assigned to the first $K-1$ clusters and the remainder will be assigned to the last cluster. Formally, let X be a $N \times K$ assignment matrix in which an element (u, k) is one if object u is assigned to cluster k and zero otherwise. Then the intersection matrix can be defined as $X'X$.

In the next step, the offspring from crossover will be locally optimized via the fast iterative Bregman co-clustering algorithm from [5]. This is an important step due to the fact that most objects will be assigned to the last cluster. However, since the iterative co-clustering first estimates averages and then assigns users and items, we hope that most of the blocks will have good quality averages via fewer but selected users or items.

Finally, we should either discard a current solution or the offspring to preserve the total number of solutions. The following function can be used to discard the worst solution:

$$\widetilde{p} = arg\max_p \sum_u \sum_i w_{ui}(\varepsilon_{ui} - \frac{\sum_{p' \neq p} \bar{\varepsilon}_{\rho_{p'}(u)\gamma_{p'}(i)}}{P}).$$

where $\widetilde{p}$ represents the worst solution in the population. If the worst solution is the offspring we just discard it, otherwise the worst solution will be replaced with the offspring.

3.2.4 Experimental results

In this section, we present the results of experiments performed to evaluate the effectiveness of our method. We used the Movielens dataset[2] consisting of 100,000 ratings (1-5) by 943 users on 1682 movies. We used mean absolute error (MAE) to evaluate and compare different methods. Four methods were used for comparison:

1. Baseline: based on the model proposed in Section 3.1.
2. COCL: The method presented in [38].
3. ECOCL: Evolutionary co-clustering without ensembles.
4. ECOCLE: Evolutionary co-clustering with ensembles.
5. IKNN: Incremental KNN method [87].
6. SVD: We compared our results with those from [95] for SVD on the same dataset and similar settings.

In our experiments, we used 5-fold cross-validation. First a part of data was held for offline training. Then the rest of data was included in an online phase which is a combination of incremental training and prediction. In the online phase, first a

[2]http://www.grouplens.org/data/

predicted rating was used for computing prediction error, and then the new case was incorporated into the model. The sequence of data was randomized. We performed incremental training based on three different strategies: “20%-80%”, in which 20% of data was used for training and 80% for incremental training; “50%-50%,” and “80%-20%.”

All of the support parameters such as ω in (3.4) were set to 3. For the COCL method, we implemented the algorithm for different user and item cluster numbers and the best result (10 user clusters and 2 item clusters) is reported. The number of ensembles for ECOCLE was set to 25 and the iteration limit was 250.

The results for all methods are summarized in Table 3.3. First, the baseline is reasonably good compared to other methods when less data is available during the training phase. As more data is provided for the offline phase, other methods are more accurate than the baseline. Also, evolutionary co-clustering algorithm (ECOCLE) is more successful when more data is available. Using ECOCLE in all phases gives the best results. Evolutionary co-clustering without ensembles (ECOCL) still outperforms other methods while its performance is slightly better than baseline for the 20%-80% case. We did not perform SVD and only report the result of Sarwar et al. [95]. For 20%-80%, SVD has the poorest performance. However, as more data is available for training, it gets more competent. Note that the experimental protocol of [95] was different than ours in that new users and items were incrementally added to the model in one step, based on a training/test split. However, it is non-trivial to update an SVD model based on new data. Therefore, the performance of incremental

	20%-80%	50%- 50%	80%-20%
ECOCLE	.7563	.7284	.7155
ECOCL	.7626	.7433	.7336
Baseline	.7645	.7586	.7555
COCL	.7781	.7560	.7513
IKNN	.7646	.7489	.7436
SVD	$> .79$	$> .75$	$> .73$

Table 3.3: Average MAE of different methods

	offline	online
ECOCLE	3.350	1.872
ECOCL	1.783	.008
Baseline	.000	.006
COCL	.008	.009
IKNN	.053	1.532

Table 3.4: Average time (milliseconds) of different methods per rating

SVD in our protocol might be similar.

The time of both offline and online training is reported in the Table 3.4. The offline phase of ECOCLE needs more time due to the evolutionary algorithm. However, since this phase only needs to be done once, greater offline training time can be ignored. Online time is the sum of both incremental online training and online prediction. ECOCLE and IKNN have similar online speeds, while the accuracy for ECOCLE is much higher. The time problem could be mitigated by parallelizing the co-clustering operations, since updating is independent for the different solutions in the ensemble.

3.2.5 Conclusion

Online collaborative filtering methods that can incorporate new data in real time are advantageous in many practical situations. In this work, we extended the idea of CF via co-clustering to fulfill this need. As our empirical results showed, our method achieved very good accuracy compared to other incremental methods. Training was comparatively slow, but still manageable, and could be improved by a straightforward parallelization.

3.3 Latent logistic allocation

In [61], we proposed a kernel-based generative probabilistic model for learning co-occurrence data. In co-occurrence data, there are a number of entities and the data includes the frequency of two entities co-occurring. In LLA, similar to other latent variable approaches, we assume that the position of the entities in a latent space is responsible for generating the co-occurrence data. Therefore, the main task of learning is to recover the latent variables for each entity. We exploit a kernel-based approach in which the relationship between entities is captured via a kernel which is a function of the position of entities in the space. We explore two different kernels, linear and radial basis function, for implementing LLA. Given the intractability of inference for the posterior distribution, we use approximate inference via variational approaches. Our experiments show that this algorithm outperforms latent Dirichlet allocation, the state-of-the-art generative probabilistic model for co-occurrence data in text modeling, document categorization and information retrieval.

Co-occurrence data includes the number of times two entities have occurred together. Many problems studied in the machine learning community involve co-occurrence data. Examples are text data (with bag-of-word assumption): the co-occurrence between a word and a document, image-keyword data: the co-occurrence between image features and keywords, citation data (author level): the co-occurrence between two authors — the number of times one cited another and vice versa, and market basket data: the co-occurrence between items purchased with each other.

Our proposed algorithm, latent logistic allocation (LLA), follows a latent variable model in modeling co-occurrence data. In LLA, the latent variables of both types of entities are assumed to be generated from Gaussian priors and the relationships between entities are computed based on a kernel which is a function of latent variables. Here, we explored two different kernels: a linear (BL) kernel and a radial basis function (RBF) kernel. Afterwards, given the categorical nature of the co-occurrence data, the logistic function is used to translate the continuous value of the relationship from the kernel to a categorical relation.

Although LLA uses a kernel function to capture the relationship between entities, there is a big difference between LLA and kernel-based learning algorithms [82]. In kernel-based learning algorithms such as support vector machines, entities with different labels need to be separable by hyperplanes in the feature space. Since this is usually not the case, mapping entities into a new space where they can be separated by a hyperplane is useful. The new space might be infinite in dimensions which is impossible computationally. To remedy this problem, a kernel trick might be used; instead of mapping entities into a new space, a kernel function over the original feature space is used which is a surrogate for the dot product between entities in the new feature space.

In comparison, LLA is not a discriminative model and separability is not an issue. However, we believe that entities have some intrinsic features that explain the creation of data. In co-occurrence data, the interaction between features of two entities determines the number of times they co-occur. Generally, in co-occurrence data, there is no access to these features and consequently the shape of interaction is not known. Latent variables are used for discovering the features, and the kernel function is the representative of the interaction between entities. For example, using a linear kernel, we are mapping entities into a space where the interaction between them is explained by the dot product of the latent variables of entities. In this sense, we do not use kernel functions as a trick as there is no original feature space. As will be explained in the next section, previous latent variable models such as LDA mostly used a linear kernel.

LLA is a fully generative probabilistic model. It is possible to learn the model via the maximum likelihood approach. However, such an approach will be highly prone to over-fitting the data as the dimensionality of the hidden space is increased. Therefore, we choose to use a Bayesian approach to learn the hidden variables. Given the intractability of computing the integral for Bayesian inference, we choose approximate inference via variational approaches.

Based on the intrinsic characteristics of the co-occurrence data, we propose 4 different groups: hetero-directed, hetero-undirected, homo-directed and homo-undirected. Hetero stands for heterogeneous, the case of different entities and homo stands for homogeneous entities. Directed (undirected) refers to the case that the relation between entities is directed (undirected) in the generative model. Although we explain the generative model for all, we concentrate on the hetero-directed case in presenting the approximate inference algorithm and in the experimental result. The approximate inference presented for the hetero-directed case is directly applicable to other cases. Therefore, we do not claim that our model outperforms other models handling those scenarios — if there are any. The scope of current work is showing that LLA outperforms the state-of-the-art model LDA — which is only capable of handling the hetero-directed case, and providing the possibility to apply LLA to other cases.

3.3.1 Model

Latent logistic allocation (LLA) is a generative probabilistic model for modeling co-occurrence data. To explain different types of co-occurrence data, we use graph notation. Let $G = (U, E)$ represent an unweighted graph with multiple edges; that is, multiple edges are allowed to exist between any two nodes. Each edge depicts a token in the co-occurrence dataset. So if edge or token (i, k) occurred 3 times, then v_{ik} (the i^{th}, k^{th} element of co-occurrence matrix) is 3, and there are 3 edges between nodes i and k in the equivalent graph. The reason we use graph notation is to consider some specific relationships that cannot be represented by the matrix notation. The matrix only represents the number of co-occurrence but not the direction between entities if

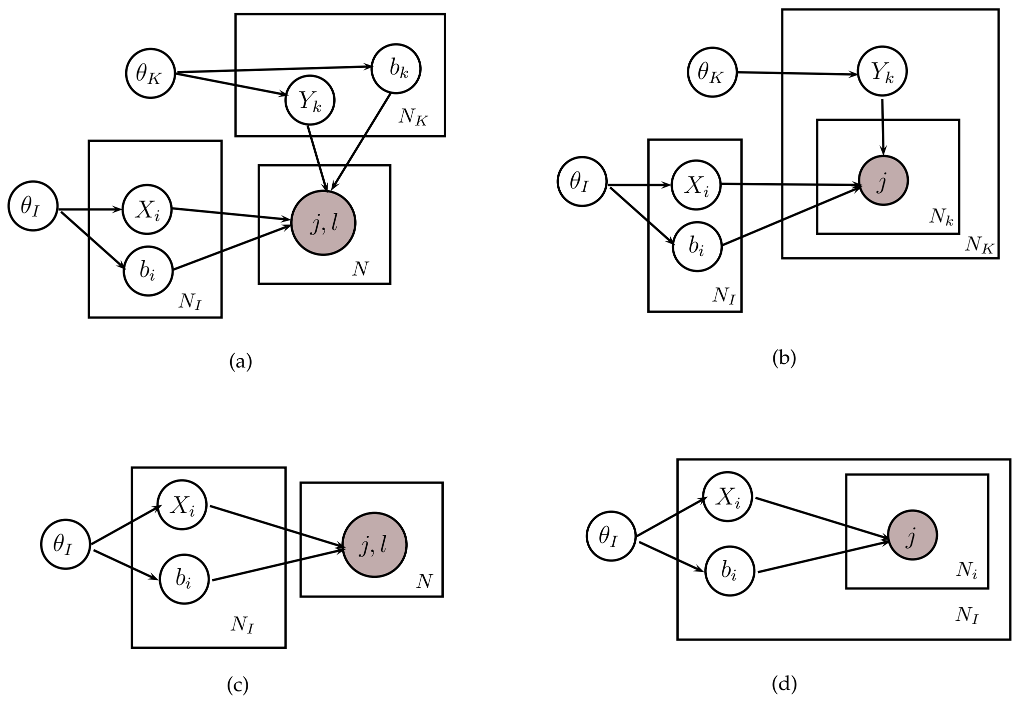


Figure 3.11: The graphical model of LLA, (a) Heterogeneous undirected, (b) Heterogeneous directed, (c) Homogeneous undirected, and (d) Homogeneous directed

co-occurrence is not symmetric. Based on the structure of the graph G, there are two types of structural attributes that describe a co-occurrence data. First is whether the graph is directed or undirected. Second is whether the nodes U are homogeneous or heterogeneous.

When the co-occurrence graph is directed, it means that one type of entity is responsible for generating another type, therefore it is logical to define the generative model as a conditional probability. In the case of an undirected graph, the joint probability is more appropriate. In the case of heterogeneous nodes, nodes in U are divided into two groups, U_1 and U_2, and edges can only be defined between nodes of different types; this heterogeneous graph is a bipartite graph. In the case of homogeneous nodes, there is only one type of node. Here, for the sake of simplicity, we assume that there is no edge from a node to itself — an assumption that can be easily relaxed.

Based on the categories for the co-occurrence data, four different classes can be defined:

- **Hetero-directed**: Each token consists of two different entities and one type of entity is responsible for generating another type. The most popular example of this type is text data where a document is responsible for generating words. Therefore, the link direction is from document nodes to word nodes.
- **Hetero-undirected**: Each token consists of two different entities and both entities are generated simultaneously from a joint distribution. An example of this type is the co-occurrence of image features and keywords. One might argue that image features generate keywords, but in fact, keywords are explanations for images generated by a human, so the undirected assumptions seems more valid.
- **Homo-directed**: Each token consists of two entities from the same type and one entity is responsible for generating the other. An example is co-citation data. Entities are scholars who cite other scholars in their research papers. Each citation is directed — from one scholar to another.
- **Homo-undirected**: Each token consists of two entities from the same type and both entities are generated simultaneously from a joint distribution. An

example is the co-occurrence between items from market basket data. Each token consists of two items that have been purchased together. Here, we cannot say one item is responsible for purchasing another item. All we know is that the shopper chose them both.

We present a separate model for each category. However, it is possible to apply the model for a directed case to data that fits into the undirected case and vice versa. For example, one might assume that the text data is undirected as pLSI treats both words and documents similarly (i.e. the undirected assumption). On the other hand, LDA assumes the same example to be a directed case.

In LLA, the relationship between entities is captured by embedding them in a latent variable space. Following the same notation from previous sections, i indexes the entities of the first type and k indexes the entities of the second type (in the heterogeneous case). Similar to other latent space models, the dimension D of the latent space is an algorithmic input and can be chosen in a Bayesian manner. Here, we treat D as a known parameter. The learned positions of entities in the latent space are denoted by X_i for the entities of the first type and Y_k for the entities of the second type. Furthermore, b_i and b_k represent the biases of entities. By biasness we refer to the situation in which some entities tend to occur more often, such as some words which have higher frequency than others.

In the current model, we assume Gaussian priors on all latent variables. This is an arbitrary choice of distribution and one may assume any other distribution. The relationship between entities is computed via a kernel function. This way the

Gaussian assumption of the latent space is not critical, since the kernel-trick idea can be applied. More precisely, using an appropriate kernel, the latent space can be transfered into other spaces. Now we present LLA for each category.

3.3.1.1 Hetero-undirected

Since edges in the graph are indirected, we need to model the data via the joint probability:

$$P(i,k) = \frac{1}{Z}\exp(\mathcal{K}(X_i, Y_k) + b_i + b_k), \tag{3.15}$$

where $Z = \sum_{ik} \exp(\mathcal{K}(X_i, Y_k) + b_i + b_k)$, and the kernel function $\mathcal{K}(X_i, Y_k)$ converts the values for the two vector variables with D dimensions to a real value scalar.

The graphical model of the generative process for hetero-undirected data is shown in Figure 3.11 (a). First, latent variables are generated for N_I entities of the first type from the prior parameters θ_I and N_K entities of the second type from the prior parameters θ_K. Then for each of the N tokens, a pair is chosen from the $N_I \times N_K$ possible pairs via a multinomial distribution with probabilities from (3.15). The generative process can be summarized as follows:

1. For each entity i:
   (a) Choose entity latent variable $X \sim N(\mu_{0I}, \sigma_{0I}^2 \mathcal{I})$
   (b) Choose entity bias variable $b \sim N(\beta_{0I}, \xi_{0I}^2)$
2. For each entity k:
   (a) Choose entity latent variable $Y \sim N(\mu_{0K}, \sigma_{0K}^2 \mathcal{I})$
   (b) Choose entity bias variable $b \sim N(\beta_{0K}, \xi_{0K}^2)$

3. For each token:

    (a) Choose a pair $(j, l) \sim Multinomial(P(.,.))$

where $\mathcal{I}$ denotes the identity matrix and P is computed using (3.15).

3.3.1.2 Hetero-directed

We assume without loss of generality that in the co-occurrence graph, the direction of edges are from entities indexed by k to entities indexed by i. The probability model for this data is defined as a conditional probability:

$$P(i|k) = \frac{1}{Z_k} \exp(\mathcal{K}(X_i, Y_k) + b_i), \tag{3.16}$$

where $Z_k = \sum_i \exp(\mathcal{K}(X_i, Y_k) + b_i)$. Note that we do not include a bias parameter for entity k, since we are conditioning on k and the bias of k has no effect. Even if b_k is inserted into (3.16), it will be canceled out. To justify such an approach for hetero-directed data, let us consider text data. In text data, the author of each document is generating words independent of words being generated by other authors. Such an assumption has been made in the LDA model.

The graphical model of the generative process for hetero-directed data is shown in Figure 3.11 (b). First, latent variables are generated for the N_I entities of the first type from the prior parameters θ_I and for the N_K entities of the second type from the prior parameters θ_K. Then for each of the N_k tokens in entity k, an entity of the first type is chosen from N_I possible entities via a multinomial distribution with probabilities from (3.16). The generative process can be summarized as follows:

1. For each entity i:

   (a) Choose entity latent variable $X \sim N(\mu_{0I}, \sigma_{0I}^2\mathcal{I})$

   (b) Choose entity bias variable $b \sim N(\beta_{0I}, \xi_{0I}^2)$

2. For each entity k:

   (a) Choose entity latent variable $Y \sim N(\mu_{0K}, \sigma_{0K}^2\mathcal{I})$

   (b) For each token:

      i. Choose $j \sim Multinomial(P(.|k))$

where P is computed from (3.16).

3.3.1.3 Homo-undirected

Since edges are indirected, we need to model the data via the joint probability:

$$P(i, j) = \frac{1}{Z} \exp(\mathcal{K}(X_i, X_j) + b_i + b_j), \tag{3.17}$$

where $Z = \sum_{i<j} \exp(\mathcal{K}(X_i, X_j) + b_i + b_j)$, and b_i and b_j are scalars.

The graphical model of the generative process for homo-undirected data is shown in Figure 3.11 (c). First, latent variables are generated for N_I entities from the prior parameters θ_I. Then, for each of the N tokens, a pair is chosen from $N_I(N_I - 1)/2$ possible pairs via a multinomial distribution with probabilities from (3.17). The generative process can be summarized as follows:

1. For each entity i:

   (a) Choose entity latent variable $X \sim N(\mu_{0I}, \sigma_{0I}^2\mathcal{I})$

   (b) Choose entity bias variable $b \sim N(\beta_{0I}, \xi_{0I}^2)$

2. For each token:

(a) Choose a pair $(j, l) \sim Multinomial(P(.,.))$

where P is computed from (3.17).

3.3.1.4 Homo-directed

The model for this data is defined as a conditional probability:

$$P(j|i) = \frac{1}{Z_i} \exp(\mathcal{K}(X_i, X_j) + b_j), \tag{3.18}$$

where $Z_i = \sum_{j \neq i} \exp(\mathcal{K}(X_i, X_j) + b_j)$. The graphical model of the generative process for homo-directed data is shown in Figure 3.11 (d). First, latent variables are generated for the N_I entities θ_I. Then for each of the N_i tokens in entity i, an entity is chosen from $(N_I - 1)$ possible entities via a multinomial distribution with probabilities from (3.18). The generative process can be summarized as follows:

1. For each entity i:
   - (a) Choose entity latent variable $X \sim N(\mu_{0I}, \sigma_{0I}^2 \mathcal{I})$
   - (b) Choose entity bias variable $b \sim N(\beta_{0I}, \xi_{0I}^2)$
   - (c) For each token:
     - i. Choose $j \sim Multinomial(P(.|i))$

where P is computed from (3.18).

3.3.1.5 Approximate inference

In this section, we only provide an inference algorithm for the hetero-directed case. Deriving the inference algorithm for other scenarios will be straightforward based on the method proposed in this section. We study two kernel functions: linear and radial basis function (RBF). If one is interested in implementing LLA via the maximum likelihood approach, then using other kernels will be easy as long as the kernel function is differentiable. However, the Bayesian analysis presented in this section will be specific for the above-mentioned kernels.

The likelihood of the whole dataset given the assumption that the probability of each token is independent from other tokens given the hidden variables is as follows:

$$P(U|X,Y,b) = \prod_u P(i_u|k_u, X, Y, b) = \prod_{ik} P(i|k, X, Y, b)^{v_{ik}}, \tag{3.19}$$

where U is the set of all tokens (i.e. the whole dataset), and v_{ik} is the number of times the token (i, k) has occurred. Given (3.16) and (3.19), the log-likelihood is as follows:

$$\log P(U|X,Y,b) = \sum_{ik} v_{ik}\mathcal{K}(X_i, Y_k) + \sum_i v_{i.} b_i - \sum_k v_{.k} \log \sum_i \exp(\mathcal{K}(X_i, Y_k) + b_i), \tag{3.20}$$

where $v_{i.} = \sum_k v_{ik}$ and $v_{.k} = \sum_i v_{ik}$. To estimate the hidden variables X, Y and b, we can maximize the log-likelihood (3.20). However, the maximum likelihood approach has problems such as overfitting. A Bayesian approach will result in a more robust solution by giving the posterior distribution of hidden parameters. Given the Bayes rule, the posterior distribution of latent variables given the data is as follows:

$$P(X,Y,b|U) = \frac{P(U|X,Y,b)P(X)P(Y)P(b)}{\int P(U|X',Y',b')P(X')P(Y')P(b')dX'dY'db'}, \tag{3.21}$$

which is not computable analytically due to the intractability of the integral in the denominator. Therefore, we chose to use variation inference [56] which is a popular algorithm for approximate inference in graphical models.

In variational approximation, instead of finding the true posterior, we estimate a variational distribution for each latent variable. The main idea is minimizing the difference between the true posterior and the surrogate variational distribution so then we can use the variational distribution for making inference about latent variables. If $Q(X, Y, b)$ shows the variational distribution over latent variables, we are interested in minimizing the Kullback-Leibler divergence between the true posterior and its approximation: $KL[Q(X, Y, b)||P(X, Y, b|U)]$. This is equivalent to the following problem [8]:

$$\log P(U) \geq E_Q[\log P(U|X, Y, b)] - KL[Q(X, Y, b)||P(X, Y, b)], \tag{3.22}$$

where $E_Q[.]$ is expectation with regard to variational distributions. Here we use the naive mean field algorithm [103] and assume the variational distributions are independent Gaussians with the following parameters: $X_i \sim N(\mu_i, \sigma_i^2\mathcal{I})$, $Y_k \sim N(\mu_k, \sigma_k^2\mathcal{I})$, and $b_i \sim N(\beta_i, \xi_i^2)$.

Note that it is possible to use a covariance matrix instead of $\sigma^2\mathcal{I}$ and the only reason we chose to use independent coordinates is simplicity in optimization. Substituting $P(U|X, Y, b)$ with its value from (3.20), the lower bound on the probability of data in 3.22 can be written as:

$$\begin{aligned}\mathcal{L}(\mu, \beta, \sigma^2, \xi^2) = \sum_{ik} v_{ik} E_Q[\mathcal{K}(X_i, Y_k) + b_i]& \\ - \sum_k v_{.k} E_Q[\log \sum_i \exp(\mathcal{K}(X_i, Y_k) + b_i)]& \\ - KL(Q(X)||P(X)) - KL(Q(Y)||P(Y)) - KL(Q(b)||P(b)).&\end{aligned} \tag{3.23}$$

Given the Gaussian distribution for priors and variational distributions, all integrals for computing the expectations in equation 3.23 are analytically solvable for many kernel functions except for the part $\sum_k v_{.k} E_Q[\log \sum_i \exp(\mathcal{K}(X_i, Y_k) + b_i)]$ which is intractable because of the log-sum-exp format. It is possible to use the concavity of the log function to define an upper bound on the log-sum-exp function:

$$\log a \leq \phi a - \log \phi - 1, \tag{3.24}$$

where the equality holds iff $\phi = 1/a$. Such an approach has been used in several works in the variational inference context [14, 9, 10]. Therefore, the new bound is as follows:

$$\begin{aligned}\mathcal{L}(\mu, \beta, \sigma^2, \xi^2) = const. + \sum_{ik} v_{ik} E_Q[\mathcal{K}(X_i, Y_k) + b_i] \\ - \sum_k v_{.k} \phi_k \sum_i E_Q[\exp(\mathcal{K}(X_i, Y_k) + b_i)] \\ - KL(Q(X)||P(X)) - KL(Q(Y)||P(Y)) - KL(Q(b)||P(b)),\end{aligned} \tag{3.25}$$

where the constant does not depend on decision variables and we set $\phi_k = [\sum_i E_Q(exp(\mathcal{K}(X_i, Y_k) + b_i))]^{-1}$ to tighten the lower bound with regard to the log-sum-exp part. Now, we provide the specific lower bounds for linear and RBF kernels.

Linear kernel

Here, we provide the variational lower bound for the linear kernel: $\mathcal{K}(X_i, Y_k) = X_i Y_k^T$. The only tricky part in (3.25) is deriving the integral $E_Q[\exp(X_i Y_k^T + b_i)]$. Let $x_1 \sim N(\mu_1, \sigma_1)$ and $x_2 \sim N(\mu_2, \sigma_2)$, then it can be shown that the following equation holds:

$$E[exp(x_1 x_2)] = \frac{\exp(\frac{\sigma_2^2 \mu_1^2 + \sigma_1^2 \mu_2^2 + 2\mu_1 \mu_2}{2(1 - \sigma_1^2 \sigma_2^2)})}{\sqrt{1 - \sigma_1^2 \sigma_2^2}} \tag{3.26}$$

given the condition that $\sigma_1^2\sigma_2^2 < 1$, otherwise the integral will be infinite. This happens because of our estimation via the upper bound given in (3.24) and the integral on the log-sum-exp function is bounded. Also, given the moment generation function for Gaussian distribution we know $E_Q(\exp(b_i)) = \exp(\beta_i + \xi_i^2/2)$. Based on (3.26), we have:

$$E_Q[\exp(X_iY_k^T + b_i)] = \eta_{ik}^D \exp(\eta_{ik}^2\zeta_{ik} + \beta_i + \xi_i^2/2) \tag{3.27}$$

where $\zeta_{ik} = (\sigma_i^2\mu_k\mu_k^T + \sigma_k^2\mu_i\mu_i^T + 2\mu_i\mu_k^T)/2$ and $\eta_{ik} = (1 - \sigma_i^2\sigma_k^2)^{-1/2}$. As a result, the lower bound in (3.25) can be written as:

$$\begin{aligned}\mathcal{L}(\mu, \beta, \sigma^2, \xi^2) = const. &+ \sum_{ik} v_{ik}\mu_i\mu_k^T + \sum_i v_{i.}\beta_i \\ &- \sum_k v_{.k}\phi_k \sum_i \eta_{ik}^D \exp(\eta_{ik}^2\zeta_{ik} + \beta_i + \xi_i^2/2) \\ &- \frac{1}{2}\sum_i[-D\log\sigma_i^2 + D\frac{\sigma_i^2}{\sigma_{0I}^2} + \frac{(\mu_i - \mu_{0I})(\mu_i - \mu_{0I})^T}{\sigma_{0I}^2}] \\ &- \frac{1}{2}\sum_k[-D\log\sigma_k^2 + D\frac{\sigma_k^2}{\sigma_{0K}^2} + \frac{(\mu_k - \mu_{0K})(\mu_k - \mu_{0K})^T}{\sigma_{0K}^2}] \\ &- \frac{1}{2}\sum_i[-\log\xi_i^2 + \frac{\xi_i^2}{\xi_0^2} + \frac{(\beta_i - \beta_0)^2}{\xi_0^2}].\end{aligned} \tag{3.28}$$

To find variational values, we need to optimize (3.28). Since $\sigma_i^2\sigma_k^2 < 1$, to have an unconstrained optimization problem, we use an auxiliary variable χ and the logistic function in our experiments: $\sigma^2 = \frac{1}{1+\exp(-\chi)}$. Any unconstrained optimization algorithm can be used to solve (3.28). In our experiments, we used gradient ascent due to its simplicity.

RBF

Here, we provide the variational lower bound for the radial basis function (RBF) kernel: $\mathcal{K}(X_i, Y_k) = -\delta^2(X_i, Y_k) = -(X_i - Y_k)(X_i - Y_k)^T$. Note the difference between the shape of RBF in kernel-based learning approaches and here as we only use the negative squared Euclidean distance part of RBF. The only tricky part in (3.25) is deriving the integral $E_Q[\exp(-\delta^2(X_i, Y_k) + b_i)]$. Let $x_1 \sim N(\mu_1, \sigma_1)$ and $x_2 \sim N(\mu_2, \sigma_2)$, then it can be shown that the following equation holds:

$$E[exp(-(x_1 - x_2)^2)] = \frac{exp(-\frac{(\mu_1-\mu_2)^2}{1+2(\sigma_1^2+\sigma_2^2)})}{\sqrt{1 + 2(\sigma_1^2 + \sigma_2^2))}}. \tag{3.29}$$

Therefore we have:

$$E_Q[\exp(-\delta^2(X_i, Y_d) + b_k)] = \eta_{ik}^D \exp(-\eta_{ik}^2 \delta^2(\mu_i, \mu_k) + \beta_i + \xi_i^2/2) \tag{3.30}$$

where $\eta_{ik} = [1 + 2(\sigma_i^2 + \sigma_k^2))]^{-1/2}$. As a result, the lower bound in equation 3.25 can be written as follows:

$$\begin{aligned}\mathcal{L}(\mu, \beta, \sigma^2, \xi^2) = &\ constant \\ & - \sum_{ik} v_{ik} \delta^2(\mu_i, \mu_k) - (\sum_i v_{i.} \sigma_i^2 + \sum_k v_{.k} \sigma_k^2) D + \sum_i v_{i.} \beta_i \\ & - \sum_k v_{.k} \phi_k \sum_i \eta_{ik}^D \exp(-\eta_{ik}^2 \delta^2(\mu_i, \mu_k) + \beta_i + \xi_i^2/2) \\ & - \frac{1}{2} \sum_i [-D \log \sigma_i^2 + D \frac{\sigma_i^2}{\sigma_{0I}^2} + \frac{(\mu_i - \mu_{0I})(\mu_i - \mu_{0I})^T}{\sigma_{0I}^2}] \\ & - \frac{1}{2} \sum_k [-D \log \sigma_k^2 + D \frac{\sigma_k^2}{\sigma_{0K}^2} + \frac{(\mu_k - \mu_{0K})(\mu_k - \mu_{0K})^T}{\sigma_{0K}^2}] \\ & - \frac{1}{2} \sum_i [-\log \xi_i^2 + \frac{\xi_i^2}{\xi_0^2} + \frac{(\beta_i - \beta_0)^2}{\xi_0^2}]. \end{aligned} \tag{3.31}$$

To find variational values, we need to optimize (3.31). Since $\sigma^2 \geq 0$, to have an unconstrained optimization problem, we use an auxiliary variable χ and the exponential function in our experiments: $\sigma^2 = \exp(\chi)$. Any unconstrained optimization algorithm can be used to solve (3.31). In our experiments, we used gradient ascent with multiple random starts.

3.3.2 Experimental results

In our experiments, we evaluated LLA in three application areas: text modeling, document categorization, and information retrieval. In all three tasks, we treat text data as hetero-directed data where the direction is from documents to words. We set the prior mean to 0 for both words and documents, and prior variance to 2 for words and 10 for documents. No fine tunning was done for finding priors. We chose smaller prior variance for words due to the fact that the support for words is often lower compared to the support for documents. There are some words that occur only 2 or 3 times while the number of occurrence of documents (the number of words in them) is usually much higher. Throughout this section, we represent LLA with linear kernel as LLA-L and LLA with RBF kernel as LLA-RBF.

We use LDA for the sake of comparison, as both LLA and LDA are Baysian latent variable models that can be applied to text data and to the best of our knowledge, there is no other model with this characteristic. Furthermore, Blei et al. [11] showed that LDA outperforms some other comparable algorithms such as pLSI and mixture of unigrams. For implementing LDA, the Gibbs sampling algorithm and the settings in [44] were used. The number of topics for LDA and the dimensions for LLA were set to 20, 50, 100, and 200 in all experiments.

For the first two tasks (text modeling and document categorization), we used

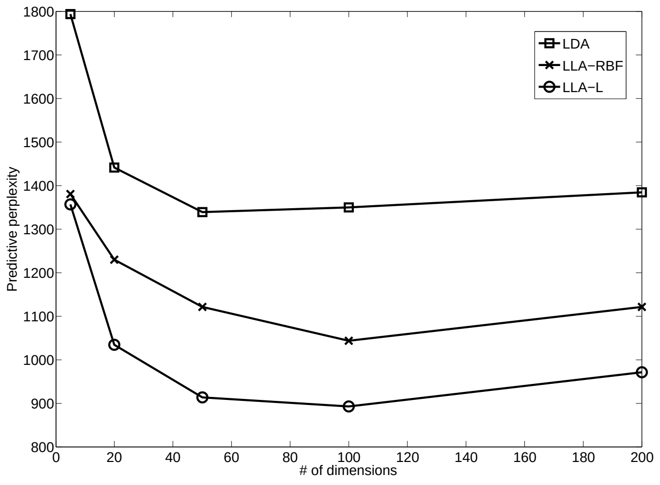


Figure 3.12: The predictive perplexity for TDT-2 dataset

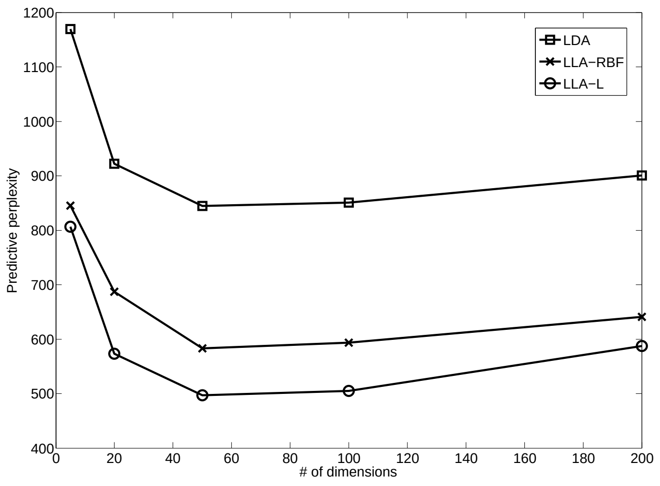


Figure 3.13: The predictive perplexity for Reuters21578 dataset

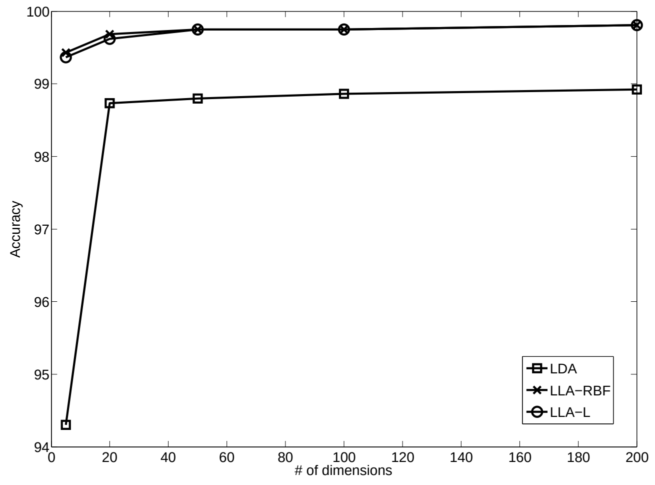


Figure 3.14: The test accuracy for TDT-2 dataset

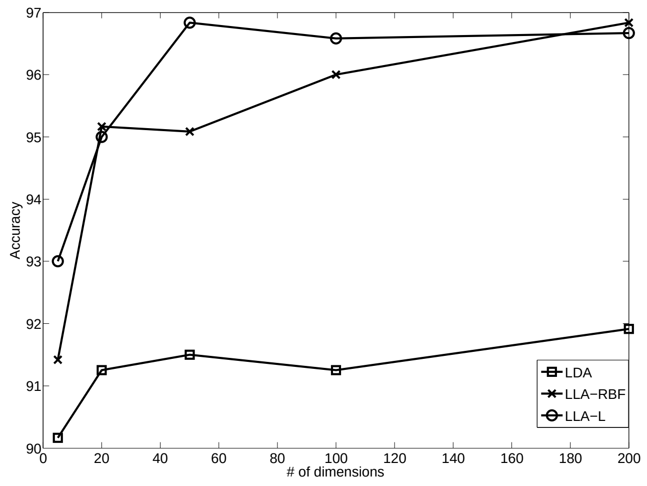


Figure 3.15: The test accuracy for Reuters21578 dataset

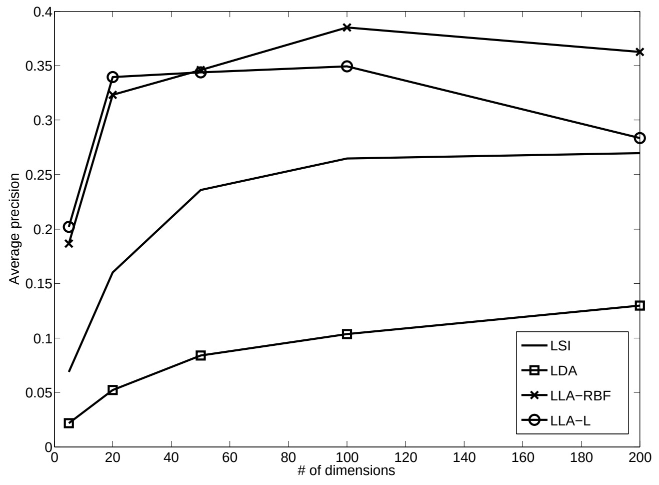


Figure 3.16: Average precision for queries of CRAN dataset

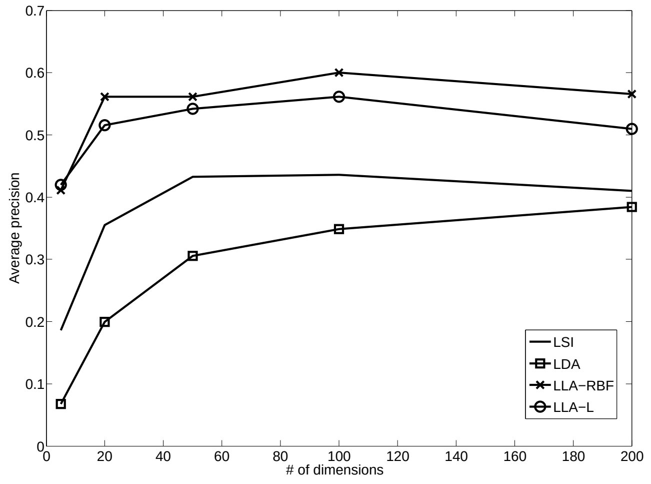


Figure 3.17: Average precision for queries of MED dataset

a subset of the TDT-2 and Reuters21578 datasets[3]. In each dataset, we selected 5 categories so that the number of documents in categories is almost equal. Words that occurred fewer than 3 times were excluded. Our subset of TDT-2 includes 8,676 words and 1,584 documents and our subset of Reuters21578 includes 4,711 words and 1203 documents. We split these data into training and testing sets (80%-20%) category-wise — the test set includes around 20% of each category. For both models, first the training set was used to train the parameters for words and documents, then the test dataset was used for training only test documents — we use the space of words from the training phase to map the new documents in the latent space.

Figures 3.12 and 3.13 represent the results of text modeling for TDT-2 and Reuters21578, respectively. We used the metric perplexity [11] on the test subset for comparing results. Perplexity simply measures the probability a model gives to each token and a lower perplexity implies a stronger model. The definition of predictive perplexity is

$$preplexity(\mathcal{D}_{test}) = \exp\left(-\frac{\sum_{ku}\log P_{model}(w_{ku})}{\sum_k N_k}\right),$$

where $\mathcal{D}_{test}$ is the test data set and w_{ku} is the word for token u and document k. Both LLA-L and LLA-RBF outperform LDA in text modeling. LLA-L outperforms LLA-RBF; however, in Reuters21578 dataset the difference is not sizable.

For the document categorization task, both LDA and LLA can be treated as dimensionality reduction algorithms. For classifying documents, we trained a support vector machine (SVM) classifier on the training subset. We used the LIBSVM

[3]http://www.zjucadcg.cn/dengcai/Data/TextData.html

software package [22] in these experiments. The linear kernel was used for SVM in all experiments. Note that the test subset were not included in either the model training (Bayesian inference) nor classification training (SVM). We use the metric accuracy for comparison. Accuracy is defined as the percentage of points are labeled correctly.

Figures 3.14 and 3.15 represent the results of document categorization for TDT-2 and Reuters21578, respectively. Again, LLA outperforms LDA in both datasets. The performance of LLA-L and LLA-RBF is very close in this task.

For information retrieval task, we used datasets CRAN and MED[4]. The CRAN dataset includes 3,763 words, 1398 documents and 225 queries, and MED includes 7014 words, 1033 documents, and 30 queries. In this task, we included the result of latent semantic indexing (LSI) as a successful information retrieval algorithm. In both LDA and LLA, queries were treated as new documents and mapped into the latent space. Then, the cosine similarity between queries and documents was used to compute a score for retrieval. The metric average precision is used to compare algorithms. Average precision is the average over the precision of all relevant documents given the ranked list which conveys information regarding both precision and recall. For this experiment, we avoid using regular weighting procedures such as tf-idf since it is not possible to incorporate in LDA.

Figures 3.16 and 3.17 represent the results of information retrieval for CRAN and MED, respectively. LLA outperforms other algorithms and the performance of LLA-L and LLA-RBF is again similar.

[4]http://web.eecs.utk.edu/research/lsi/

3.3.3 Query-based visualization

In [63], we proposed an approach for visualizing retrieved list of documents for a query based on LLA with RBF kernel. Visualization is one of the most important exploratory tools for data analysis and mining. Using Euclidean distance, radial basis function, embedded entities can be used for visualization. The proposed Bayesian approach enables accurate embedding in high-dimensional space which is not useful for visualization. Therefore, we propose a method to embed a filtered number of entities for a queryquery-based visualization. Our experiments show that our proposed models outperform co-occurrence data embedding, the state-of-the-art model for visualizing co-occurrence data.

It is hard to capture the essence of real-world data in two dimensions due to high complexity. On the other hand, visualizing a large number of data points is confusing rather than informative. Therefore, a way to present a filtered version of data is useful. To this end, we propose a query-based visualization method. Although we present this algorithm in the context of information retrieval, it can be applied to any query-answering problem for co-occurrence data.

Visualizing co-occurrence data with heterogeneous nodes—generally heterogeneous data—has been rarely studied. Most of the literature concentrates on embedding only one type of data (e.g. multi-dimensional scaling [24]). In the context of text data—which is co-occurrence data with heterogeneous nodes—most visualization approaches usually embed only documents or words via embedding algorithms such as multidimensional scaling [108]. The state-of-the-art algorithm to visualize co-occurrence data with heterogeneous data is co-occurrence data embedding (CODE) [42].

The main intuition behind CODE is that if two points are related then they should be very close in the latent space. Therefore, the result of embedding can be presented as a visualization of entities. Although the embedding is based on the relationship between entities from different types, we expect the entities from the same type to be close due to the transitivity of distance.

Although LLA-RBF outperforms CODE even in a 2-dimensional space, the difference intensifies in higher dimensions. Nevertheless, it is not possible to interpret more than 3 dimensions visually. Our other contribution is proposing a query-based visualization to embed a filtered number of entities.

Here, we present query-based visualization for information retrieval; however, it can be applied to other dyadic data—see section 3.1 for a similar approach in collaborative filtering.

In query-based visualization (QBV), documents, query words, and the query are embedded in a Euclidean space to help the user in identifying documents of interest. Unfortunately, 2 dimensions is barely enough to capture the complexity of a data, while higher dimensions cannot be interpreted visually. Additionally, representing all data to a user is beyond a person's processing ability. Therefore, visualizing only top-N (N can be specified by user) documents is of interest. These top-N documents can be chosen by an arbitrary retrieval method.

Therefore, a two-phase visualization can be used. First, the data will be embedded in a high-dimensional space and then a group of entities can be chosen via some filtering approach to be re-embedded in a 2-dimensional space by classic algorithms such as multidimensional scaling (MDS) [24]. The second embedding phase is straightforward, since we already have the distances from the first phase which satisfy all requirements for the Euclidean distance and MDS can be applied directly. In an information retrieval context, our proposed approach is embedding words, documents, and the query in a high-dimensional space, and then using the distances in that space, embedding selected objects in a 2-dimensional space via multidimensional scaling.

Another approach may be embedding the filtered data directly into a 2-dimensional space. However, such an approach is not desirable for two reasons. First, since there are few entities in the filtered data, generalization is expected to be poor which deteriorates the result. Second, embedding entities separately for each query is very time consuming and inefficient, especially given the high number of queries in the retrieval systems.

3.3.4 Experimental results

In our experiments, we compare CODE and LLA-RBF in the context of information retrieval. Text data is considered as hetero-directed and so we use a conditional probability model.

We study the quality of the embedding using two evaluation metrics. First is the proximity of an embedded query to the relevant documents. This can be measured with average precision (AP). The definition of AP that we used is averaging the precision of all relevant documents at the point they are retrieved. Let $AP(S, R)$ be the average precision where R is the set of relevant documents and S is the score of a method for ranking all documents for retrieval. Then given an embedded query q, the average precision is $AP(-\delta^2_{q.}, R)$ where $\delta^2_{q.}$ shows the distance of the query to all documents. Second, the proximity of relevant documents is of interest. It is important whether all relevant documents are close to each other compared to other documents. To measure this, average relevant documents proximity (ARDP) is proposed:

$$ARDP = [\sum_c \frac{\sum_{k \in R_c} AP(-\delta_{k.}, R_c \backslash \{k\})}{|R_c| - 1}]/N_C, \tag{3.32}$$

where c indexes categories or queries (any group of relevant documents), k indexes documents, R_c is the set of relevant documents in c, and N_C is the total number of categories or queries. ARDP is partially similar to doc-doc measure used in [42]. More precisely, we consider each relevant document as an embedded query and then we compute the AP of retrieving other relevant documents. This metric measures how well we embed data, since relevant documents are expected to be closer and the visualization is better as a result.

In LLA-RBF, we set the prior mean to 0 for both words and documents, and prior variance to 1 for words and 2 for documents. No fine tunning was done for finding priors. We chose smaller prior variance for words due to the fact that the support for words is often lower compared to the support for documents. There are some words that occur only 2 or 3 times while the number of occurrences of documents (the number of words in them) is usually much higher. We used the same four datasets as the previous section.

Table 3.5 presents the result of ARDP for all datasets. We implemented CODE and LLA-RBF in a 2-dimensional space and then computed the ARDP score. LLA-RBF outperforms CODE in all 4 datasets (in CRAN the performance is close).

	TDT2	Reuters21579	CRAN	MEDLINE
CODE	.9023	.4582	.0637	.1561
LLA-RBF	**.9490**	**.5568**	**.0646**	**.1797**

Table 3.5: ARDP on a 2-dimensional space (Best results are in bold)

	CRAN dataset		MEDLINE dataset	
	AP	ARDP	AP	ARDP
CODE-2	.1591	.1299	.4046	.4648
LLA-RBF-QBV	**.3231**	**.3498**	**.6145**	**.6384**

Table 3.6: query-based visualization + LLA-RBF versus CODE with 2 dimensions (best results are bold)

Finally, we explored query-based visualization using LLA-RBF. First, we selected top-100 documents for each query using latent semantic indexing [30] which is a successful method in information retrieval. Note that it is possible to use LLA-RBF for filtering documents directly but here we need a method for both CODE and LLA-RBF for the sake of comparison. Then, we re-embedded all filtered documents, query words, and the query in a 2-dimensional space via MDS using distances obtained from implementing LLA-RBF in a 100-dimensional space. We compare the result to CODE's result in a 2-dimensional space. Table 3.6 represents the result. The performance of query-based visualization is dramatically better than CODE.

Figures 3.18 and 3.19 represent a typical snapshot of visualizing a query using CODE and query-based visualization using LLA-RBF respectively, for a specific query in the MEDLINE dataset. Note the distinction between relevant and irrelevant documents in query-based visualization while they are highly mixed in CODE. Additionally, the query is far away from other entities in CODE which makes interpretation difficult. Query words might help user to identify which area in the space is more relevant.

3.3.5 Conclusion

In this section, we propose a new latent model for co-occurrence data, latent logistic allocation (LLA). Experimental results show that this model outperforms the state-of-the-art latent model, latent Dirichlet allocation in 3 tasks: text modeling, document categorization, and information retrieval. LDA is a widely used and cited approach in the machine learning and text mining communities. We believe the superiority of LLA over LDA in our experiments is mainly because of the flexibility of the latent space in LLA. The latent space of LDA is restricted to non-negative normalized variables (summed to 1) and that is the reason Dirichlet priors are used. On the other hand, there is no restriction on the latent space of LLA and any prior

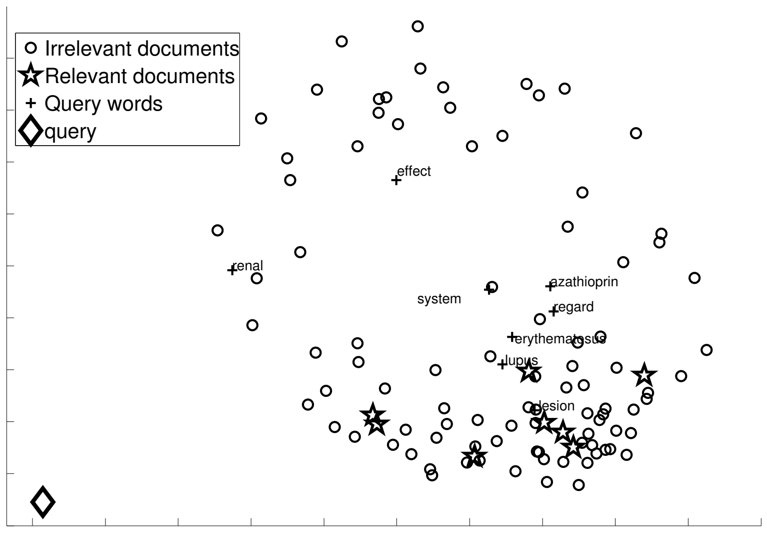


Figure 3.18: Visualization of a typical query from MEDLINE dataset using CODE with 2 dimensions

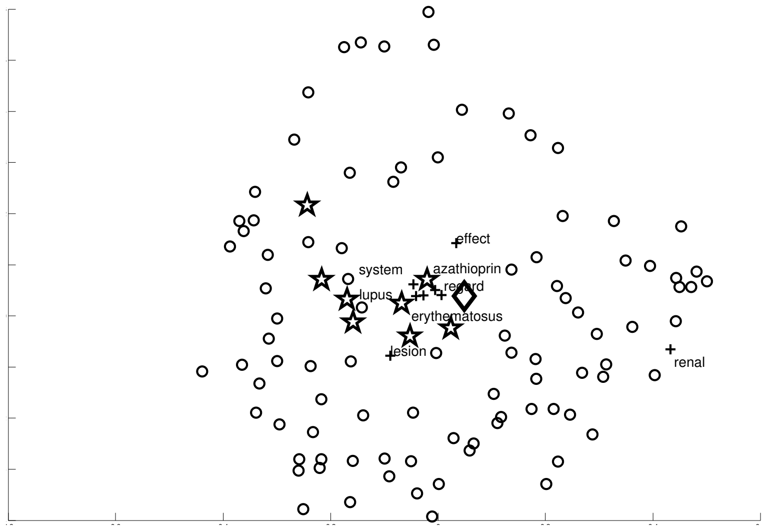


Figure 3.19: Visualization of a typical query from MEDLINE dataset using query-based visualization + LLA-RBF with 100 dimensions

can be used. Furthermore, LDA can use only dot product to estimate the probability of a word given a document while LLA gives the possibility to use other kernels. LLA can incorporate biases of words in modeling which is not possible for LDA. Finally, LLA can handle other scenarios rather than hetero-directed the only scenario LDA can handle. All these factors allow LLA to have more flexibility learning power.

Additionally, we proposed a method to embed filtered data from a high-dimensional embedding for a queryquery-based visualization which was successful in our experiments.

3.4 Analyzing the language evolution

In [57], we proposed an extension to latent Dirichlet allocation (LDA) [11] to analyze the language evolution in a preliminary science classroom. More accurately, we introduce a topic model to analyze the temporal change in the spoken language of a science classroom based on a dataset of conversations among a teacher and students. One of the key goals is discovering the root of the change in the language usage of students. To accomplish this, we defined 4 categories which generate words: 1) background (general) 2) activity, 3) session subject, and 4) personal. Our experimental results support the hypothesis that the change in the language of students mainly consists of using more activity-based language which can be interpreted as using more scientific discourse.

In traditional off-line classrooms, evaluation is mainly based on exams. Other parameters such as attendance or participation are considered but not as much as exam grades. Considering the fact that learning influences language, we believe that the change in the spoken language of students is a representative of progress. Although a teacher has day-to-day interactions with students, it might be hard to detect the change in spoken discourse of students. Therefore, an automatic tool which analyzes temporal aspects of students' spoken language may yield interesting patterns as feedback to different teaching methodologies. Also, such a tool, besides offering additional educational evaluations, may be used to detect students who are not making progress.

In this work, we use a dataset of dialogues in a science classroom to investigate the changes over time in the language of students. As expected, the language of students becomes more similar to the language of instructor over time. There are two explanations for this change: 1) the personal language usage of students is becoming more similar to the personal language usage of teacher 2) students are learning to use scientific discourse and since the teacher is using scientific discourse as well, their languages are becoming more similar. To identify the main source of change, we designed and implemented a topic model which distinguishes the sources of words. We assume each word may be generated from one of four different sources: 1) background, 2) activity, 3) session, and 4) personal. Background words are general words such as "I" or "is". Activity words are the words that are related to the scientific discourse. For example if a student is reasoning about a scientific claim, he uses activity-based words. The following conversations are from dialogues in the classroom during a claim and evidence activity:

- Student A: our claim is the more mass you have, the more force you get out of the object
- Teacher: evidence?
- Student B: the more washers you put in the cup, the faster the erasers are going to fall

Obviously, words such as "claim" or "evidence" are related to this activity. Session words are related to the subject of the classroom, in the above examples, words such as "mass" or "force" are related to the session subject. Finally, the personal words are related to the specific way to convey concepts such as "actually".

Using the topic model, we show that the change in the language of students is mainly because of an increase in the activity language usage. This means that students are learning to use more terms related to scientific discourse which is a sign of learning. No significant change was observed in personal, session, and background language usage. Although it might seem that growth in session language is a sign of learning, we note that students use session language comparatively more when they are passive and mostly responsive. More precisely, when students comparatively use a lot of words related to the subject of the session, they are mostly responding to the teacher's questions with quick answers.

Our main goal is extracting useful patterns and knowledge out of classroom conversations. We focus on temporal changes in conversation to investigate the changes in students' language. This may offer key insights into students' learning and also potential feedback on different teaching methods.

Number	**Activity**	**# utterance**	**# session**
1	categorization	2834	5
2	claim and evidence	1861	8
3	discussing initial ideas before doing experiment	437	4
4	doing experiment	914	4
5	reflection	4	1
6	what experts say	1522	6

Table 3.7: The list of activities in the classroom

Our dataset consists of a snapshot of conversations in a science classroom. Figure 3.20 shows the structure of the dataset. The parties in the conversation consist of a teacher and 14 students. There are 17 sessions in temporal order over a one-year period and there are a number of utterances in each session (total of 7572 utterances). Each utterance starts when a new person starts talking until the next person starts talking (there were 3 utterances in the example in the previous section). There are 2010 unique words in this dataset. The data was transcribed by human from the videos of the classroom. Note that traditional text analysis preprocessing such as stop-wording or stemming were not performed since some patterns such as the tense of the verb, negative statements, or “wh” words usage are important for us.

Each utterance in the classroom has been assigned manually to an activity. These activities have been identified by human judgment. Table 3.7 shows the title of each activity, the number of sessions they appear in and the number of utterances they include. Activity 5 is infrequent and only happens in one session but other activities occur in 4 or more than 4 sessions.

The key question to be addressed is whether the language used by students

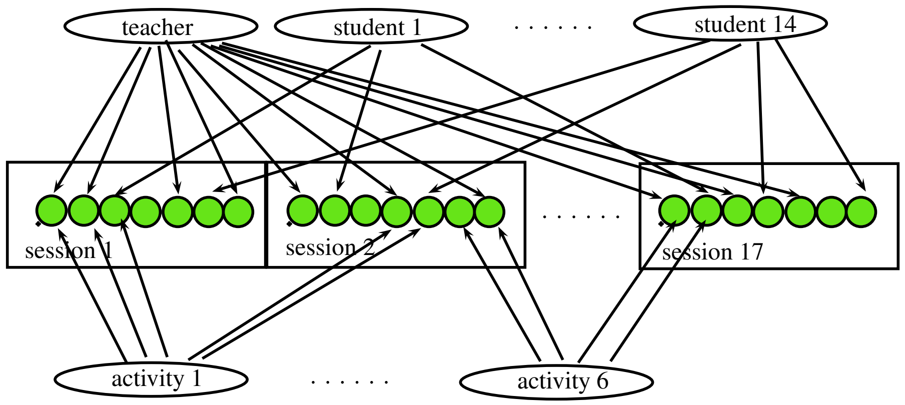


Figure 3.20: The structure of the dataset of the classroom conversations

is changing over time and if yes, how it is changing. Figure 3.21 shows the cosine similarity between the normalized word vectors of teacher and students (all students treated as one person) over different sessions. Although the linear growth of this similarity is not statistically significant at 0.05, a logarithmic growth in similarity is observed.

Table 3.8 shows the test of linear relationship in the cosine similarity between the normalized word vectors of the teacher and all individual students over time. The word vector of a person is a vector with length of the number of words (here 2010) and each value of the vector depicts the frequency of a word. The language of students (all but one) is becoming more similar to teacher's language but none of them are statistically significant.

Table 3.9 represents the words for which the usage has significantly changed (at 0.05 significance). We only considered words that occurred at least in 4 sessions for the students and 8 sessions for the teacher. There are some names such as "tori" or

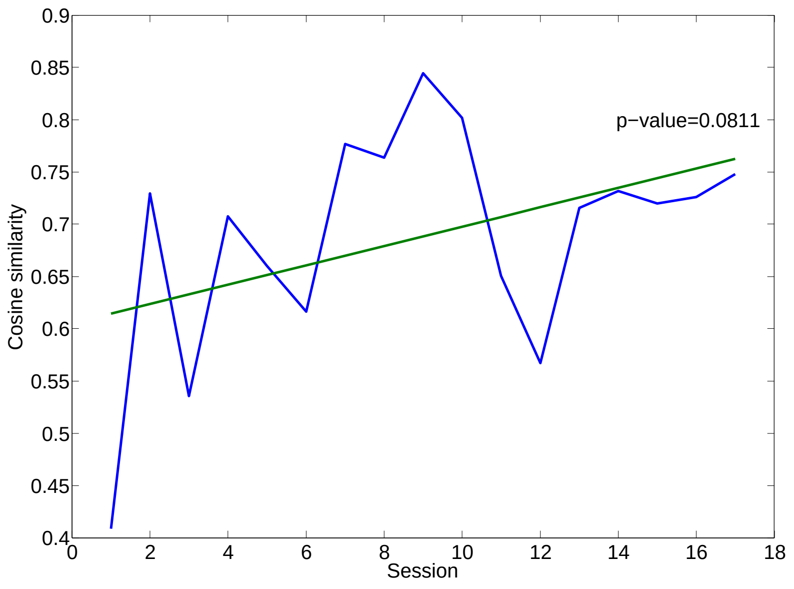


Figure 3.21: The cosine similarity between the normalized word vectors of the teacher and students (all students treated as one person) over time

	St 1	St 2	St 3	St 4	St 5	St 6	St 7	St 8	St 9	St 10	St 11	St 12	St 13	St 14
slope	.0104	.0129	.0202	.0118	.0163	.0120	.0054	.0088	-.0008	.0077	.0014	.0111	.0128	.0124
p-value	.3561	.0611	.0539	.3846	.1506	.2200	.5439	.4245	.9387	.5057	.8388	.4343	.3219	.2412

Table 3.8: Test of linear relationship over time in the cosine similarity between the normalized word vector of the teacher and individual students

"tanner" in this table. Students' usage of comparison words such as "like" or "kind" is increasing over time. On the other hand, the teacher usage of some key words such as "claim", "evidence", or "why" is less frequent in later sessions. It might be due to the fact that students are able to use these types of discourses more effectively so the teacher needs to mention them less.

	Increase in usage	Decrease in usage
Student 1	like, at, blake, need	was
Student 2	no, has, need	and, might
Student 4	you, like, can, them	in
Student 5	go, for	
Student 6	it, like, you, down, first, probably, which, at, called	were, would
Student 7	to	and, might
Student 8	no, kind	or
Student 9	it	
Student 11	in, would	because
Student 12	down, just	is, no
Student 13	you, ok, would, them, see	was
Student 14	says, are, make, sure, ok, pretty, different	too
Teacher	we, would, or, find, need, our, thinking, tori, little, move, her, use, well, listen, over	your, evidence, why, claim, group, good, tanner, questions, fifth, remember, people

Table 3.9: The words for which usage is increasing or decreasing at 0.05 significance

In summary, the language model of students is changing over sessions but the essence of the change is not obvious. Is the change because of personal usage of language or is it because of using more terms related to scientific discourse? To investigate this question we designed a latent Dirichlet allocation based topic model [11] to model the language usage and track changes over time, which is described in the next section.

There is a limited literature on applying data and text mining tools to classroom spoken language [91]. Most of the related work applies data mining to extract patterns from discussion forums. As an example, [32] summarized the information in an asynchronous discussion forum by applying data mining algorithms to show the quality of discussions. However, none of these types of works attempt to analyze the text data. In another related work, Singley and Lam[98] introduce a tool, classroom Sentinel, to mine information related to students on the Web for finding useful patterns about the progress of a classroom. Similarly, this work has not used text

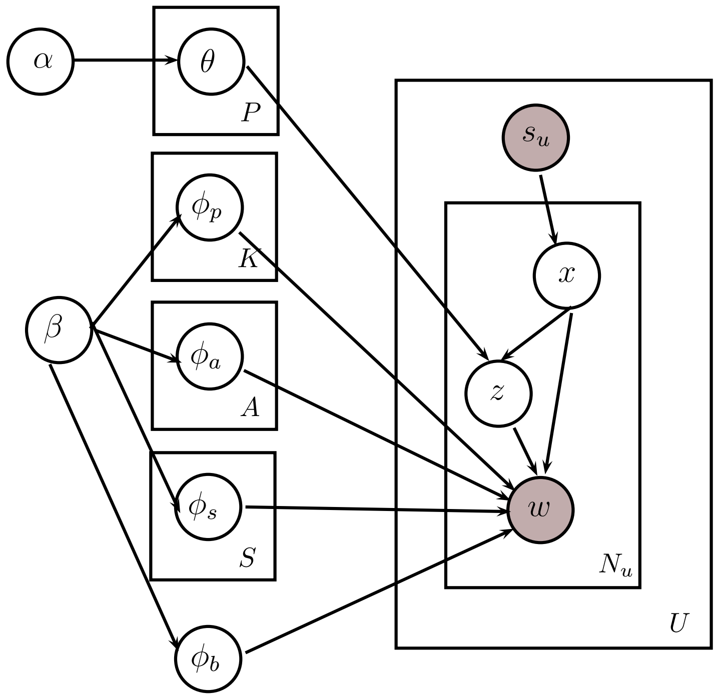


Figure 3.22: The graphical model of the proposed topic model

information of the classroom.

3.4.1 Model

Compared to the situation in LDA, in our dataset, instead of documents, there are utterances. However, the size of each utterance is very small and it might consist of only 1 word. As a result, assuming each utterance as a document and generating a specific language model for each utterance causes severe overfitting. Therefore, instead of utterance, we assume a language model for higher level entities. Each utterance is associated with three different higher level entities: an activity, a session, and a person. Here, we can assume that each entity can independently generate words. For example, the activity "claim and evidence" generates words such as "claim", and "support". Each session has a scientific subject such as the cardiovascular system. Then, words such as "heart" or "blood" can be seen as words generated by the session language model. A person usually has special ways of conveying concepts which results in specific language. For example, some people use words such as "actually" or "you know" more often.

Additionally, as mentioned in the previous section, we did not perform stemming and stopwording, so the data includes very general words such as "it", and "is". Therefore, we assume another entity is generating general words; we call it the background language representing general English language usage. Such an approach has been used in [46]. In summary, each word in an utterance can be generated from one of the following language models:

- Activity language
- Session language
- Personal language
- Background language

For the personal language models, we adapt a a similar strategy as used in the author-topic model [92]. In the author-topic model, each author has a distribution over different topics. When a group of scholars writes a paper, for each token, first, an author is chosen from a uniform discrete distribution. Then, a topic is chosen from the selected author language topic distribution, and finally a word is chosen from the topic language model. Here, instead of authors, there are a teacher and a group of students. However, there is no uncertainty about which person said the word as there was in the author-topic model. The uncertainty is about which of the four language models has been used to generate a word.

For the personal language models, we assume a person-topic model. That is, for each person there is probabilistic distribution over K topics ($\theta_{zp} = P(z|p)$). For each topic z, there is a distribution over different words ($\phi^{(p)}_{zw} = P(w|z)$). Note that we used the index (p) to distinguish the language model of personal language (which is of personal topics over words) from other language models.

For the activity language models, each activity a generates words independently from a distribution on words ($\phi_{aw} = P(w|a)$). Similarly for session language models, each session s generates words independently from a distribution on words ($\phi_{sw} = P(w|s)$). Finally, the background language model generates words from a distribution on words ($\phi^{(b)}_{w} = P_b(w)$).

Figure 3.22 represents the graphical model for the proposed topic model. There are U utterances in the dataset. Each utterance u is associated with a set $S_u = \{a_u, s_u, p_u\}$ which shows which person in which session and over which activity said the utterance. In each utterance, there are N_u tokens. For each token, a latent variable x is chosen from a uniform multinomial distribution which shows which language model (background, session, activity, or personal) has been used to generate the word. Note that the distribution over categories can be learned. If $x = p$ (personal language generated the word), then a topic z is chosen from a multinomial distribution with parameter θ_{p_u}, and then a word w is chosen from a multinomial distribution with parameter $\phi^{(p)}_{z}$. Similarly, if $x = b$, $x = a$, or $x = s$, a word w is chosen from a multinomial distribution with parameter ϕ_b, ϕ_{a_u}, or ϕ_{s_u} respectively.

Using the topic model proposed here, we will be able to distinguish the type of words students are using and, as a result, better understand the essence of changes. In summary the proposed topic model is as follows:

1. For each person p:
    (a) Choose $\theta_p \sim Dir(\alpha)$
2. For each topic z:
    (a) Choose $\phi_z^{(p)} \sim Dir(\beta)$
3. For each activity a:
    (a) Choose $\phi_a \sim Dir(\beta)$
4. For each session s:
    (a) Choose $\phi_s \sim Dir(\beta)$
5. Choose $\phi_b \sim Dir(\beta)$
6. For each utterance u:
    (a) For each token t
        i. Choose $x \sim Multinomial(\{.25, .25, .25, .25\})$
        ii. if $x = b$, then Choose a word $w \sim Multinomial(\phi^{(b)})$
        iii. if $x = a$, then Choose a word $w \sim Multinomial(\phi_{a_u})$
        iv. if $x = s$, then Choose a word $w \sim Multinomial(\phi_{s_u})$
        v. if $x = p$:

A. Choose a topic $z \sim Multinomial(\theta_{p_u})$

B. Choose a word $w \sim Multinomial(\phi_z^{(p)})$

For inference and learning the parameters, we used the version of Gibbs sampling used in [44]. Gibbs sampling is a special case of Monte Carlo Markov chain algorithm which is used to sample from the posterior distribution. In Gibbs sampling, the full conditional distribution of a parameter given the rest of the parameters is derived analytically and then samples are drawn iteratively. In Gibbs sampling for LDA, the topic of tokens is sampled given all other parameters are set. After enough iterations, all parameters can be estimated using the drawn samples. In our model, there are two types of hidden variables: the 4-category usage variable x and the topic z where the latter is in effect when $x = p$ (person is using her personal language). The conditional probability for latent variables of token i is as follows:

$$P(z_i = k, x_i = c | z_{-i}, x_{-i}) = \begin{cases} \frac{n_{w_i k}^{(-i)} + \beta}{n_k^{(-i)} + W\beta} \frac{n_{p_i k}^{(-i)} + \alpha}{n_{p_i}^{(-i)} + K\alpha} & \text{if } c = p \\ \frac{n_{w_i e_i}^{(-i)} + \beta}{n_{e_i}^{(-i)} + W\beta} & \text{if } c = e \neq p \end{cases}$$

where z_i, and x_i are the latent variables for token i, z_{-i}, x_{-i} are the assignment of latent variables for all tokens except for i, and $n_{ab}^{(-i)}$ and $n_a^{(-i)}$ show the count of tokens assigned to entities a and b. After running the Gibbs sampling algorithm, parameters can be estimated via harmonic mean [44].

3.4.2 Experimental results

We ran the proposed topic model on the dataset with parameters: $\alpha = 1$, $\beta = .01$, and $K = 5$. The Gibbs sampling algorithm was run for 3000 iterations and then 200 samples after each 10 iterations were drawn when the algorithm was run for

	St 1	St 2	St 3	St 4	St 5	St 6	St 7	St 8	St 9	St 10	St 11	St 12	St 13	St 14	All
slope	.000	.000	.003	.001	**.006**	-.002	-.002	-.000	-.001	.003	-.002	.003	**.006**	**.004**	**.001**
P-value	.974	.698	.355	.623	**.040**	.491	.252	.944	.784	.499	.650	.075	**.003**	**.003**	**.043**

Table 3.10: The test of linear regression of change in activity language model for individual students and all of them as a whole

another 2000 iterations.

To explore the main cause of change in students' language and the reason that their languages are becoming more similar to the teacher's, we used 4-category language usage versus personal language topic usage. The 4-category language usage is the probability distribution of using 4 different categories: background, activity, session, and personal. For example, let us assume a person has said 100 words during a session. The Gibbs sampling algorithm has assigned 20 of them to background, 25 to activity, 30 to session, and 25 to personal language. Then the vector for the 4-category usage probability of that person in the session is $\{.20, .25, .30, .25\}$. Furthermore, from the 25 words assigned to personal language, assume the assignment to 5 personal topic is $\{2, 8, 1, 0, 14\}$, resulting in a personal topic usage probability vector: $\{2/25, 8/25, 1/25, 0, 14/25\}$. Since such a vector can be identified for each person, the similarity between the teacher and students can be computed from the cosine similarity between these vectors. Figures 3.23 and 3.24 present the similarity between teacher's and students' 4-category language usage probability and personal topic usage probability respectively. For 4-category language usage probability, the same pattern as general cosine similarity (Figure 3.21) is observed while for personal topic usage probability no increase or decrease is detected. It means that the change in students' language is happening at a higher level than in personal usage of language. Figures 3.25 and 3.26 show the change in 4-category language usage probability over sessions. While no special pattern is observed for teacher, the activity usage probability for students is rising. Table 3.10 shows the linear regression test for the activity-based language usage probability of students and the increasing pattern for 3 of them, and for the whole student group, is significant at .05. No significant pattern was observed for other categories.

Comparing Figures 3.25 and 3.26 reveals another key difference between the language usage pattern of the teacher and students. Students tend to use more session based language than the teacher. That may be due to the fact that teacher uses a lecture based language which is based on more frequent usage of personal language while students tend to talk about the course content which is related to the subject of the session. Note that the relative decrease of session language usage does not mean that students are using the related terms less but it means they are using them less frequently compared to other categories. Based on Figure 3.25, it seems that there is a shift from session based language to activity based language which, as discussed earlier, can be interpreted as a shift from a passive presence to an active presence tied with more usage of scientific discourse.

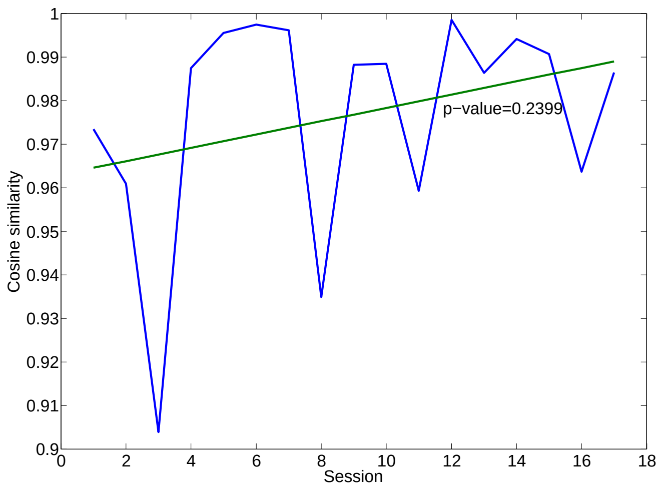


Figure 3.23: The cosine similarity between the 4-category language usage probability vector of the teacher and students (all students treated as one person) over time

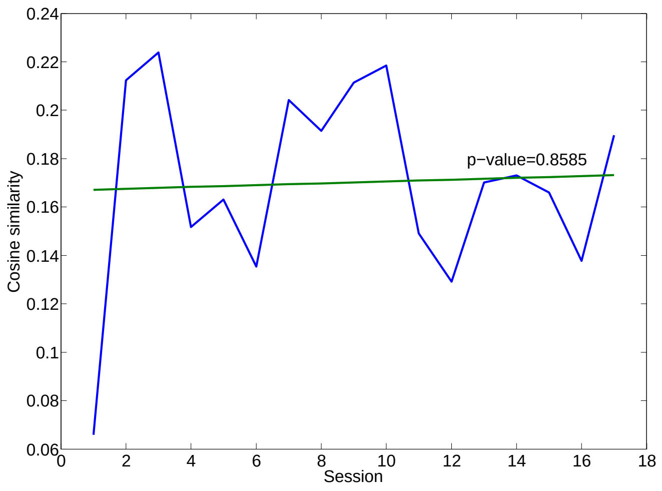


Figure 3.24: The cosine similarity between the personal topic usage probability vector of the teacher and students (all students treated as one person) over time

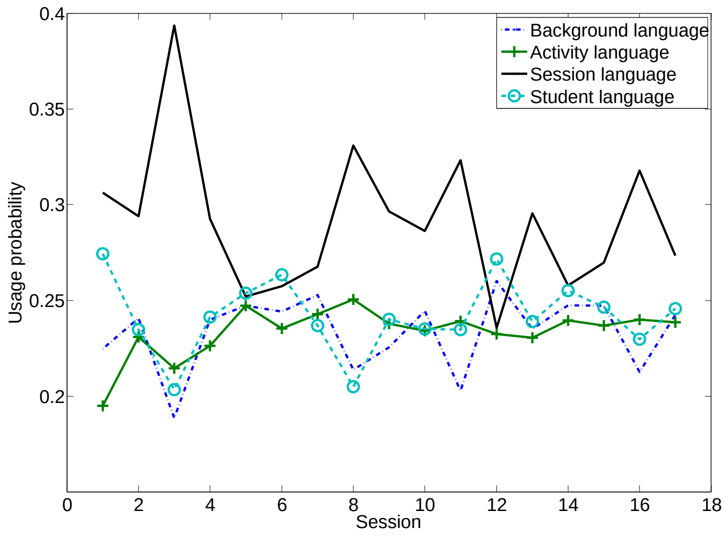


Figure 3.25: The 4-category language usage of students (all students treated as one person) over time

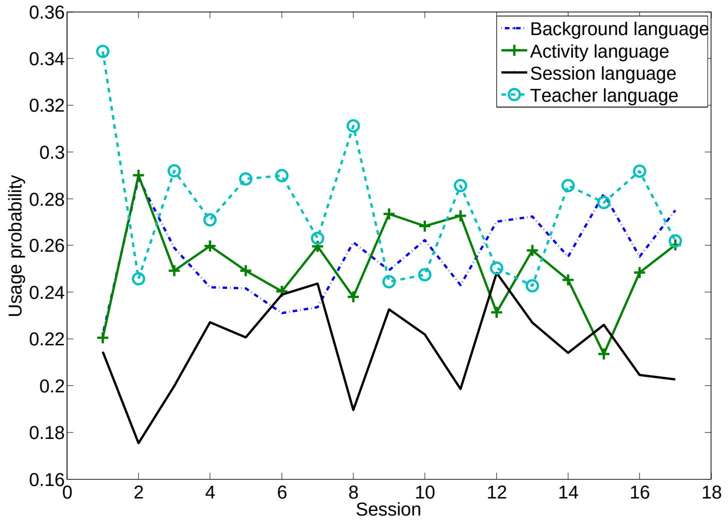


Figure 3.26: The 4-category language usage of the teacher

3.4.3 Conclusion

In this section, we introduced a customized topic model to decompose the spoken speech of a science classroom. Our experimental results show that students usage of activity language increases over time which is a sign of learning. Such an approach can be used to investigate the effect of teaching methods or represent an individual's progress. However, there are many factors that are not controlled in our dataset such as the order of session topics or activities over time. Therefore, the change in the language of students in classroom might have been caused by some other factors rather than learning. For example, students can be forced to use some words by some structured questions. Nevertheless, given the observed unstructured format of the classroom, we conjecture that learning is the strongest factor responsible for change.

CHAPTER 4
LATENT FEATURE NETWORKS

This section pictures a general framework for multi-relational learning — latent feature network. In the first section, the high-level properties of the proposed model is described. In the second section, a latent feature network for microarray classification is presented. Finally, a latent feature network for link prediction in a social network using side information is presented.

4.1 General model

In latent feature networks (LFN), each relationship is represented by a component. Each component is a latent variable model in which the relationship between two entities is modeled as a function of latent variables. We call these latent variables latent features since they are expected to represent the intrinsic features of the entities. If an entity is participating in different relationships, then it will have different local latent feature variables for each relation. However, different latent feature variables for an entity are forced to be dependent. This dependency can be an equality constraint in one extreme — the same latent variable for each relation — which underfits the data. On the other extreme, these local latent variables could be independent which results in overfitting the data. The main expected contribution of LFN is designing a general model for statistical relational learning that can model arbitrary relation types and is powerful in learning.

Let $X_i^{(r,k)}$ represent the latent feature variable of entity i with type k for

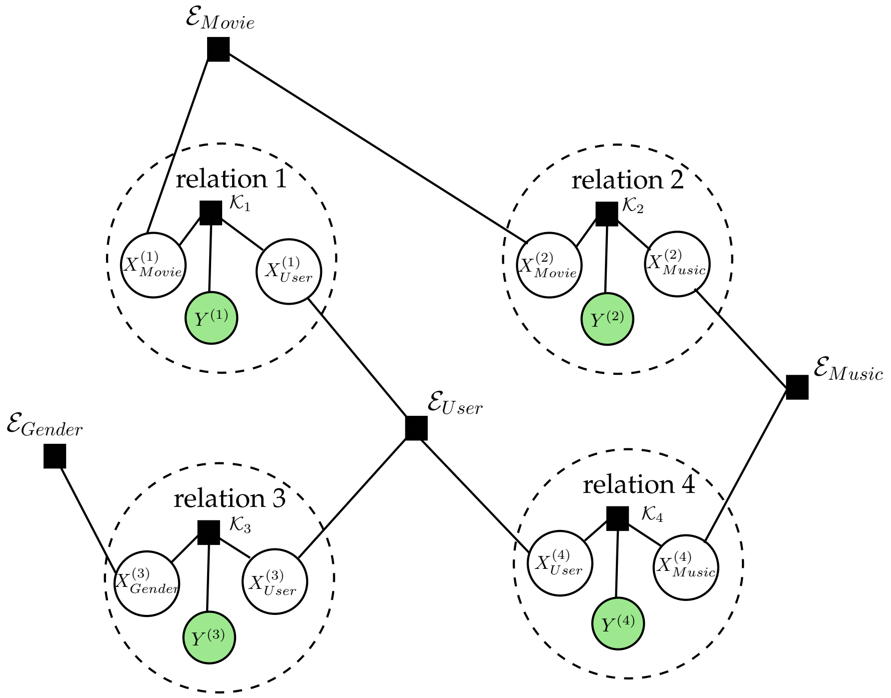


Figure 4.1: A latent feature network for 4 relations among 4 entity types

relation r. $Y^{(r)}_{ij}$ shows the relation between entity i from the first entity type and entity j from the second entity type participating in r. Y may be fully or partially observed while X is always hidden. Here, we consider dyadic relations — between two entity types. Generalizing to n-ary relations is straightforward from the modeling viewpoint. We define two functions to capture the dependencies among the variables. The first one is the relation kernel function $\mathcal{K}_r(X^{(r,k_{r1})}, X^{(r,k_{r2})}, Y^{(r)})$ which models the relationship r between entity type k_{r1} and k_{r2}. Note that a relationship between entity i of the first type and entity j of the second type may use latent variables of other entities rather than i and j as we saw in the co-occurrence data modeling in Section 3.3. Another dependency regards correlation between the same entities participating in different relationships. Let $\mathcal{E}_k()$ be a function of $X^{(r,k)}$ for all relations r in which entity type k participates. This function enforces dependency among the same entities of type k.

To clarify the notation, we give a fictional example about a multi-context recommendation system. In this example, entity types are:

1. Movie
2. User
3. Book
4. Gender.

Note that gender is usually considered as feature in machine learning algorithms but we can model it as a relationship as well. Assume that 4 relation types are observed:

1. Movie-User: rating of a user for a movie
2. Movie-Book: is true if a movie is based on a book
3. User-Gender: gender of a user
4. Book-User: rating of a user for a book.

Using the representative indexes above, $X^{(1,1)}$ and $X^{(2,1)}$ represent the local latent variables of movies in Movie-User and Movie-Book relationships. $X^{(1,2)}$, $X^{(3,2)}$, and $X^{(4,2)}$ represent the local latent variables of users in Movie-User, User-Gender, and Book-User relationships respectively. $X^{(2,3)}$ and $X^{(4,3)}$ represent the local latent variables of books in Movie-Book and Book-User relationships. $X^{(3,4)}$ represents the latent variable of gender categories (male and female) for the User-Gender relationship. $Y^{(1)}$ is the rating of a user for a movie, $Y^{(2)}$ is one if a movie is based on a book, $Y^{(3)}$ is the assignment of a gender to a user and $Y^{(4)}$ is is the rating of a user for a book.

A latent feature network can be represented by a factor graph. Random variable nodes represent latent feature variables X and relationship or observed variables Y. Function nodes represent relationship kernels $\mathcal{K}$ and dependencies between latent feature variables $\mathcal{E}$.

Figure 4.1 represents a latent feature network for the example explained above. Movie participates in relations 1 and 2 and therefore a latent variable for each relation is defined — $X^{(1,1)}$ and $X^{(2,1)}$. Similarly, user participates in relations 1, 3, and 4, $X^{(1,2)}$, $X^{(3,2)}$, and $X^{(4,2)}$ for each relation. Factor function $\mathcal{K}_r$ for $r = 1, ..., 4$ models the relationship as a function of latent feature variables. Factor function $\mathcal{E}_k$ for $k = 1, .., 3$ models the dependency among the latent feature variables of the same entity participating in different relationships.

Given the defined variables and relations for the latent feature network, the joint probability distribution over all random variables is given by

$$P(X, Y, \Theta) = \frac{1}{Z} \exp\left(\sum_r \mathcal{K}_r + \sum_k \mathcal{E}_k\right) \tag{4.1}$$

where $Z = \int_{X,Y,\Theta} \exp\left(\sum_r \mathcal{K}_r + \sum_k \mathcal{E}_k\right)$ is the normalizing or partition function. This may be a sum instead of integral for some relation variables Y. Θ represents the parameters that might be used in the modeling. In the example in Figure 4.1, the joint probability distribution is

$$\begin{aligned}P(X,Y,\Theta) = \frac{1}{Z}\exp[&\mathcal{K}_1(X^{(1,1)},X^{(1,2)},Y^{(1)}) + \mathcal{K}_2(X^{(2,1)},X^{(2,3)},Y^{(2)}) \\ &+ \mathcal{K}_3(X^{(3,4)},X^{(3,2)},Y^{(3)}) + \mathcal{K}_4(X^{(4,2)},X^{(4,3)},Y^{(4)}) \\ &+ \mathcal{E}_1(X^{(1,1)},X^{(2,1)}) + \mathcal{E}_2(X^{(1,2)},X^{(3,2)},X^{(4,2)}) + \mathcal{E}_3(X^{(2,3)},X^{(4,3)})]. \quad (4.2)\end{aligned}$$

Note that some of the parameter variables Θ may exist in any factor.

The main contribution of the latent feature networks is providing a unified model for learning arbitrary relational data with any number of relations and any type of relationships. Although a few latent variable models which are capable of learning multi-relational data have been proposed, they are usually limited to one type of relationships. Furthermore, one can deduce those models from LFN as LFN is more general. The most important related work in the literature is collective matrix factorization [96, 97] which assumes the same latent variable for an entity that participates in different relationships. Such approach limits the learning power as different relationships demand different features. LFN resolves this problem by assuming a latent variable for each relationship, and then it learns the dependency between relationships.

4.2 Microarray classification

In [60], we proposed a latent feature network for classifying gene expression microarray data. We use maximum a posteriori estimation to learn the latent features for the gene expression microarray classification problem. The experiments over a small dataset are promising as the LFN outperforms SVM — the state-of-the-art

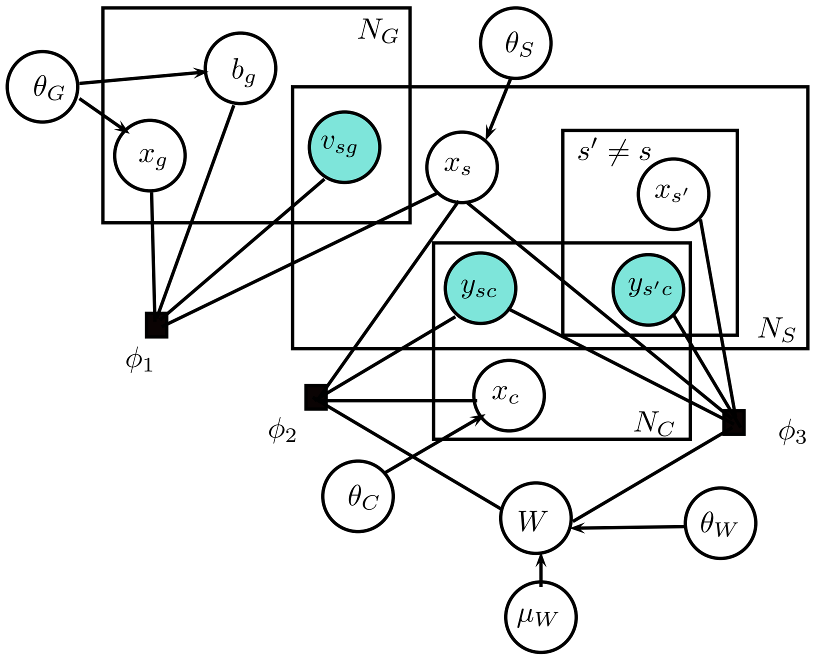


Figure 4.2: The graphical model of the latent feature network for microarray classification

classification method in microarray classification.

Because of the availability of huge gene expression microarray datasets, many machine learning and pattern recognition algorithms have emerged to analyze them [102]. One of the important applications of gene expression microarray data is in cancer diagnosis [99]. Here, we try to discover the latent features of entities in a low-dimensional space via random variable or nodes in the factor graph. That is, for each entity, a random variable is defined which is latent variable. The relationship between entities is captured via a function of the latent features — representative random variables — which is a factor in the factor graph. The graphical model for the gene expression microarray classification problem, is shown in Figure 4.2 where there are 3 types of entities: genes, samples and cancer categories. x_s, x_g and x_c are the random variables representing latent features of samples, genes and categories. Two types of relationships are of interest: expression level (shown by v_{sg}) and samples' cancer category (shown by y_{sc}). The expression level of a gene for a sample will be a function of the latent features of the gene and the sample (factor ϕ_1). Similarly, the cancer category relationship is a function of the latent features of the sample and cancer categories (factor ϕ_2). To boost classification power, we force similarity between the latent features of two samples with the same cancer category (factor ϕ_3).

The proposed latent feature network is similar to collective matrix factorization [96] in some aspects as we use weighted linear functions for factors. However, collective matrix factorization is capable of predicting values from several matrices while the goal of our model is classification. Collective matrix factorization assumes a generative model from latent features to relationships — a directed graph — which is limiting, while LFN assumes correlation between latent feature variables — which can be directed or undirected given the definition of potentials. Therefore, applying collective matrix factorization to gene expression microarray classification is not straightforward. However, non-negative matrix factorization has been used for clustering microarray data [16] where the learning is only based on the microarray data — an unsupervised approach. While this approach is worthwhile since it does not need labels for training, it is not as powerful as supervised learning approaches.

4.2.1 Model

In the latent feature network for microarray classification, we introduce one random variable per entity. In classification using microarray gene expression data, there are 3 groups of entities: x_s — a $(1 \times d)$ vector — denotes the latent feature for sample s, x_g — a $(1 \times d)$ vector — denotes the latent feature for gene g, b_g — a scalar — denotes the bias in expression for gene g, and x_c — a $(1 \times d)$ vector — denotes the latent feature for category c. Generally, latent features are vectors with the same size. However, it is possible to assume different sizes for the entities of different types. As a result, the relationship between entities can be captured as a function of entities' latent feature variables — a factor. Here, two types of relationships exist. The expression level of each gene g for each sample s, denoted by v_{sg}, and the category c of each sample s, denoted by y_{sc} which is a binary variable and is one if sample s is in category c and zero otherwise. For the expression level relationship, we define the factor as

$$\phi_1(x_g, x_s, v_{sg}) = \exp\left(-\tau \frac{(x_g x_s^T + b_g - v_{sg})^2}{2}\right). \tag{4.3}$$

Such a factor implies that the distribution over v_{sg} is Gaussian with mean $x_g x_s^T + b_g$ and precision (one over variance) τ. For the category relationship, we define the factor

$$\phi_2(x_s, x_c, y_{sc}) = \exp(x_s W x_c^T y_{sc}), \tag{4.4}$$

where W — a $(d \times d)$ matrix — is a weight matrix to transfer x_s from the microarray gene expression data space to the classification space. Note that (4.4) resembles the numerator of the multinomial logistic regression probability. However, they are not equivalent due to our third factor definition. In the third factor, the relationship between categories and latent features of two different samples is captured with the factor

$$\phi_3(x_s, x_{s'}, y_{sc}, y_{s'c}) = \exp\left(\frac{x_s W W^T x_{s'}^T y_{sc} y_{s'c}}{\sum_{s''} y_{s''c}}\right), \tag{4.5}$$

where $\sum_{s''} y_{s''c}$ is a normalizing term for unbalanced categories.

The factor graph representing the explained model is shown in Figure 4.2 via plate notation. In plate notation, the enumeration over random variables is denoted by plates. In classification based on gene expression data, there are N_G genes, N_S samples, and N_C categories. For the relationship between two distinct samples, the plate with index $s' \neq s$ is used. The priors over latent features are shown by directed relationships. The Gaussian prior with mean zero is assumed for all latent variables (relaxing the zero-mean assumption is straightforward). Each θ in the graphical model is the precision for the relevant variable where we assumed Multivariate Gaussian with independent elements. The prior mean of matrix W is the matrix μ_W.

The probability distribution over all random variables is

$$\begin{aligned}
&P(X, W, Y, V|\Theta) = \\
&\frac{1}{Z}\exp(-\alpha\tau\sum_{gs}\frac{(x_g x_s^T + b_g - v_{sg})^2}{2} + (1-\alpha)\left(\sum_{sc} x_s W x_c^T y_{sc} + \sum_{s<s',c}\frac{x_s W W^T x_{s'}^T y_{sc} y_{s'c}}{\sum_{s''} y_{s''c}}\right) \\
&\quad - \frac{\theta_G}{2}\sum_g(||x_g||_2^2 + b_g^2) - \frac{\theta_S}{2}\sum_s ||x_s||_2^2 - \frac{\theta_C}{2}\sum_c ||x_c||_2^2 - \frac{\theta_W}{2}||W - \mu_W||_2^2),
\end{aligned} \tag{4.6}$$

where Z is the partition function and $0 < \alpha < 1$ is the weight for combining expression level factor and classification factor. Such a weighting is necessary since the data is usually unbalanced — very fat in the gene expression side.

For learning the latent features, we only use the samples for which the category is given — the training set. However, it is possible to use the unlabeled data given the latent feature network in a semi-supervised learning manner — similar to the work of Zhu and Ghahramani [111] — which we leave for future directions. Therefore, in the learning phase, the latent feature parameters are learned given the observed variables v_{sg} and y_{sc}. Here we use maximum a posteriori probability (MAP) estimation which is the mode of the posterior distribution over the latent feature variables.

As another strategy, instead of finding the right weight parameter α, we optimize the probability distribution by alternating between the factors for the expression level and the category. That is, in an iterative way, first we optimize the log-posterior function over variables x_g, b_g, and x_s based on the factor f_1, and then over variables x_s, x_c, and W based on the factors f_2 and f_3. This is because of the fact that based on the joint probability distribution (4.6), the only common random variable between f_1, f_2, and f_3 is x_s. To make x_s satisfy both groups, we update the prior of x_s between alternating. As mentioned, the prior over x_s is Gaussian with mean zero and precision θ_S. Given $\hat{x}_s$ as the output of one half of the algorithm, the prior over x_s will be a Gaussian with mean $\hat{x}_s/2$ and precision θ_S which is the posterior of the x_s given the first prior and the new received $\hat{x}_s$. That way, the prior mean over x_s for updating based on factor 1 is half of the x_s as the output of updating based on factors 2 and 3 and vice versa. We show the updated prior over x_s by μ_s.

For optimizing the log-posterior function based on the first factor, first we update x_g and b_g to fit them to the x_s coming from factors 2 and 3. The equation for updating x_g is

$$x_g = (-\sum_s (b_g - v_{sg}) x_s)(\sum_s x_s^T x_s + \theta_G I)^{-1} \tag{4.7}$$

for all g, where I is the identity matrix. The equation for updating b_g is

$$b_g = \frac{-\sum_s (x_s x_g^T - v_{sg})}{N_S + \theta_G} \tag{4.8}$$

for all g. The equation for updating x_s is

$$x_s = (\theta_S \mu_s - \sum_g (b_g - v_{sg}) x_g)(\sum_g x_g^T x_g + \theta_S I)^{-1}. \tag{4.9}$$

Optimizing the log-posterior function based on the second and third factors is more complex due to the difficulty of computing the partition function. This is because of the exponential growth for summing over all possible y_{sc} variables in computing the partition function. To remedy this problem, instead of optimizing the log-posterior function given all variables, we work on the conditional probability of one sample given the labels for the rest of samples:

$$P(y_{sc}, X, W | y_{s' \neq s, c}) = (const) \frac{\exp\left(\sum_c x_s W x_c^T y_{sc} + \sum_{s' \neq s, c} \frac{x_s W W^T x_{s'}^T y_{sc} y_{s'c}}{\sum_{s'} y_{s'c}}\right)}{\sum_{y_{sc}} \exp\left(\sum_c x_s W x_c^T y_{sc} + \sum_{s' \neq s, c} \frac{x_s W W^T x_{s'}^T y_{sc} y_{s'c}}{\sum_{s'} y_{s'c}}\right)}$$
$$\exp\left(-\frac{\theta_S}{2} \sum_s ||x_s - \mu_s||_2^2 - \frac{\theta_C}{2} \sum_c ||x_c||_2^2 - \frac{\theta_W}{2} ||W - \mu_W||_2^2\right), \tag{4.10}$$

where *const* does not depend on any decision variable and sum over y_{sc} means enumerating over all categories for sample s. In this phase, we update the variables with regard to one sample via gradient ascent. The order of going over different samples is randomized. Note that the $x_{s' \neq s}$ variables are updated given (4.10) for sample s. The inside loop for updating $x_{s'}$ is randomized as well.

```
Input [$\{y_{sc}\}$, $\{v_{sg}\}$, $\theta_C$, $\theta_S$, $\theta_G$, $\theta_W$, $\mu_W$]
Output [$\{x_s\}$,$\{x_g\}$,$\{b_g\}$,$\{x_c\}$,$W$]
Initialize [$\{x_s\}$,$\{x_g\}$,$\{b_g\}$,$\{x_c\}$,$W$]
$\forall s$ $\mu_s \leftarrow x_s/2$
Repeat
  Phase 1
     Update $\{x_g\}$ using (4.7)
     Update $\{b_g\}$ using (4.8)
     Update $\{x_s\}$ using (4.9)
  $\forall s$ $\mu_s \leftarrow x_s/2$
  Phase 2
     Repeat
        For s $\in$ RandomizedOrder
           optimize (4.10) over $\{x_s\}$,$\{x_c\}$,$W$ by gradient ascent
        End
     Until(convergence)
  $\forall s$ $\mu_s \leftarrow x_s/2$
Until (convergence)
```

Figure 4.3: Iterative algorithm for microarray classification

The algorithm for learning latent features is presented in Figure 4.3. Note that we repeat phase 2 until convergence since in phase 1 we use closed-form updates while in phase 2 we use gradient ascent and changes are slower.

Finally, given the latent feature variables, we can classify test sample t. First, using

$$x_t = (-\sum_g (b_g - v_{tg})x_g)(\sum_g x_g^T x_g + \theta_G I)^{-1}, \tag{4.11}$$

we map the sample t in the new space. Then using

$$P(y_{tc}|Y_{train}, X, W) = \frac{\exp\left(\sum_c x_t W x_c^T y_{tc} + \sum_{sc} \frac{x_t W W^T x_s^T y_{tc} y_{sc}}{\sum_s y_{sc}}\right)}{\sum_{y_{tc}} \exp\left(\sum_c x_t W x_c^T y_{tc} + \sum_{sc} \frac{x_t W W^T x_s^T y_{tc} y_{sc}}{\sum_s y_{sc}}\right)}, \tag{4.12}$$

we can predict the class of the sample t.

4.2.2 Experimental results

For experimental evaluation, we used the 9-tumors dataset explained in [99]. In this dataset, there are 60 samples, 5726 genes, and 9 categories. We compared LFN to SVM as a best known classifier for this problem [99]. LIBSVM [22] was used for evaluating SVM.

For SVM, we optimized over RBF and polynomial kernels via cross-validation. The best result was a linear kernel with $C = 200$. As preprocessing, we tried different normalizing procedures and for SVM, linear scaling data to [0,1] interval worked best.

For LFN, we optimized over $\theta_G = \theta_S = \theta_s = \theta$ (equal θ for all entities for simplicity of optimization) and $\theta = 30$ gave the best result. The step size for gradient ascent was set to $1.0e-5$. Latent features were initialized using the standard normal distribution. As preprocessing, we tried different normalizing procedures and normalizing by dividing data by the standard deviation of each gene worked best.

Due to the small size of the data, we repeated 10-fold cross-validation experiments 10 times — a total of 100 experiments. In each run, data was split into 10 mutually exclusive groups randomly. We did not implement stratified splitting since it impaired randomness given the small size of the dataset.

The results are given in Table 4.1. We use accuracy — the percentage of correctly labeled samples — for comparing LFN and SVM. LFN outperforms SVM in all runs and for eight runs the differences were significant at 0.05. In total, LFN outperforms SVM with *p*-value equal to 1.4e-10.

	run 1	run 2	run 3	run 4	run 5	run 6	run 7	run 8	run 9	run 10	total
SVM	48.33	53.33	53.33	51.67	55.00	55.00	51.67	53.33	51.67	58.33	53.17
LFN	**65.00**	63.33	**71.67**	**63.33**	**71.67**	**68.33**	**70.00**	**63.33**	**66.67**	68.33	**67.17**
p-value	0.002	0.109	0.006	0.044	0.032	0.026	0.009	0.012	0.041	0.130	1.4e-10

Table 4.1: The accuracy results of ten runs of 10-fold cross-validation experiments (total of 100 runs)

4.2.3 Conclusion

In this section, we propose a novel classification approach for gene expression microarray classification via latent feature network. LFN assumes a latent feature variable for each entity — a random variable in a factor graph — and captures the relationship between entities with a function of the latent feature variables — a factor in a factor graph.

We use maximum a posteriori probability (MAP) estimation to learn the latent features for the gene expression microarray classification problem. The experiments over a small dataset are promising as LFN outperforms support vector machine (SVM) — the state-of-the-art classification method in microarray classification [99]. We conjecture that LFN is successful because it behaves as a supervised dimensionality reduction method. It embeds entities in a low-dimensional dot product space where the samples from the same class are more similar to each other. In comparison to SVM that uses kernel trick to transfer original space to a dot product space, LFN constructs the dot product space from scratch using both features and labels.

4.3 Social network analysis

In this section, a latent network model is proposed for link prediction in social networks. Latent variable models are successful approaches for social network analysis [48]. A social network consists of social actors and edges between them which usually convey concepts such as friendship. In latent variable models, individuals are mapped into a latent space and the relationship between them is a function of the position of individuals in the latent space.

In this section, we exploit side information for better social network analysis. Side information might be the interaction of individuals with other entities. One example is the ratings of individuals for items as we have seen in the collaborative filtering problem. Here, we propose a latent feature network for modeling a social network with a side network of bookmarked URLs. We derive a Gibbs sampling algorithm for Bayesian inference and run experiments on a real world dataset — Delicious dataset[1]. We use the two networks that exist in this dataset: a bookmarking network and a social network. Based on the experiments, using both networks significantly outperforms using only the social network for link prediction.

[1]http://www.grouplens.org/node/462

4.3.1 Model

In the latent feature network for social network analysis, we introduce one random variable per entity. In link prediction with a side network of bookmarking, there are two groups of entities: individuals and URLs. We use latent variable $X_i^{(1)}$ — a $1 \times k_1$ vector — for features of individual i participating in the social network relationship, latent variable $X_i^{(2)}$ — a $1 \times k_2$ vector — for features of individual i participating in the bookmarking relationship, and Y_h — a $1 \times k_2$ vector — for features of URL h participating in the bookmarking relationship. Note that the social network relationship is modeled by a k_1-dimensional latent variable feature space while the bookmarking relationship is modeled by a k_2-dimensional latent feature space. Here, two types of relationships exist: the friendship between individuals i and i' denoted by the friendship link binary variable $l_{ii'}$ which is one if there are friends and zero otherwise, and the bookmarking relationship between individual i and URL h denoted by the the bookmarking binary variable b_{ih} which is one if user i has bookmarked URL h and zero otherwise.

The generative model for the proposed model is depicted in 4.4 which is as follows:

1. Choose precision matrix $\Lambda_X \sim \mathcal{W}ishart(W_0, \nu_0)$
2. Choose precision matrix $\Lambda_Y \sim \mathcal{W}ishart(W_0, \nu_0)$
3. For each individual i:
   (a) chose latent variables $\begin{pmatrix} X_i^{(1)} \\ X_i^{(2)} \end{pmatrix} \sim \mathcal{N}ormal(0, \Lambda_X)$
4. For each URL h:

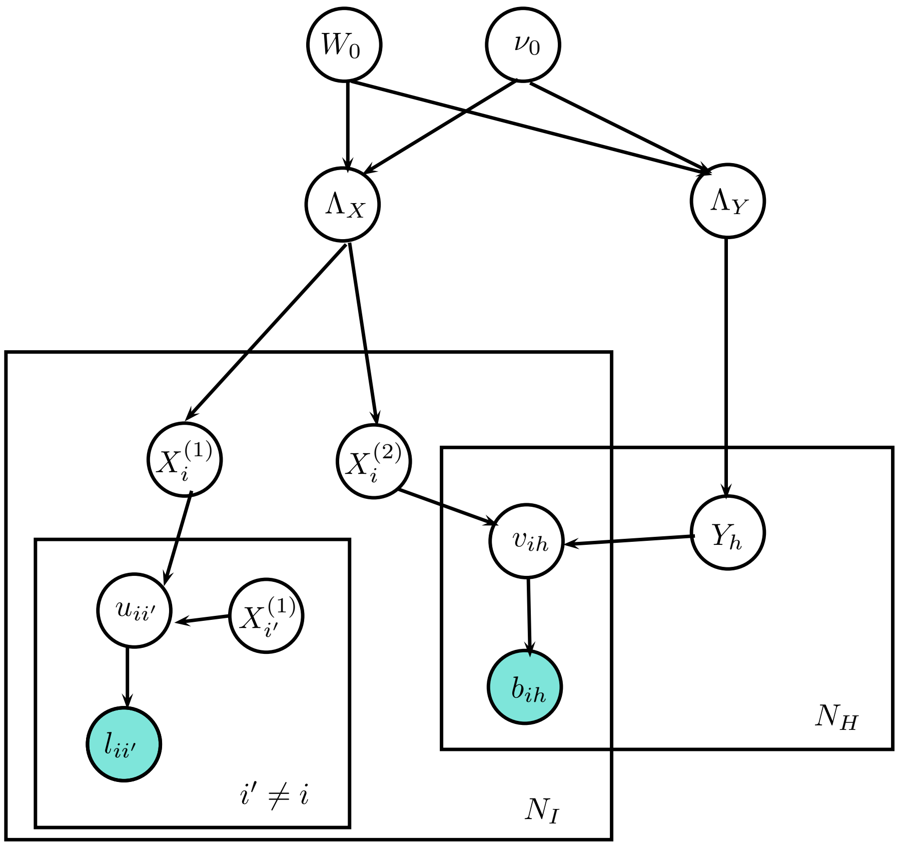


Figure 4.4: The graphical model of the latent feature network for social network analysis with side information

(a) Choose latent variable $Y_h \sim \mathcal{N}ormal(0, \Lambda_Y)$

5. For each pair of individuals (i, i') where $i \neq i'$:

   (a) Choose latent variable $u_{ii'} \sim \mathcal{N}ormal(X_i^{(1)} {X_{i'}^{(1)}}^T, 1)$

   (b) Set link variable $l_{ii'} = \begin{cases} 1 & \text{if } u_{ii'} > 0 \\ 0 & \text{otherwise} \end{cases}$

6. For each pair of individual and URL (i, h):

   (a) Choose latent variable $v_{ih} \sim \mathcal{N}ormal(X_i^{(2)} {Y_h}^T, 1)$

(b) Set bookmark variable $b_{ih} = \begin{cases} 1 & \text{if } v_{ih} > 0 \\ 0 & \text{otherwise} \end{cases}$

Here we use auxiliary random variables $u_{ii'}$ and v_{ih} similar to what is used in [2]. The benefit of such a modeling is that the conditional probability distributions are known which is very helpful in inference. Note that the precision matrix Λ_X enforces transferring knowledge between two different relationships. In the following, we derive the Gibbs sampling algorithm to derive the posterior distribution of needed statistics.

4.3.1.1 Gibbs sampling

To infer the latent variables we use Gibbs sampling which was explained in the second chapter. In Gibbs sampling, we need to derive the conditional probability of each latent variable given the rest of latent variables. The conditional distribution of precision matrices is as follows:

$$\Lambda_X | X \sim \mathcal{W}ishart\left((W_0 + \sum_i X_i^T X_i)^{-1}, N_I + \nu_0 \right), \tag{4.13}$$

where $X_i = \begin{pmatrix} X_i^{(1)} \\ X_i^{(2)} \end{pmatrix}$, and

$$\Lambda_Y | Y \sim \mathcal{W}ishart\left((W_0 + \sum_h X_h^T X_h)^{-1}, N_H + \nu_0 \right). \tag{4.14}$$

The conditional distribution of auxiliary random variables $u_{ii'}$ and v_{ih} are as follows:

$$u_{ii'} | l_{ii'}, X_i^{(1)}, X_{i'}^{(1)} \sim 1(u_{ii'} \geq 0)^{l_{ii'}} 1(u_{ii'} < 0)^{1-l_{ii'}} \mathcal{N}ormal\left(X_i^{(1)} {X_{i'}^{(1)}}^T, 1 \right), \tag{4.15}$$

and

$$v_{ih} | b_{ih}, X_i^{(2)}, Y_h \sim 1(v_{ih} \geq 0)^{b_{ih}} 1(v_{ih} < 0)^{1-b_{ih}} \mathcal{N}ormal\left(X_i^{(2)} Y_h^T, 1 \right). \tag{4.16}$$

Finally, the conditional distribution of latent feature variables Y_h and X_i are as follows:

$$Y_h|X^{(2)}, v, \Lambda_Y \sim \mathcal{N}ormal\left(\left(\textstyle\sum_i v_{ih} X_i^{(2)}\right)\Sigma_h, \Sigma_h\right), \tag{4.17}$$

where

$$\Sigma_h = (\Lambda_Y + \sum_i {X_i^{(2)}}^T X_i^{(2)})^{-1}, \tag{4.18}$$

and

$$X_i|Y, X_{-i}, u, v, \Lambda_X \sim \mathcal{N}ormal\left(\left(\textstyle\sum_{i' \neq i} u_{ii'} X_{i'}^{(1)} \quad \sum_h v_{ih} Y_h\right)\Sigma_i, \Sigma_i\right), \tag{4.19}$$

where X_{-i} includes all latent variables X except for X_i and

$$\Sigma_i = \left(\Lambda_X + \begin{pmatrix} \sum_{i' \neq i} {X_{i'}^{(1)}}^T X_{i'}^{(1)} & 0 \\ 0 & \sum_h Y_h^T Y_h \end{pmatrix}\right)^{-1}. \tag{4.20}$$

The Gibbs sampling algorithm is depicted in Figure 4.5. First, we iteratively sample random variables for a specific number of iterations which is known as burn-in phase. Then we sample needed statistics which are the score of a link score — $X_i^{(2)} {X_i^{(2)}}^T$ for each $i' \neq i$.

4.3.2 Experimental results

For experimental evaluations, we used a subset of the Delicious dataset. We sampled from this dataset so that remaining individuals and URLs have enough support in the social network matrix and the bookmarking network. In the subset, there are $N_I = 869$ individuals and $N_H = 2214$ URLs. The main goal is predicting the friendship link between two individuals. For evaluation, for each individual i we randomly selected a friend and eliminated the link from the social network. Then for each

```
Input [$\{l_{ii'}\}$, $\{b_{ih}\}$, $W_0$, $\nu_0$]
Output [$Pr(l_{ii'} = 1)$]
Initialize [$\Lambda_X$,$\Lambda_Y$,$\{X_i^{(1)}\}$,$\{X_i^{(2)}\}$,$\{Y_h\}$,$\{u_{ii'}\}$,$\{v_{ih}\}$]
Repeat
    Sample $\Lambda_X|X$
    Sample $\Lambda_Y|Y$
    $\forall_{i<i'}$ sample $u_{ii'}|l_{ii'}, X_i^{(1)}, X_{i'}^{(1)}$
    $\forall_{i,h}$ sample $v_{ih}|b_{ih}, X_i^{(2)}, Y_h$
    $\forall_h$ sample $Y_h|X^{(2)}, v, \Lambda_Y$
    $\forall_i \in RandomizedOrder$ sample $X_i|Y, X_{-i}, u, v, \Lambda_X$
Until (Sufficient samples achieved)
```

Figure 4.5: Gibbs sampling algorithm for the social network analysis with side bookmarking network

selected link, 50 individuals who were not friends with individual i were randomly selected. We use metric average rank for evaluation:

$$AverageRank = \frac{\sum_i Rank(j_i, S_i)}{N_I} \tag{4.21}$$

where j_i is the index of the individual who is friends with individual i and S_i is the set of individuals who are not friends with individual i, and $Rank(j_i, S_i)$ computes the rank of the individual with true friendship among all individuals. The rank can be between 1 and 51.

Here, we compare two algorithms: the LFN algorithm explained in the previous section which exploits both friendship and bookmarking relationship — shown by LFN-FB — and an LFN algorithm that only exploits the friendship relationship

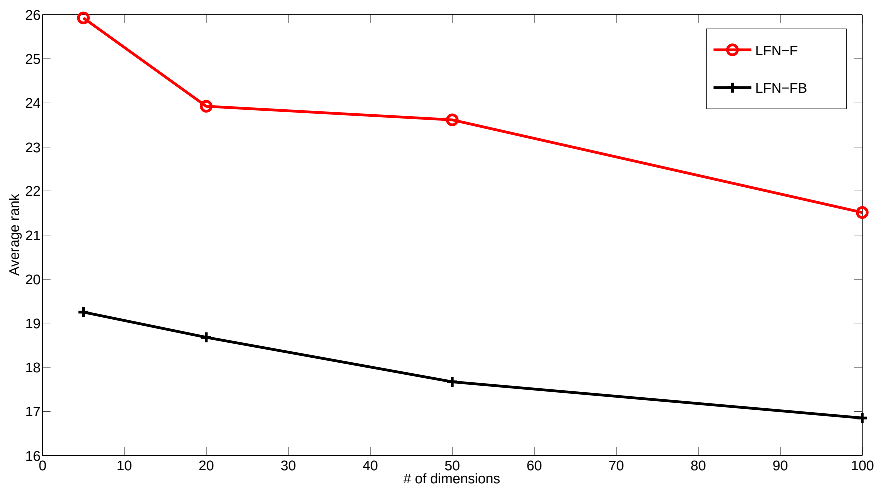


Figure 4.6: The average rank results for LFN-F versus LFN-FB with different dimensions

with the same setup of LFN-FB but without the bookmarking component — shown by LFN-F. LFN-F is very similar to the algorithm proposed in [48] but with Probit model instead of Logit model.

The results are given in Figure 4.6. The lower average rank is better. Using all dimensions, LFN-FB outperforms LFN-F (significant at p-value$< 10^{-4}$).

4.3.3 Conclusion

In this section, we derive a model and inference algorithm for social network analysis with side information. We show that link prediction via latent space models can be improved by using extra information given latent feature network setup.

CHAPTER 5
CONCLUSION AND FUTURE DIRECTIONS

5.1 Summary and conclusion

In this dissertation, we explored using latent variable models for modeling and analyzing relational learning problems. In Chapter 3, single-relational models were proposed for tasks such as prediction or visualization. In Chapter 4, we propose a component-wise latent variable model for learning multi-relational data — latent feature network (LFN). The path from the single-relational models to LFN may be considered as bottom-up. We started by learning only one relation in early works and then moved to multi-relational learning by proposing latent feature networks.

In Sections 3.1 and 3.2, we present two models for collaborative filtering applications. The dominant latent variable model for collaborative filtering — matrix factorization — uses dot product. In Section 3.1 we proposed using squared Euclidean distance instead of dot product. Such an approach helps to visualize data, facilitates fast recommendation generation, and gives an efficient way to include new users and items in the system. The disadvantage of using Euclidean distance is that the resulting objective function is harder to optimize. We have not compared these two mathematically but according to experiments, optimizing the objective function resulting from using dot product is easier. For example, dot product kernel is not very sensitive to the initial solution — for iterative algorithms such as gradient descent — while for Euclidean distance kernel, random multi-start is very helpful.

In Section 3.2, a scalable online algorithm is explained for collaborative filtering. Contrary to other latent variable models discussed in this chapter, this model uses discrete latent variables represented in the format of co-clustering. Given the combinatorial nature of the resulting problem, we used an evolutionary algorithm for better results.

In Section 3.3, we propose a novel model for learning co-occurrence data — latent logistic allocation. The state-of-the-art models in the literature use discrete latent variables for modeling categorical data including co-occurrence data. In this work, we showed that we still can use the power of continuous latent variables even in categorical data. The proposed model outperforms the state-of-the-art algorithms in several areas. Furthermore, we moved towards a more general model for giving a flexible framework for non-continuous data.

We present a latent topic model for analyzing the language evolution in a science classroom in Section 3.4. Contrary to other latent variable models, in this model, latent variables have real meaning. That is, we used latent variables to resolve a confusion about hidden variables.

In Section 4.1, we present latent feature network as a general-purpose framework for multi-relational learning. Two specific derivations of LFN were presented in the following sections. Section 4.2 gives a model for two components: one real-valued and another categorical for microarray data classification. This is an example of learning data that can be modeled by propositional learning approaches (e.g. SVM) with relational approaches. In Section 4.3, a latent feature network for social network analysis with a side network is derived. We employed Gibbs sampling for Bayesian inference. The experimental results show that using the side information can drastically improve the link prediction task in a social network.

The results presented in Sections 4.2 and 4.3 reveal the usefulness of multi-relational learning via latent feature networks. In the microarray classification problem, the fat shape of the feature space — there are only few samples and many features — causes poor performance of propositional models. On the other hand, the features are homogeneous entities — different genes. Latent feature network uses this to transfer all genes to an expression relationship space and then use that space for better prediction. We conjecture that in many-heterogeneous-features situations, our method cannot perform as well since the unified space for all features may not be as effective. In the social network analysis example, we examine transferring information via latent feature networks. This helps to perform better using the side information compared to the situation in which we only use directly related information.

5.2 Future directions

The future directions can be divided into 3 modules: modeling, learning and inference, and applications. The high level of the modeling part is addressed in Chapter 4. However, more details need to be specified based on a specific application. Inference and learning is the next module, which is very important. No matter how perfect the model, lack of a strong inference and learning algorithm results in failure. Applications are the final module. A good model accompanied with a powerful

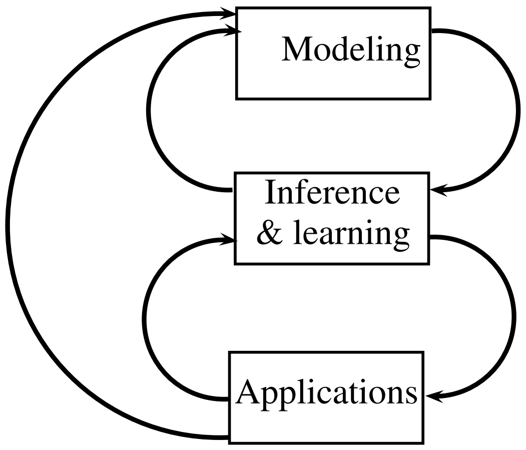


Figure 5.1: Three modules regarding the future directions for improving the latent feature network

inference engine will have no value if is not useful in real life.

Figure 5.1 represents 3 modules of the future directions. Modeling should be refined in order to design the inference and learning algorithm. Similarly, inference and learning is needed to apply the proposed ideas to a problem. However, this is not a one-way precedence. The lessons learned from designing the inference algorithm may encourage some changes in the model. For example, a repeated observed overfitting in some type of variables might be resolved with some modeling tricks. Applications have the potential for improving both the model and the inference algorithm. There may be some relationship types that we have not embedded in the model. Given an application including the special relationship, we can converge to a general model. Also, an inference algorithm may not be scalable enough for an application, demanding a faster but less accurate inference and learning strategy.

5.2.1 Modeling

The big picture of latent feature network is explained in the previous chapter. There are some decisions to make regarding the definition relation kernels $\mathcal{K}$ and entity dependency functions $\mathcal{E}$. For the relation kernel, we might use a dot product kernel or Euclidean distance kernel as discussed in Section 3.3. Dot product may be easier for learning while Euclidean distance can be used for visualization (Sections 3.1 and 3.3.3). However, if we assume Gaussian distributions as priors for some generative parts of the model, then squared Euclidean distance function mixes well with the Gaussian kernel. Therefore, an exploratory approach about which kernel helps more in what situation will be helpful. Also, we can try other kernels used by kernel-based approaches such as support vector machines.

The next function is the dependency function $\mathcal{E}$ that enforces correlation between latent feature variables of the same entity participating in different relationships. For the sake of simplicity, let $x_1, x_2, ..., x_n$ denote the latent feature variables of the same entity participating in n different relations. Agarwal et al. [1] called these latent variables, local variables and then they defined a global latent variable for the entity and assumed a Gaussian generative direction from a linear transformation of the global variable to the local variables. That is,

$$x_i \sim Gaussian(A_i z, \sigma_i^2), \tag{5.1}$$

where A_i is the linear transformation matrix to move from the global feature z to the local feature x_i but with introducing noise with the strength of variance σ_i^2. σ_i^2 can be learned from data. Given this generative model, the dependency function is

$$\mathcal{E}(x_1, ..., x_n) = -\sum_i \frac{1}{\sigma_i^2} ||x_i - A_i z||_2^2 \tag{5.2}$$

where z, A_i, and σ_i^2 are subsets of the parameter variables Θ.

Such an approach gives a weak dependency among the local latent variables. All local variables are conditionally independent given the global latent variable — the dependency among them flows through the global variable. As a result, it is hard to model the different degrees of dependency among different pairs of local latent variables. For example, suppose we want to perform collaborative filtering for the same set of users in three different contexts: movies, fiction books, and laptops. We expect that a stronger correlation exists between the latent variables of users in movies and fiction books context, compared to among the former and laptops. Therefore, a mutual dependence model captures the specific correlation between pairs of local latent variables. Such an approach resembles kernel-based approaches which incorporate closeness of data records in the feature space. In Section 4.3, the dependency function can be defined as:

$$\mathcal{E}(x_1, ..., x_n) = -\begin{pmatrix} x_1 & x_2 & ... & x_n \end{pmatrix} \Lambda_x \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{pmatrix} \tag{5.3}$$

where the relationship between local latent variables can be learned directly.

Another decision is about the priors over latent feature variables. While the Gaussian distribution enjoys some attractive traits such as straightforward inference, assuming exponential families as priors gives a more generalized capability. The choice of priors depends on the assumption on latent feature variables. Categorical latent variables — e.g. co-clustering in Section 3.2 and latent Dirichlet allocation (LDA) [11] — require prior distributions such as the Dirichlet distribution.

In our completed work, only dyadic relations are considered. However, there are some types of relations that exist among 3 different types of entities and breaking them into dyadic relationships may not be wise. Incorporating an n-ary relation requires an operator function with more than two operands — a generalized version of Euclidean distance or dot-product.

In many relational data, there is a relational structure that is essentially different from the relation between two entities. One example is the structure posed by time. While time can be considered as an entity, the dynamic and linear nature of time has some intrinsic characteristics that make it a different type of structure. Another example is sequential data, such as texts in which each word has a preceding and a succeeding word. Therefore, some properties such as part-of-speech of each word depends on the properties of its neighbors. The question is how we can incorporate these structures into the latent feature network model.

Depending on the goal of each relational component in the latent variable network, a relation might be learned in a discriminative or non-discriminative way. A discriminative approach is of interest if predicting the relation is the main goal. For example, in the gene expression microarray classification (Section 4.2), a generative — non-discriminative — approach was used for learning the expression level while a discriminative strategy was employed for the categorization problem. The goal is finding strategies that give high discriminative power when needed and efficient learning capability when discrimination is not required.

Finally, we are interested in addressing the updating of an existing constructed model for a problem when some new relationships or entities are available. The component-wise design of latent feature network facilitates adding a relationship by only assuming a new local latent variable for the new relationship and learning the correlation between the new local latent variable and other peers. Adding a new entity type to the system is straightforward since it means no relations exist in the system for this entity.

In summary the directions for the research regarding modeling are

1. Choice of relationship Kernel function and entities dependency function
2. Choice of priors
3. Modeling n-ary relationships
4. Modeling structural relationships
5. Modeling discriminative learning
6. Updating a system based on the model.

5.2.2 Inference and learning

Exact inference and learning are intractable when there are loops in the graphical model. In latent feature networks there are a lot of loops even if we only consider one relation. Furthermore, using continuous latent feature variables, computing some integrals is intractable. Moreover, we are interested in fully Bayesian treatment of latent random variables and parameter variable to give robust results.

Two Bayesian methods — widely used in the graphical models context — for approximate inference are variational methods and Markov chain Monte Carlo (see Section 2.1.2). Variation methods define an optimization problem over inference where the difference between a surrogate probability distribution Q and true posterior probability distribution P over latent variables is minimized. Based on the definition of latent variable models, such a problem is non-convex. Let $P = \frac{1}{Z}\exp(X_i X_j^T Y_{ij})$ denote a simple relationship between two types of entities indexed by i and j and with the observed relation random variable Y_{ij} and T denote transpose. This function is non-concave, but assuming X_i is constant, we retrieve the exponential family distribution which is concave in the parameters although still non-linear. So far, we used the simple gradient ascent algorithm in the completed work but some more advanced numerical optimization methods based on convex analysis have a lot of potential.

Markov chain Monte Carlo algorithms sample the distribution space and estimate the needed values via averaging over samples. The mechanism of MCMC algorithms is similar to stochastic heuristic search methods. Although heuristic search methods mostly have been used in the context of combinatorial optimization problems, some of them such as simulated annealing has been successfully applied to continuous problems. Heuristic search ideas have been successfully applied to MCMC algorithm for guiding the sampling more intelligently. On one hand, we want to sample from regions with high probability and on the other hand, we do not want to become stuck in one mode of the probability distribution — a local optimum. An algorithm such as tabu search may be used if we have enough samples of one region and we do not want to sample from that region anymore. Furthermore, some unobserved variables are still discrete in the latent feature networks which result in mixed continuous-discrete problem that suggests using heuristic optimization methods even more.

Finally, it is possible to combine variation methods and MCMC. MCMC needs a proposal function which shows the direction to move in the probability solution space. Via variation methods, we can guide the proposal distribution towards the true posterior regions.

In summary the directions for the research regarding inference and learning are

1. using convex analysis for better optimizing the variational objective function
2. using heuristic ideas in Markov chain Monte Carlo
3. combining MCMC and variational inference

5.2.3 Applications

Many machine learning problems can fit into the relational learning framework. The classic transactional flat data can be considered as relational data as well where each attribute or feature is considered as a relation. Applying latent feature networks to transactional data is expected to give properties such as robustness to noise (probabilistic model for each feature), automatic feature selection (no feature influences classification directly), and robustness to missing values (only some observed features are enough to recover the latent feature of a sample). Note that the microarray classification problem is a flat transactional classification application and a relational learning model proposed in Section 4.2 works well compared to transactional learning approaches such as support vector machines.

Among those, the following problems seem interesting:

1. collaborative filtering in different contexts with side information [1, 96]: The recommendation system problem but given different types of products and side information regarding users and items.
2. Text categorization [110, 74], text data modeling and document categorization simultaneously
3. Marketing data such as customer targeting [65] where transferring knowledge between product domains is helpful [109].
4. applications that are used for other models such as rule-based or frame-based relational learning approaches.

REFERENCES

[1] D. Agarwal, B.C. Chen, and B. Long. Localized factor models for multi-context recommendation. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 609–617. ACM, 2011.

[2] J.H. Albert and S. Chib. Bayesian analysis of binary and polychotomous response data. *Journal of the American Statistical Association*, pages 669–679, 1993.

[3] B. Anderson and J.B. Moore. *Optimal Filtering*, volume 1. Prentice-Hall Information and System Sciences Series, Englewood Cliffs: Prentice-Hall, 1979.

[4] C. Andrieu, N. De Freitas, A. Doucet, and M.I. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50(1):5–43, 2003.

[5] A. Banerjee, I. Dhillon, J. Ghosh, S. Merugu, and D.S. Modha. A generalized maximum entropy approach to Bregman co-clustering and matrix approximation. *The Journal of Machine Learning Research*, 8:1919–1986, 2007.

[6] W. Basalaj. Incremental multidimensional scaling method for database visualization. In *Proc. Visual Data Exploration and Analysis VI, SPIE*, volume 3643, pages 149–158, 1999.

[7] D. Billsus and M.J. Pazzani. Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning*, volume 46-53, page 48, 1998.

[8] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[9] D. Blei and J. Lafferty. Correlated topic models. *Advances in Neural Information Processing Systems*, 18:147, 2006.

[10] David M. Blei and John D. Lafferty. Dynamic topic models. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 113–120, New York, NY, USA, 2006. ACM.

[11] D.M. Blei, A.Y. Ng, and M.I. Jordan. Latent Dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[12] H. Blockeel. Statistical relational learning. *Handbook of Neural Information Processing*, 2011.

[13] I. Borg and P.J.F. Groenen. *Modern Multidimensional Scaling.* Springer New York, 1997.

[14] G. Bouchard. Efficient bounds for the softmax function, applications to inference in hybrid models. *Advances in Neural Information Processing Systems*, 2007.

[15] M. Brand. Fast online SVD revisions for lightweight recommender systems. In *SIAM International Conference on Data Mining*, pages 37–46, 2003.

[16] J.P. Brunet, P. Tamayo, T.R. Golub, and J.P. Mesirov. Metagenes and molecular pattern discovery using matrix factorization. *Proceedings of the National Academy of Sciences of the United States of America*, 101(12):4164, 2004.

[17] L. Candillier, F. Meyer, and M. Boulle. Comparing state-of-the-art collaborative filtering systems. *Lectures Notes in Computer Science*, 4571:548, 2007.

[18] P. Carbonetto. *New Probabilistic Inference Algorithms that Harness the Strengths of Variational and Monte Carlo Methods.* PhD thesis, University of British Columbia, Department of Computer Science, 2009.

[19] P. Carbonetto and N. De Freitas. Conditional mean field. *Advances in Neural Information Processing Systems*, 19:201, 2007.

[20] P. Carbonetto, M. King, and F. Hamze. A stochastic approximation method for inference in probabilistic graphical models. In *Advances in Neural Information Processing Systems*, volume 22, pages 216–224, 2009.

[21] Matthew Chalmers. A linear iteration time layout algorithm for visualising high-dimensional data. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 127–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[22] C.C. Chang and C.J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):2–27, 2011.

[23] Robert G. Cowell, A. Philip Dawid, Steffen L. Lauritzen, and David J. Spiegelhalter. *Probabilistic Networks and Expert Systems: Exact Computational Methods for Bayesian Networks.* Springer Publishing Company, Incorporated, 1st edition, 2007.

[24] M.A.A. Cox and T.F. Cox. Multidimensional scaling. *Handbook of Data Visualization*, pages 315–347, 2008.

[25] T.F. Cox and M.A.A. Cox. *Multidimensional Scaling.* CRC Press, 2001.

[26] T. Damoulas and M.A. Girolami. Combining information with a Bayesian multi-class multi-kernel pattern recognition machine. *World Scientific Review*, 2008.

[27] A.S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: Scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web*, pages 271–280. ACM New York, NY, USA, 2007.

[28] J. Dauwels. On variational message passing on factor graphs. In *Information Theory, 2007. ISIT 2007. IEEE International Symposium on*, pages 2546–2550. IEEE, 2007.

[29] N. De Freitas, S. Russell, et al. Variational MCMC. In *Uncertainty in artificial intelligence*, pages 120–127, 2001.

[30] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for Information Science*, 41(6):391–407, 1990.

[31] W.S. DeSarbo, D.J. Howard, and K. Jedidi. Multiclus: A new method for simultaneously performing multidimensional scaling and cluster analysis. *Psychometrika*, 56(1):121–136, 1991.

[32] L.P. Dringus and T. Ellis. Using data mining as a strategy for assessing asynchronous discussion forums. *Computers & Education*, 45(1):141–160, 2005.

[33] D. Fisher, K. Hildrum, J. Hong, M. Newman, M. Thomas, and R. Vuduc. Swami: A framework for collaborative filtering algorithm development and evaluation. In *SIGIR 2000*, 2000.

[34] I.K. Fodor. A survey of dimension reduction techniques. Technical report, Lawrence Livermore National Laboratory,https://computation.llnl.gov/casc/sapphire/pubs /148494.pdf, 2002.

[35] F. Forbes and G. Fort. Combining Monte Carlo and mean-field-like methods for inference in hidden Markov random fields. *Image Processing, IEEE Transactions on*, 16(3):824–837, 2007.

[36] B.J. Frey. Extending factor graphs so as to unify directed and undirected graphical models. In *Proc. 19th Conf. Uncertainty in Artificial Intelligence*, pages 257–264, 2003.

[37] B.J. Frey, F.R. Kschischang, H.A. Loeliger, and N. Wiberg. Factor graphs and algorithms. In *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, volume 35, pages 666–680, 1997.

[38] T. George and S. Merugu. A scalable collaborative filtering framework based on co-clustering. In *Proceedings of the IEEE Conference on Data Mining*, pages 625–628, 2005.

[39] L. Getoor and J. Grant. PRL: A probabilistic relational language. *Machine Learning*, 62(1):7–31, 2006.

[40] L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning*. The MIT Press, 2007.

[41] W.R. Gilks, S. Richardson, and D.J. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman & Hall/CRC, 1996.

[42] A. Globerson, G. Chechik, F. Pereira, and N. Tishby. Euclidean embedding of co-occurrence data. *Journal of Machine Learning Research*, 8:2265–2295, 2007.

[43] Amir Globerson and Naftali Tishby. Sufficient dimensionality reduction. *J. Mach. Learn. Res.*, 3:1307–1331, March 2003.

[44] T.L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences of the United States of America*, 101(Suppl 1):5228, 2004.

[45] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57, New York, NY, USA, 1984. ACM.

[46] Viet Ha-Thuc and Jean-Michel Renders. Large-scale hierarchical text classification without labelled data. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 685–694, New York, NY, USA, 2011. ACM.

[47] D. Heckerman. A tutorial on learning with Bayesian networks. *Innovations in Bayesian Networks*, pages 33–82, 2008.

[48] P.D. Hoff, A.E. Raftery, and M.S. Handcock. Latent space approaches to social network analysis. *Journal of the American Statistical Association*, 97(460):1090–1098, 2002.

[49] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, pages 50–57, New York, NY, USA, 1999. ACM.

[50] C.J. Hogger. *Essentials of Logic Programming*, volume 1. Oxford University Press, USA, 1990.

[51] F.J. Igo Jr, M. Brand, K. Wittenburg, D. Wong, and S. Azuma. Multidimensional visualization for collaborative filtering recommender systems. *Technical Report TR20003-39, Mitsubishi Electric Research Laboratories*, 2002.

[52] Finn V. Jensen. *Introduction to Bayesian Networks.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1996.

[53] F.V. Jensen and T.D. Nielsen. *Bayesian Networks and Decision Graphs.* Springer Verlag, 2007.

[54] M.I. Jordan. *Learning in Graphical Models.* Kluwer Academic Publishers, 1998.

[55] M.I. Jordan. Graphical models. *Statistical Science*, pages 140–155, 2004.

[56] M.I. Jordan, Z. Ghahramani, T.S. Jaakkola, and L.K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

[57] M. Khoshneshin, M.A. Basir, P. Srinivasan, W. N. Street, and B. Hand. Analyzing the language evolution of a science classroom via a topic model. In *KDD-2011 Workshop on Knowledge Discovery in Educational Data, San Diego, CA, August 2011*, 2011.

[58] M. Khoshneshin, M. Ghazizadeh, W. N. Street, and J.W. Ohlmann. A memetic heuristic for the co-clustering problem. In *Bionetics*, 2010.

[59] M. Khoshneshin and W. N. Street. Incremental collaborative filtering via evolutionary co-clustering. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 325–328, New York, NY, USA, 2010. ACM.

[60] M. Khoshneshin and W. N. Street. A latent feature factor graph for cancer diagnosis using microarray gene expression data. In *Sixth INFORMS Workshop on Data Mining and Health Informatics*, 2011.

[61] M. Khoshneshin, W. N. Street, and P. Srinivasan. Latent logistic allocation: A kernel-based approach. *Under review.*

[62] Mohammad Khoshneshin and W. N. Street. Collaborative filtering via Euclidean embedding. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 87–94, New York, NY, USA, 2010. ACM.

[63] Mohammad Khoshneshin, W. N. Street, and Padmini Srinivasan. Bayesian embedding of co-occurrence data for query-based visualization. In *Proceedings of IEEE International Conference on Machine Learning and Applications*, 2011.

[64] H. Khosravi and B. Bina. A survey on statistical relational learning. *Advances in Artificial Intelligence*, pages 256–268, 2010.

[65] Y.S. Kim, W.N. Street, G.J. Russell, and F. Menczer. Customer targeting: A neural network approach guided by genetic algorithms. *Management Science*, pages 264–276, 2005.

[66] R. Kindermann, J.L. Snell, and American Mathematical Society. *Markov random fields and their applications.* American Mathematical Society Providence, RI, 1980.

[67] A. Knobbe, M. De Haas, and A. Siebes. Propositionalisation and aggregates. *Principles of Data Mining and Knowledge Discovery*, pages 277–288, 2001.

[68] D. Koller and N. Friedman. *Probabilistic Graphical Models.* MIT press, 2009.

[69] Yehuda Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 426–434, New York, NY, USA, 2008. ACM.

[70] S. Kramer, N. Lavrac, and P. Flach. Propositionalization approaches to relational data mining. *Relational data mining*, page 262, 2001.

[71] M.A. Krogel, S. Rawles, F. Železný, P. Flach, N. Lavrač, and S. Wrobel. Comparative evaluation of approaches to propositionalization. *Inductive Logic Programming*, pages 197–214, 2003.

[72] J.B. Kruskal. Nonmetric multidimensional scaling: A numerical method. *Psychometrika*, 29(2):115–129, 1964.

[73] F.R. Kschischang, B.J. Frey, and H.A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

[74] S. Lacoste-Julien, F. Sha, and M.I. Jordan. Discriminative learning for dimensionality reduction and classification. In *Advances in Neural Information Processing Systems*, volume 21, 2008.

[75] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning*, pages 282–289, 2001.

[76] S.L. Lauritzen. *Graphical models.* Oxford University Press, USA, 1996.

[77] F. Liang, C. Liu, and R. Carroll. *Advanced Markov chain Monte Carlo methods: Learning from past samples.* Wiley, 2010.

[78] H.A. Loeliger. An introduction to factor graphs. *Signal Processing Magazine, IEEE*, 21(1):28–41, 2004.

[79] A. McCallum, K. Schultz, and S. Singh. Factorie: Probabilistic programming via imperatively defined factor graphs. In *Neural Information Processing Systems Conference (NIPS)*, 2009.

[80] A. Morrison, G. Ross, and M. Chalmers. Fast multidimensional scaling through sampling, springs and interpolation. *Information Visualization*, 2(1):68–77, 2003.

[81] S. Muggleton. *Inductive Logic Programming.* Morgan Kaufmann, 1992.

[82] K.R. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201, 2001.

[83] R.M. Neal. Probabilistic inference using Markov chain Monte Carlo Methods. Technical report, University of Toronto, Department of Computer Science, 1993.

[84] J. Neville and D. Jensen. Relational dependency networks. *The Journal of Machine Learning Research*, 8:653–692, 2007.

[85] K. Nigam, A.K. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2):103–134, 2000.

[86] M. Opper and D. Saad. *Advanced Mean Field Methods: Theory and Practice.* Massachusetts Institute of Technology Press (MIT Press), 2001.

[87] M. Papagelis, I. Rousidis, D. Plexousakis, and E. Theoharopoulos. Incremental collaborative filtering for highly-scalable recommendation algorithms. In *Proc. of International Symposium on Methodologies of Intelligent Systems*, pages 553–561. Springer, 2005.

[88] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *KDD 2007: Netflix Competition Workshop.*

[89] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[90] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1):107–136, 2006.

[91] C. Romero and S. Ventura. Educational data mining: A review of the state of the art. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 40(6):601–618, 2010.

[92] Michal Rosen-Zvi, Thomas Griffiths, Mark Steyvers, and Padhraic Smyth. The author-topic model for authors and documents. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI '04, pages 487–494, Arlington, Virginia, United States, 2004. AUAI Press.

[93] I.U.A. Rozanov. *Markov Random Fields.* Springer Verlag, 1982.

[94] H. Rue and L. Held. *Gaussian Markov Random Fields: Theory and Applications*, volume 104. Chapman & Hall, 2005.

[95] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Fifth International Conference on Computer and Information Science*, pages 27–28, 2002.

[96] Ajit P. Singh and Geoffrey J. Gordon. Relational learning via collective matrix factorization. In *Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 650–658, New York, NY, USA, 2008. ACM.

[97] A.P. Singh. *Efficient Matrix Models for Relational Learning.* PhD thesis, Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2009.

[98] Mark K. Singley and Richard B. Lam. The classroom sentinel: Supporting data-driven decision-making in the classroom. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 315–321, New York, NY, USA, 2005. ACM.

[99] A. Statnikov, C.F. Aliferis, I. Tsamardinos, D. Hardin, and S. Levy. A comprehensive evaluation of multicategory classification methods for microarray gene expression cancer diagnosis. *Bioinformatics*, 21(5):631, 2005.

[100] G. Takacs, I. Pilaszy, B. Nemeth, and D. Tikk. On the Gravity recommendation system. In *KDD 2007: Netflix Competition Workshop.*

[101] Michael W. Trosset, Carey E. Priebe, Youngser Park, and Michael I. Miller. Semisupervised learning from dissimilarity data. *Computational Statistics and Data Analysis*, 52(10):4643 – 4657, 2008.

[102] F. Valafar. Pattern recognition techniques in microarray data analysis. *Annals of the New York Academy of Sciences*, 980(1):41–64, 2002.

[103] M.J. Wainwright and M.I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[104] Xing Wei and W. Bruce Croft. LDA-based document models for ad-hoc retrieval. In *Proceedings of the 29th Annual International ACM Conference on Research and Development in Information Retrieval*, pages 178–185, New York, NY, USA, 2006. ACM.

[105] Y. Wexler and D Geiger. Importance sampling via variational optimization. *Uncertainty in Artificial Intelligence. Morgan Kaufmann*, 2007.

[106] John Winn and Christopher M. Bishop. Variational message passing. *J. Mach. Learn. Res.*, 6:661–694, December 2005.

[107] J.S. Yedidia, W.T. Freeman, and Y. Weiss. Generalized belief propagation. *Advances in Neural Information Processing Systems*, pages 689–695, 2001.

[108] J. Zhang. *Visualization for Information Retrieval.* Springer Verlag, 2008.

[109] Yi Zhang, Samuel Burer, and W. Nick Street. Ensemble pruning via semi-definite programming. *J. Mach. Learn. Res.*, 7:1315–1338, December 2006.

[110] Jun Zhu, Amr Ahmed, and Eric P. Xing. Medlda: Maximum margin supervised topic models for regression and classification. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1257–1264, New York, NY, USA, 2009. ACM.

[111] X. Zhu and Z. Ghahramani. Towards semi-supervised classification with Markov random fields. Technical report, CMU-CALD-02-106, Carnegie Mellon University, 2002.