NON-PARAMETRIC GRAPH-BASED METHODS FOR LARGE SCALE PROBLEMS

by

Saeed Amizadeh

BS, University of Tehran, 2004

MS, University of Tehran, 2007

MS, University of Pittsburgh, 2010

Submitted to the Graduate Faculty of

the Kenneth P. Dietrich School of

Arts and Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2013

UNIVERSITY OF PITTSBURGH

INTELLIGENT SYSTEMS PROGRAM

This dissertation was presented

by

Saeed Amizadeh

It was defended on

August 28, 2013

and approved by

Milos Hauskrecht, PhD, Associate Professor, Computer Science

Chakra Chennubhotla, PhD, Assistant Professor, Computational and Systems Biology

Greg Cooper, MD PhD, Professor, Biomedical Informatics

Marek Druzdzel, PhD, Associate Professor, School of Information Sciences

Rebecca Nugent, PhD, Associate Professor, Department of Statistics, Carnegie Mellon University

Shyam Visweswaran, MD PhD, Assistant Professor, Biomedical Informatics

Dissertation Director: **Milos Hauskrecht, PhD**, Associate Professor, Computer Science

NON-PARAMETRIC GRAPH-BASED METHODS FOR LARGE SCALE PROBLEMS

Saeed Amizadeh, PhD

University of Pittsburgh, 2013

The notion of similarity between observations plays a very fundamental role in many Machine Learning and Data Mining algorithms. In many of these methods, the fundamental problem of prediction, which is making assessments and/or inferences about the future observations from the past ones, boils down to how “similar” the future cases are to the already observed ones. However, similarity is not always obtained through the traditional distance metrics. Data-driven similarity metrics, in particular, come into play where the traditional absolute metrics are not sufficient for the task in hand due to special structure of the observed data. A common approach for computing data-driven similarity is to somehow aggregate the local absolute similarities (which are not data-driven and can be computed in a closed-from) to infer a global data-driven similarity value between any pair of observations. The graph-based methods offer a natural framework to do so. Incorporating these methods, many of the Machine Learning algorithms, that are designed to work with absolute distances, can be applied on those problems with data-driven distances. This makes graph-based methods very effective tools for many real-world problems.

In this thesis, the major problem that I want to address is the scalability of the graph-based methods. With the rise of large-scale, high-dimensional datasets in many real-world applications, many Machine Learning algorithms do not scale up well applying to these problems. The graph-based methods are no exception either. Both the large number of observations and the high dimensionality hurt graph-based methods, computationally and statistically. While the large number of observations imposes more of a computational problem, the high dimensionality problem has more of a statistical nature. In this thesis, I address both of these issues in depth and review the common solutions for them proposed in the literature. Moreover, for each of these problems, I propose novel solutions with experimental results depicting the merits of the proposed algorithms. Finally, I discuss the contribution of the proposed work from a broader viewpoint and draw some future directions of the current work.

TABLE OF CONTENTS

1.0 INTRODUCTION TO GRAPH-BASED METHODS 1
- 1.1 Motivation: The Big Picture 1
  - 1.1.1 Large Number of Observations 3
  - 1.1.2 High Dimensionality 4
- 1.2 Background 5
  - 1.2.1 The Basics 5
    - 1.2.1.1 The Similarity Graph 5
    - 1.2.1.2 The Laplacian of Graph 5
    - 1.2.1.3 Smoothness on Graphs 6
  - 1.2.2 The Laplacian Eigenmap 8
  - 1.2.3 The Diffusion Map 8
    - 1.2.3.1 Random Walk on Graph 9
    - 1.2.3.2 Random Walk and Kernel Density Estimation 10
    - 1.2.3.3 Multiscale Random Walk 11
    - 1.2.3.4 Diffusion Distance 12
    - 1.2.3.5 Diffusion Map 13
  - 1.2.4 Laplacian-based Kernels And Spectral Transform 13
  - 1.2.5 The Diffusion Operator 15
  - 1.2.6 Building The Similarity Graph 17
    - 1.2.6.1 Weighted Fully Connected 18
    - 1.2.6.2 k-nearest-neighbor (kNN) 18
    - 1.2.6.3 ϵ-neighborhood 19

1.2.6.4 Minimum Spanning Tree (MST) 20
1.2.6.5 Dealing With Non-vector Spaces 20
2.0 CASE STUDY: CONSTRUCTING GENERALIZED LOCALLY-INDUCED TEXT METRICS 22
2.1 Introduction 22
2.2 Related Work 24
2.3 The Graph-based Text Metric 26
2.3.1 Term-Term Distance Metrics 26
2.3.2 Set-Set Distance Metric 27
2.4 Experiments 30
2.4.1 Term Prediction 30
2.4.1.1 Data 30
2.4.1.2 Evaluation Metric 30
2.4.1.3 Baselines 31
2.4.1.4 Results 31
2.4.2 Query Expansion 32
2.4.2.1 Datasets 33
2.4.2.2 Experimental setup 33
2.4.2.3 Results 34
2.5 Discussion 34
3.0 LARGE N 37
3.1 Introduction 37
3.2 Related Work 39
3.2.1 Node Sparsification Methods 39
3.2.2 Edge Sparsification Methods 40
3.2.3 Decomposition-based Methods 41
3.2.4 Direct Methods 41
3.2.5 Hierarchical Methods 42
3.3 Case Study: Subsampling in The Term Space 43
3.3.1 Building The Association Graph 44

3.3.2 Eigen Decomposition of The Association Graph 44
3.3.3 Empirical Error for Ranking 45
3.4 Large-scale Transition Matrix Approximation 45
3.4.1 The Problem Statement 49
3.5 Fast Kernel Density Estimation 50
3.5.1 Single-tree Approximation 51
3.5.2 Dual-tree Approximation 52
3.5.2.1 Variational Log-likelihood 55
3.5.2.2 Dual-tree Block Partitioning 56
3.5.2.3 Variational Optimization 59
3.5.3 Anchor Trees 61
3.5.3.1 Anchor Tree Construction Algorithm 61
3.5.3.2 Time Complexity 63
3.6 Variational Dual-Tree Transition Matrix Approximation 65
3.6.1 Variational Random Walk 65
3.6.2 Optimizing The Bandwidth 67
3.6.3 Fast Inference 68
3.6.4 Partitioning and Refinement 70
3.7 Bregman Variational Dual-Tree Framework 74
3.7.1 The Bregman Divergences and The Exponential Families 75
3.7.2 Bregman Variational Approximation 79
3.7.3 Bregman Anchor Trees 81
3.8 Experiments 83
3.8.1 Methods 83
3.8.2 Efficiency and Quality vs. Problem Size 85
3.8.2.1 Data 85
3.8.2.2 Experimental Setup 85
3.8.2.3 Results 86
3.8.3 The Effect of Refinement 92
3.8.3.1 Experimental Setup 92

3.8.3.2 Results 93
3.8.4 Computational Scalability 98
3.8.5 BVDT For Frequency Data 98
3.8.5.1 The Bregman Divergence For Frequency Data 99
3.8.5.2 Simulation 102
3.8.5.3 Text Data 108
3.9 Discussion 114
4.0 LARGE *D* 117
4.1 Introduction 117
4.2 The Factorized Diffusion Approximation 119
4.2.1 Motivation 119
4.2.2 Related Work 121
4.2.3 Convergence of Diffusion Operator 122
4.2.4 Factorized Diffusion Maps 124
4.2.4.1 The Factorized Approximation 124
4.2.4.2 Error Analysis 126
4.2.4.3 Finding The Best Partition 131
4.2.5 Experimental Results 135
4.2.5.1 Synthetic Data 135
4.2.5.2 Image Data 136
4.2.5.3 SNP Data 138
4.2.6 Discussion 143
4.3 Infinite Dimension: Dealing With Functional Spaces 143
4.3.1 Motivation 143
4.3.2 Vector Representation 146
4.3.3 Large-Scale Laplacian Eigenmap 147
4.3.4 Results and Discussion 149
5.0 CONCLUSIONS AND FUTURE DIRECTIONS 151
5.1 Contributions 151
5.2 Open Questions 156

5.3 Future Directions 158

BIBLIOGRAPHY 161

LIST OF TABLES

1 Different Spectral Transforms 15
2 AUC for The Term Prediction Task 32
3 The TREC Datasets 33
4 The Results for the Document Retrieval Task 34
5 The Average Number of Misplacements 45
6 Famous Bregman Divergences 78
7 Theoretical Complexity Analysis Results 84
8 The Construction Time for the MAGIC Gamma Telescope Dataset 87
9 The Propagation Time for the MAGIC Gamma Telescope Dataset 87
10 The AUC for the MAGIC Gamma Telescope Dataset 88
11 The Construction Time for the MiniBooNE Dataset 88
12 The Propagation Time for the MiniBooNE Dataset 89
13 The AUC for the MiniBooNE Dataset 89
14 The AUC During Refinement for the MAGIC Gamma Telescope Dataset 94
15 The AUC During Refinement for the MiniBooNE Dataset 95
16 Very Large-Scale Results 98
17 The Construction Time for the Simulated Frequency Data 105
18 The Propagation Time for the Simulated Frequency Data 106
19 The Accuracy for the Simulated Frequency Data 106
20 Text Datasets 109
21 The Accuracy vs. Labeled Data Ratio for the BBC Sport News Dataset 109
22 The Accuracy vs. Labeled Data Ratio for the BBC News Dataset 110

23 The Accuracy vs. Labeled Data Ratio for the 20 Newsgroup Dataset 110
24 The Accuracy vs. Labeled Data Ratio for the NSF Research Abstracts Dataset 111
25 The Accuracy vs. Labeled Data Ratio for the Large Movie Reviews Dataset . . 111
26 Results of Greedy Partitioning 136

LIST OF FIGURES

1 The Bullseye dataset 6
2 Electrical resistance network 28
3 Single-tree approximation 53
4 Dual-tree approximation 54
5 Block Partitioning 58
6 Refinement Process 73
7 The efficiency and quality results for the MAGIC Gamma Telescope dataset . 90
8 The efficiency and quality results for the MiniBooNE dataset 91
9 The refinement results for the MAGIC Gamma Telescope dataset 96
10 The refinement results for the MiniBooNE dataset 97
11 The document generation model 100
12 The term frequncies for the five simulated document topic 104
13 The efficiency and quality results for the simulated frequency data 107
14 The efficiency and quality results for the text datasets 113
15 The synthetic 3D dataset 121
16 Clustering Results 136
17 Speactral clustering of the SNP data 142
18 The human brain connectome 145
19 The embedding of brain fibers 150

LIST OF ALGORITHMS

1 Anchor Tree Construction 63
2 Anchor Building 64
3 Datapoint Stealing 64
4 Agglomeration 65
5 Calculate $Z = \mathbf{Q}Y$ 70
6 Collect-up 70
7 Distribute-down 71
8 Symmetric Refinement 74

9 Greedy Partitioning 132
10 Approximate Large-scale Laplacian Eigen Decomposition 148

PREFACE

During my Ph.D, I have received support from a number of people and organizations without whom the completion of this thesis would not be possible.

First and foremost, I would like to express my deepest gratitude to my loving parents Maryam and Ebrahim and my brothers Vahid and Farid for their unlimited, unconditional support, encouragement and love during my whole life.

I would like to express my sincere gratitude to my Ph.D. advisor, Dr. Milos Hauskrecht, who provided me with the best advice and directions during my Ph.D. career. I would also like to thank my thesis committee, Dr. Chakra Chennubhotla, Dr. Greg Cooper, Dr. Marek Druzdzel, Dr. Rebecca Nugent and Dr. Shyam Visweswaran for their valuable feedback and discussions during my thesis defense.

The different parts of this thesis have been done in direct collaboration with individuals inside and outside of University of Pittsburgh. In particular, I would like to thank Dr. Bo Thiesson from Microsoft Research (currently at Aalborg University), Dr. Denver Dash and Dr. Mei Chen both from Intel Labs Pittsburgh, Dr. Walt Schneider, Dr. Hamed Valizadegan (currently at NASA Research) and Shuguang Wang from University of Pittsburgh for their direct contributions in this thesis. Also I want to thank my other fellow Ph.D. students and colleagues Dr. Iyad Batal, Dr. Michal Valko, Quang Nguyen, Charmgill Hong, Eric Heim, Zitao Liu, Matthias Grabmair, Ian Wong, Yuriy Sverchkov and Mahdi Pakdaman for their valuable feedback and discussions for this work.

This work would not be completed without the financial support from different institutes and organizations. In particular, I am grateful of the grants from the National Institutes of Health (NIH) as well as the Andrew Mellon Predoctoral Fellowship and two Arts and Sciences (A&S) Fellowships from the Dietrich School of Arts and Sciences, University of Pittsburgh.

Finally, I would like to thank all of my friends in the Argentine Tango community of Pittsburgh who have shared the love of dance and music with me and helped me to improve the quality of my life, beside my studies.

I am deeply grateful to all of you!

1.0 INTRODUCTION TO GRAPH-BASED METHODS

1.1 MOTIVATION: THE BIG PICTURE

The notion of *similarity/distance* between observations plays a very fundamental role in many Machine Learning and Data Mining methodologies. In many of these methods, the fundamental problem of prediction, which is making assessments and/or inferences about the future observations from the past ones, boils down to how *"similar"* the future cases are to the already observed ones. Although in many problems, the similarity between observations can be quantified as an absolute number, independent of the *context* of problem, one might be interested to modify the similarity value based on the context. For example, consider the problem of developing a system that automatically clusters the students in a classroom into groups with similar English accents for some linguistic purpose. Now imagine two American students with native American accent: in a classroom of mostly international students, these two students' accents are considered pretty similar and therefore we expect the system to put them in the same group. However, in a classroom of all American students, the accents of the same two students might be considered quite different depending on which regions in the U.S. they are from. Clearly, in this example, the similarity between the students' accents depends on the context of the classroom. This context is, in fact, the observed population containing the two observations in question. This gives rise to the notion of *data-driven* similarity metrics where the distance (or similarity) between two observations depends not only on those observations but also on other observations in the sample. By definition, the data-driven similarity metrics are *non-parametric* that are inferred from a given dataset of observations in the problem. Therefore, given two observations, we may not be able to compute their data-driven similarity using a closed-form formula.

A common solution to this problem is to somehow *aggregate* the *local* absolute similarities (which are not data-driven and can be computed in a closed-from) to infer a *global* data-driven similarity value between any pair of observations. The *graph-based* methods offer a natural framework to do so. In particular, using these methods, one creates a graph that contains the local similarities between the observations (referred to as the *similarity graph*), and then by using the spectral decomposition of this graph, aggregates these local similarities to compute a global similarity metric (kernel) between the observations. A direct consequence is that many of the Machine Learning algorithms that are designed to work with absolute distances now can be applied on those problems with data-driven distances. Probably, clustering is the most famous platform in Machine Learning that has been extended by the notion of data-driven distance resulting in *spectral clustering* frameworks [von Luxburg, 2007, Nadler et al., 2006, Ng et al., 2001b, von Luxburg et al., 2008, Bach and Jordan, 2004, Jin et al., 2006, Ng et al., 2001a, Zelnik-Manor and Perona, 2005, Maiker et al., 2009, Carreira-Perpinán and Zemel, 2005, Yan et al., 2009, Wang and Dong, 2012]. The other field of Machine Learning that has been largely affected by the use of graph-based methods is dimensionality reduction and embedding [Lafon and Lee, 2006, Belkin and Niyogi, 2002, Zhang et al., 2012, Losada and Barreiro, 2003, He et al., 2005, Tenenbaum et al., 2000, Roweis and Saul, 2000, Yan et al., 2007, Ham et al., 2004]. Graph-based techniques have been also extensively used for semi-supervised learning [Chapelle et al., 2006, Jebara et al., 2009, Fergus et al., 2009, Zhu and Lafferty, 2005, Zhu, 2005b, Zhu, 2005a, Garcke and Griebel, 2005, Zhu et al., 2006, Zhu et al., 2003, Subramanya and Bilmes, 2009, Valko et al., 2012, Liu et al., 2010].

Therefore, graph-based methods can be seen as very effective tools for many real-world problems. Because of this very reason, I have chosen these methods as the main direction of my PhD thesis work. In this chapter, I review the major algorithms and frameworks developed in the literature for graph-based methods. In Chapter 2 as a case study, I show a very successful application of these methods to compute the similarity between words with two real-world applications in document retrieval and query expansion. As the experiments show, the graph-based methods outperform many of the popular baseline methods in these domains.

Nevertheless, the major problem that I want to address in my thesis is the *scalability* of the graph-based methods. Nowadays, in nearly all fields of science and engineering there exist huge datasets which need to be analyzed. These datasets are normally a collection of thousands of records (or observations) each of which may consist of thousands of numerical values (or *features*) describing each case. Examples of such datasets in Biomedical Sciences are gene-expression data where each case is the expression levels for thousands of genes for an individual, or electronic health records of patients where each patient case may include lab tests, medications, etc. during the hospitalization period. Unfortunately, many existing standard Machine Learning techniques do not often scale-up if they are applied to complex large-scale datasets (i.e. datasets with either large number of records or large number of features or both). Not surprisingly, the graph-based methods are no exception either. Both the large number of observations and the large number of features (aka *high dimensionality*) hurt graph-based methods, computationally and statistically. While the large number of observations imposes more of a *computational* problem, the high dimensionality problem has more of a *statistical* nature.

1.1.1 Large Number of Observations

At the core of the graph-based methods is the similarity graph that captures the local absolute similarities among the observations. Depending on the characteristics of the input space, the local similarity metric and the representation of the observations, the similarity graph can be generated in numerous ways. Nevertheless, no matter how the similarity graph is generated, a common representation to store it and operate on it is in the form similarity matrix which in general takes $O(N^2)$ time and memory to maintain, where N is the number of observations or nodes in the graph. With N in the order of 10^4 or larger, we face a practical challenge as the demand for CPU and memory rapidly grows compared to the computational power of many non-cluster machines. Even if we can somehow compute and represent the similarity matrix, almost all operations on this matrix require visiting each element at least once, which means that they have at least $O(N^2)$ computational order. To address this problem, in Chapter 3, after reviewing the existing solutions in the literature, I propose a variational dual-tree based framework that can effectively reduce the order of computations to $O(N \log N)$. I show the successful application of this approach for large-scale problems.

1.1.2 High Dimensionality

High dimensionality also imposes a big challenge before the graph-based methods. More specifically, assuming the observations are randomly sampled from some *true* population, the structure represented by the finite sample in high dimension may not be the true representative of the population's structure. One way to see this intuitively is as follows: as the number of features grows the local (Euclidean) distances in the input space become larger and as a result the difference between them becomes relatively less significant. This means that with the fixed number of observations, as we increase the dimension, we start losing the *true* (manifold) intrinsic structure in the data. Therefore, if one is interested to estimate the quantities of the true population structure (e.g. the clusters in spectral clustering), one needs to (exponentially) increase the number of observations as the dimensionality increases, and this points to the well-known *curse of dimensionality* problem. To address this issue, in Chapter 4, I propose a divide-and-conquer approach which tries to exploit the independence structure of the input space to overcome the curse of dimensionality. I provide the error analysis for the proposed framework as well as some experiments showing the effectiveness of the proposed algorithm. Moreover, in Chapter 4, I demonstrate how graph-based methods can deal with those problems with functional input space with infinite dimensions in practice. I show a successful application of the proposed technique for the analysis of human brain fibers.

Finally, in Chapter 5, I discuss the contribution of the proposed work from a broader viewpoint and draw some future directions of the current work.

1.2 BACKGROUND

In this section, the required background materials for the proposed methods in this thesis as well as the most popular graph-based methods in the literature are presented.

1.2.1 The Basics

1.2.1.1 The Similarity Graph Let $\mathscr{D} = \{x_1, x_2, \ldots, x_N\}$ represent a set of N objects (aka observations, datapoints, records, cases, etc.). For some of the methods presented in this proposal, we assume the members of $\mathscr{D}$ are real vectors in $\mathbb{R}^d$, but in general, $\mathscr{D}$ can represent any set of objects. The *similarity graph* $G = \langle \mathscr{D}, E, W \rangle$ (aka the *association graph*) is a weighted graph with the members of $\mathscr{D}$ as its nodes, the edge set E, and $W = [w_{ij}]_{N \times N}$ as its weighted adjacency matrix, where w_{ij} is non-negative and represents the similarity weight between x_i and x_j for all $i, j \in \{1, \ldots, N\}$. Intuitively, the higher w_{ij}, the more similar (as defined in the context problem) x_i and x_j are. $w_{ij} = 0$ means there is no edge between x_i and x_j in G. W is also called the *similarity matrix*. Figure 1A shows the benchmark Bullseye dataset which has three circular shape clusters (shown in different colors). Figure 1B depicts the similarity graph on the Bullseye dataset constructed using k-nearest-neighbor method with $k = 5$. For now, we assume the similarity graph is given; how one can build the similarity graph from $\mathscr{D}$ is postponed to Section 1.2.6.

1.2.1.2 The Laplacian of Graph Let $D = [d_{ij}]_{N \times N}$ be a diagonal matrix with diagonal entries $d_{ii} = \sum_{j=1}^{N} w_{ij}$. The *Laplacian* matrices of the similarity graph G are defined as:

- The *unnormalized Laplacian*: $L = D - W$
- The *symmetric normalized Laplacian*: $L_{sym} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2}$
- The *asymmetric normalized Laplacian* (aka the *random walk Laplacian*): $L_{rw} = D^{-1} L = I - D^{-1} W$

The following basic properties hold for the Laplacian matrices (interested readers may refer to [Chung, 1997, von Luxburg, 2007] for more detailed properties and the proofs):

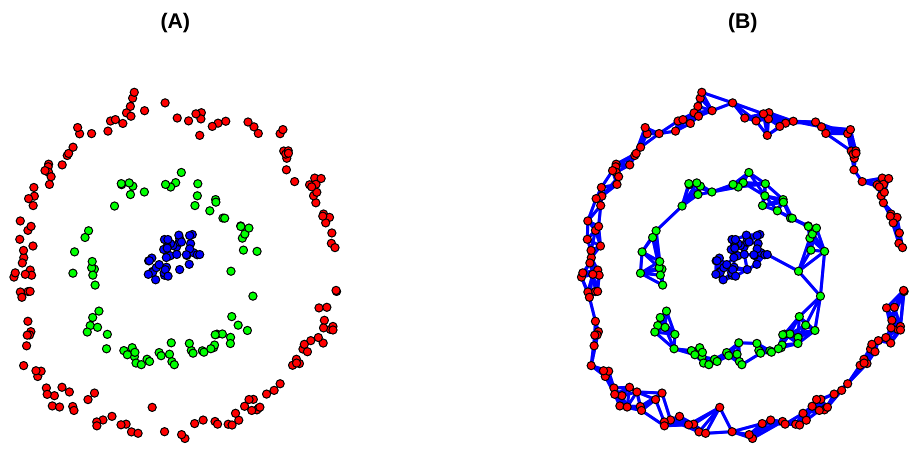

Figure 1: (A) The Bullseye dataset consisting of three circular clusters (B) The similarity graph constructed using kNN method with $k = 5$.

(a) L, L_{sym} and L_{rw} are all positive semi-definite matrices with the smallest eigenvalue equal to 0; that is, $0 = \mu_1 \leq \mu_2 \leq \ldots \leq \mu_N$

(b) The corresponding eigenvector to the smallest the eigenvalue $\mu_1 = 0$ is the constant one vector $\mathbb{1}$ for L and L_{rw} and $D^{1/2}\mathbb{1}$ for L_{sym}

(c) μ is an eigenvalue of L_{rw} with eigenvector u iff μ is an eigenvalue of L_{sym} with eigenvector $D^{1/2}u$

(d) μ is an eigenvalue of L_{rw} with eigenvector u iff μ and u solve the generalized eigen problem $Lu = \mu Du$

(e) The similarity graph G consists of K connected components $G_1, \ldots, G_K$ iff the multiplicity of the eigenvalue $\mu_1 = 0$ of L, L_{sym} and L_{rw} is equal to K. In this case, for L and L_{rw}, the eigenspace of μ_1 is spanned by the indicator vectors $\mathbb{1}_{G_1}, \ldots, \mathbb{1}_{G_K}$. For L_{sym}, this eigenspace is spanned by the vectors $D^{1/2}\mathbb{1}_{G_1}, \ldots, D^{1/2}\mathbb{1}_{G_K}$

1.2.1.3 Smoothness on Graphs Let $f : \mathscr{D} \mapsto \mathbb{R}^m$ be a vector-valued function that maps the elements of $\mathscr{D}$ to the m-dimensional real vectors. Given the similarity graph G with weight matrix W and the matrix $F = [f_{ij}]_{N\times m}$ where f_{ij} is the jth component of the vector $f(x_i)$, the *smoothness* of f with respect to G is defined as:

$$\Delta_G(f) \triangleq \sum_{i,j} w_{ij} \| f(x_i) - f(x_j) \|_2^2 = tr(F^T L F) = \sum_{i=1}^{m} F_i^T L F_i \tag{1.1}$$

where F_i is the ith column of F. Since the similarity weights w_{ij} are non-negative, $\Delta_G(f)$ is always non-negative. We say f is smooth with respect to G iff the values of function on neighboring nodes with high similarity weight is close. Intuitively, if f is relatively smooth with respect to G, $\Delta_G(f)$ is relatively small, close to 0. For example, if f is constant over the connected components of G, we have $\Delta_G(f) = 0$, However, if $f(x_i)$ is far from $f(x_j)$ for two nodes x_i and x_j with high similarity weight w_{ij}, the factor $w_{ij}\|f(x_i) - f(x_j)\|_2^2$ has a significant contribution to $\Delta_G(f)$.

By plugging in the eigen-decomposition of L, i.e. $L = \sum_{j=1}^{N} \mu_j u_j u_j^T$, in Eq. (1.1), we get:

$$\Delta_G(f) = \sum_{i=1}^{m} \sum_{j=1}^{N} \mu_j a_{ij}^2 \tag{1.2}$$

where $a_{ij} = F_i^T u_j$ is the projection of F_i on the jth eigenvector u_j of L. If one defines f as $f(x_i) = u_k(i)$, where $u_k(i)$ is the ith element of the eigenvector u_k, from Eq. (1.2), one can show that $\Delta_G(f) = \mu_k$. In other words, those eigenvectors of L with small eigenvalues define smooth functions on G. To generalize this result, let $f : \mathscr{D} \mapsto \mathbb{R}$ is defined as $f(x_i) = \sum_{j=1}^{m} \alpha_j u_j(i)$ for some coefficients α_j; that is, f is defined as a linear combination of the first m eigenvectors. Then we have $\Delta_G(f) = \sum_{j=1}^{m} \mu_j \alpha_j^2$. If μ_m is close to 0, then $\Delta_G(f)$ is relatively small and therefore f is relatively smooth. In other words, a linear combination of the smooth eigenvectors of L (with small eigenvalues) is also smooth on G.

1.2.2 The Laplacian Eigenmap

The *Laplacian eigenmap* [Belkin and Niyogi, 2002] is defined as the mapping $\mathscr{M}_L^m : \mathscr{D} \mapsto \mathbb{R}^m$ such that,

$$\forall x_i \in \mathscr{D} : \mathscr{M}_L^m(x_i) = [u_1(i), u_2(i), \ldots, u_m(i)]^T \tag{1.3}$$

where $u_j(i)$ here represents the ith element of the jth eigenvector of L_{rw}. Intuitively, the Laplacian eigenmap $\mathscr{M}_L^m$ uses the first m eigenvectors of L_{rw} (corresponding to the m smallest eigenvalues) to embed the nodes of the similarity graph into $\mathbb{R}^m$.

It turned out that the embedding given by the Laplacian eigenmap is *optimal* in the sense that it maps the nodes with high similarity in the graph to close vectors in $\mathbb{R}^m$ [Belkin and Niyogi, 2002]. To see this, consider the following optimization problem:

$$\begin{aligned} Y^* \leftarrow \arg \min_{Y \in \mathbb{R}^{N \times m}} & \; tr(Y^T L Y) \\ \text{s.t. } & \; Y^T D Y = I \end{aligned} \tag{1.4}$$

That is, we want to find the smoothest embedding matrix $Y_{N\times m}$ (with the minimum smoothness) such that the columns of Y are linearly independent in $\mathbb{R}^m$ (enforced by the constraint). Also, the D matrix in the constraint ensure the nodes with higher degree (i.e. d_{ii}) are considered to be more important. One can show that the solution to above optimization can be obtained by concatenating the first m eigenvectors (corresponding to the m smallest eigenvalues) of the generalized eigenvalue problem $Lu = \mu D u$, which are, in fact, the first m eigenvectors of L_{rw}. In other words, the Laplacian eigenmap $\mathscr{M}_L^m$ generates an embedding which is optimal w.r.t. the smoothness criterion. Furthermore, [Belkin and Niyogi, 2002, Belkin and Niyogi, 2005] have shown that in the limit the Laplacian eigenmap corresponds to the Laplace-Beltrami operator on manifolds.

1.2.3 The Diffusion Map

The *diffusion map* (aka the *random walk diffusion*) is another popular spectral embedding method that has been extensively studied in the literature. In fact, the random walk diffusion framework form the foundation for the proposed framework in Section 3.6 in Chapter

3 of this proposal. In this section, we give a brief overview of the diffusion framework; interested readers may refer to [Nadler et al., 2006, Lafon and Lee, 2006, Coifman et al., 2008, Coifman et al., 2005a, Coifman et al., 2005b, Lee and Wasserman, 2010, Amizadeh et al., 2012b] for further details.

1.2.3.1 Random Walk on Graph Assume a random walk on the similarity graph G with the transition probability matrix $P = [p_{ij}]_{N\times N}$ where p_{ij} is the probability of jumping from x_i to x_j in G given the random walker is currently standing on x_i. p_{ij} is set proportional to the similarity weight w_{ij}, in particular:

$$\forall i,j : p_{ij} = \frac{w_{ij}}{\sum_{k=1}^{N} w_{ik}} = \frac{w_{ij}}{d_{ii}} \tag{1.5}$$

As a result, each row i of the transition matrix P form a valid probability distribution modeling the transition distribution from node x_i to other nodes of the graph. In the matrix form, we have:

$$P = D^{-1}W = I - L_{rw} \tag{1.6}$$

The following properties hold for P:

(a) The eigenvalues of P are between 0 and 1 with the greatest one λ_1 (the spectral radius) equal to 1. [1]

(b) Each eigenvalue λ_i of P corresponds to two eigenvectors: a left eigenvector ϕ_i where $P^T\phi_i = \lambda_i\phi_i$ and the right eigenvector ψ_i where $P\psi_i = \lambda_i\psi_i$. That is, $P = \sum_{i=1}^{N} \lambda_i \psi_i \phi_i^T$

(c) Each left eigenvector ϕ_i is normalized as $\|\phi_i\|^2_{1/\phi_1} = \sum_{j=1}^{N} \frac{\phi_i^2(j)}{\phi_1(j)} = 1$

(d) Each right eigenvector ψ_i is normalized as $\|\psi_i\|^2_{\phi_1} = \sum_{j=1}^{N} \psi_i^2(j)\phi_1(j) = 1$

(e) The left and the right eigenvectors of P are bi-orthonormal; that is, $\phi_i^T\psi_i = \delta(i = j)$, where δ is the indicator function

(f) For each i,j, we have that $\psi_i(j) = \frac{\phi_i(j)}{\phi_1(j)}$

(g) The first eigenvalue is $\lambda_1 = 1$ and corresponds to $\psi_1 = \mathbb{1}$ and $\phi_1 = \frac{diag(D)}{tr(D)}$

(h) λ is the eigenvalue of P with the right eigenvector ψ iff $\mu = 1-\lambda$ is the eigenvalue of L_{rw} with eigenvector ψ

[1]As opposed to the eigenvalues of the Laplacian matrices, we assume the eigenvalues of P are sorted in the descending order.

(i) The larger eigenvalues of P correspond to the smooth right eigenvectors w.r.t. to the similarity graph G, with the largest eigenvalue (i.e. $\lambda_1 = 1$) corresponding to the smoothest right eigenvector $\psi_1 = \mathbb{1}$ which is in fact, a constant function.

1.2.3.2 Random Walk and Kernel Density Estimation

The concept of random walk on similarity graphs is closely related to the notion of *mixture modeling* and *kernel density estimation (KDE)* [Yu et al., 2005]. As mentioned in the beginning of this chapter, the similarity graph encodes the geometrical structure of the observed datapoints in the input space. The probability density function that generated the data in the first place also reflects the structure of the data. As a result, one may wonder whether these two paradigms are somehow related. As we argue in this section, the inherent connection between random walk and kernel density estimation provides one way of understanding this relationship. Later in Chapter 3, we incorporate this connection to extend the fast solutions for KDE to large-scale graph-based problems. Let us start with the formal definition of kernel density estimation.

Let $\mathscr{M} = \{m_1, m_2, \dots, m_M\}$ denote a set of kernel centers in $\mathbb{R}^d$. The kernel density estimate for any data point $x \in \mathbb{R}^d$ is then defined by the following weighted sum or mixture model:

$$p(x) = \sum_j p(m_j) p(x|m_j) \tag{1.7}$$

where $p(m_j) = \frac{1}{N}$, $p(x|m_j) = \frac{1}{Z_\sigma} k(x, m_j; \sigma)$, and k is a kernel profile with bandwidth σ and normalization constant Z_σ. For the commonly used Gaussian kernel, $k(x, m_j; \sigma) = \exp(-\|x - m_j\|^2 / 2\sigma^2)$ and $Z_\sigma = (2\pi\sigma^2)^{\frac{d}{2}}$.

Now let us assume we want kernel density estimates for the observed datapoints $\mathscr{D} = \{x_1, x_2, \dots, x_N\}$ in $\mathbb{R}^d$ that also define the kernel centers. That is, $\mathscr{D} = \mathscr{M}$, and we use the $\mathscr{D}$ and $\mathscr{M}$ notation to emphasize the two different roles every example x_i takes - one as a *query* datapoint (denoted by x_i), the other as a *reference* kernel center (denoted by m_i). Let us exclude the kernel centered at m_i from the kernel density estimate at that datapoint. In this case, the estimate at $x_i \in \mathscr{D}$ can be defined as the $N-1$ component mixture model:

$$p(x_i) = \sum_{j \neq i} p(m_j) p(x_i|m_j) = \sum_{j \neq i} \frac{1}{N-1} \frac{1}{Z_\sigma} k(x_i, m_j; \sigma), \tag{1.8}$$

where $p(m_j) = 1/(N-1)$.

Furthermore, one can use the Bayes rule to derive the posterior membership probability distribution $p(m_j|x_i)$:

$$p(m_j|x_i) = \frac{p(m_j)p(x_i|m_j)}{\sum_{h\neq i} p(m_k)p(x_i|m_h)} = \frac{p(x_i|m_j)}{\sum_{h\neq i} p(x_i|m_h)} = \frac{k(x_i,m_j;\sigma)}{\sum_{h\neq i} k(x_i,m_h;\sigma)}, \tag{1.9}$$

where the second equality comes from the fact that $p(m_j) = \frac{1}{N-1}$ for all $j \in \{1,\ldots,N\}$. The membership probability $p(m_j|x_i)$ expresses the probability that the datapoint x_i is indeed generated by the kernel m_j. $p(m_j|x_i)$ defines a valid probability distribution on the kernels $m_j \in \mathscr{M}$; that is, $\forall x_i \in \mathscr{D} : \sum_{j=1}^{N} p(m_j|x_i) = 1$.

Now let us investigate the connection to the graph-based methods. As we will see later in this chapter, a very common approach to build similarity graphs is to use a kernel to transform the distance between two datapoints into similarity. In other words, using this method, the similarity weight between x_i and x_j is defined to be $w_{ij} = k(x_i,m_j;\sigma)$, where σ is the bandwidth parameter. This means that, the similarity weigh w_{ij} and the likelihood $p(x_i|m_j)$ are equal up to a constant factor: $p(x_i|m_j) = \frac{1}{Z_\sigma} w_{ij}$. Furthermore, we have:

$$p(m_j|x_i) = \frac{k(x_i,m_j;\sigma)}{\sum_{h\neq i} k(x_i,m_h;\sigma)} = \frac{w_{ij}}{\sum_{h\neq i} w_{ih}} = p_{ij} \tag{1.10}$$

That is, the posterior membership probability $p(m_j|x_i)$ is equal to the transition probability of the random walk from datapoint x_i to datapoint x_j.

1.2.3.3 Multiscale Random Walk

Let $p_t(x_i,x_j)$ represents the probability of reaching x_j after t random jumps (also referred as *time steps*) starting on node x_i. For $t = 1$, we simply have $p_1(x_i,x_j) = p_{ij}$. Then the matrix P_t is defined as $P_t = [p_(x_i,x_j)]_{N\times N}$; again we simply have $P_1 = P$. The matrix P_t is in fact matrix P multiplied by itself t times; that is, $P_t = P^t$. The discrete parameter t is referred to as the *scale* of the random walk. The matrix P_t has the same left and right eigenvectors as P with the same eigenvalues exponentiated to the power of t; that is, $P_t = \sum_{i=1}^{N} \lambda_i^t \psi_i \phi_i^T$. Since, the absolute value of the P's eigenvalues are strictly less than 1, by exponenting to the power of t, the smaller eigenvalues of P (corresponding to non-smooth eigenvectors) diminish faster for P_t leaving the smooth eigenvectors. As $t \to +\infty$, all the eigenvalues of P_t diminish except for the first eigenvalues, $\lambda_1^{\infty} = 1^{\infty} = 1$. In this case, we get:

$$P_{\infty} = \sum_{i=1}^{N} \lambda_i^{\infty} \psi_i \phi_i^T = \psi_1 \phi_1^T = \mathbb{1}[\frac{diag(D)}{tr(D)}]^T \tag{1.11}$$

That is, we have $p_{\infty}(x_i, x_j) = \phi_1(j) = \frac{d_jj}{\sum_{k=1}^{N} d_{kk}}$; in other words, the elements of ϕ_1 represent the asymptotic probability of landing at each nodes of G regardless of the starting node.

If one views P as some kind of *local* similarity matrix, where high jumping probability p_{ij} between two neighboring nodes indicates their high similarity, then P_t can be seen as the extension of the local similarity matrix from immediate neighboring nodes to the nodes that are reachable within t steps. The probability $p_t(x_i, x_j)$ also accounts for all possible paths from x_i to x_j in t steps. Therefore, if there are many paths of length t in between x_i to x_j, the jumping probability (or equivalently the *global* similarity) is higher. This in fact makes the whole framework robust to the noisy data.

1.2.3.4 Diffusion Distance As mentioned in the previous section, P_t can be looked at as a sort of similarity/distance metric at scale t, but in reality, it is not a legitimate one. However, P_t can be used to define a valid multiscale distance metric called *diffusion distance*. The square diffusion distance at scale t between two nodes x_i and x_j in graph G is defined as:

$$D_t^2(x_i, x_j) = \|p_t(x_i, \cdot) - p_t(x_j, \cdot)\|_{1/\phi_1}^2 = \sum_{x_k \in \mathcal{D}} \frac{(p_t(x_i, x_k) - p_t(x_j, x_k))^2}{\phi_1(x_k)} \tag{1.12}$$

According to the diffusion distance, two nodes x_i and x_j must be close if the transition distributions from these nodes to the rest of the graph are similar. As mentioned before, $\phi_1(x_k)$ is the probability of ending up at node x_k in infinite steps. Therefore, the weight $1/\phi_1(x_k)$ in the above sum penalizes the contribution of the *popular* nodes (with high asymptotic landing probability) in the distance. Intuitively, having a similar transition probability from x_i and x_j to a popular node x_k does not necessarily imply that x_i and x_j must be similar.

1.2.3.5 Diffusion Map The *diffusion map* at scale t is defined as the mapping $\mathscr{M}_D^{t,m} : \mathscr{D} \mapsto \mathbb{R}^K$ such that,

$$\forall x_i \in \mathscr{D} : \mathscr{M}_D^{t,m}(x_i) = [\lambda_1^t \psi_1(i), \lambda_2^t \psi_2(i), \ldots, \lambda_K^t \psi_m(i)]^T \tag{1.13}$$

where λ's are the eigenvalues of P with the right eigenvectors ψ's. One can show that [Lafon and Lee, 2006]:

$$D_t^2(x_i, x_j) = \|\mathscr{M}_D^{t,N}(x_i) - \mathscr{M}_D^{t,N}(x_j)\|_2^2 \simeq \|\mathscr{M}_D^{t,m}(x_i) - \mathscr{M}_D^{t,m}(x_j)\|_2^2 \text{ for } m < N \tag{1.14}$$

In other words, $\mathscr{M}_D^{t,m}$ maps the graph nodes into a m-dimensional vector space in which the Euclidean distance approximates the diffusion distance in the original graph. This is, in fact, the main contribution of the spectral embedding methods which is transforming the graph nodes into a vector space in which the regular Euclidean distance approximates the the distance metric in the original space. By doing so, the spectral embedding methods make it possible for many Euclidean-based Machine Learning algorithms to be applied on non-Euclidean problems on the similarity graph.

1.2.4 Laplacian-based Kernels And Spectral Transform

The similarity graph G (and the Laplacian matrices derived from it) are built based upon some absolute (non-data-driven) local similarity in the input space and therefore are only meaningful *locally*. This means that if x_i and x_j are located in each other's *locality* in the input space, then the corresponding w_{ij} entry in G is a meaningful similarity value; otherwise, it does not convey any information regarding the similarity between x_i and x_j. Note that the notion of locality is problem-dependent; for example, if the input space is an Euclidean vector space, being in a same locality means having small Euclidean distance. Another example is the term space (which we will see in Chapter 2). In the term space, two terms can be seen in the same locality if they co-occure in the same sentence for instance.

To obtain a *global, data-driven* similarity metric which reflects a meaningful similarity value between any pair of datapoints, we need to somehow transform the local similarity matrix W (or equivalently L) into a global similarity kernel matrix $K = [k_{ij}]_{N\times N}$. For this purpose, we consider the *Laplacian-based kernel* K_m [Zhu et al., 2006] where K_m is a symmetric positive semi-definite kernel matrix of rank m $(1 \leq m \leq N)$ whose eigenvectors are the same as the first m eigenvectors of L (corresponding to the m smallest eigenvalues). That is, $K_m = \sum_{i=1}^{m} \theta_i u_i u_i^T$ where u_i's are the eigenvectors of L. Note that, unlike many standard kernels in Machine Learning literature (e.g. the Gaussian kernel), the Laplacian-based kernels are data-driven, meaning that the value of k_{ij} not only depend on x_i and x_j, it is also a function of all datapoint in the dataset $\mathscr{D}$.

The eigenvalues of K_m (i.e. θ_i's) cannot be chosen arbitrarily; they must satisfy certain conditions. First off, to get a valid kernel matrix, we need K_m to be positive semi-definite; that is, the eigenvalues θ_i's should be all non-negative. Second, we require the kernel value k_{ij} to be high if x_i and x_j are locally close. This naturally translates to the smoother eigenvectors of K_m should be assigned higher eigenvalues. Since the eigenvectors of K_m are the same as those of L, those with the smaller indices are smoother w.r.t. G. As a result, we need the eigenvalues to satisfy the ordering condition $\theta_1 \geq \theta_2 \geq \ldots \geq \theta_m \geq 0$. In fact, one can set θ_i as a direct function of how smooth u_i is; that is, $\theta_i = g(\Delta_G(u_i))$. The smoother u_i, the lower $\Delta_G(u_i)$ and the higher θ_i will be. Thus, g should be a *non-increasing* function of $\Delta_G(u_i)$. As shown before, the smoothness of u_i is simply equal to μ_i (the ith smallest eigenvalue of L); therefore, we have $\theta_i = g(\mu_i)$, where $g : \mathbb{R} \mapsto \mathbb{R}$ is a non-increasing function. Function g is referred to as *spectral transform* [Zhu et al., 2006].

Given a spectral transform g, one can define the general mapping $\mathscr{M}_g^m : \mathscr{D} \mapsto \mathbb{R}^m$ as:

$$\forall x_i \in \mathscr{D} : \mathscr{M}_g^m(x_i) = [\sqrt{g(\mu_1)}u_1(i), \sqrt{g(\mu_2)}u_2(i), \ldots, \sqrt{g(\mu_m)}u_m(i)]^T \tag{1.15}$$

Using $\mathscr{M}_g^m$, one can show that $k_{ij} = \langle \mathscr{M}_g^m(x_i), \mathscr{M}_g^m(x_j) \rangle$, where $\langle \cdot, \cdot \rangle$ denotes the inner product of two vectors. In other words, $\mathscr{M}_g^m$ maps the graph nodes in a m-dimensional feature space where the inner product is equal to the kernel values in the original space. Moreover, the Euclidean in the embedded space is equal to the distance defined base on the kernel matrix in the original space:

$$\|\mathscr{M}_g^m(x_i) - \mathscr{M}_g^m(x_j)\|_2^2 = k_{ii} + k_{jj} - 2k_{ij} \tag{1.16}$$

It turns out that the eigen mappings we have discussed in this chapter along with many others in the literature are special cases of the general mapping $\mathscr{M}_g^m$ in Eq. (1.15) with specific choices for the spectral transformation function g [Zhu et al., 2006]. Table 1 shows some of these methods along with the choice of g in each case. Note that in all cases, g is a non-increasing function of μ, an eigenvalue of the Laplacian matrix. The last row in Table 1 shows the *non-parametric* kernel where g does not have a functional form and its values for the eigenvalues μ are determined non-parametrically with the ordering constraint enforced. [Zhu et al., 2006] have modeled and solved the problem of the non-parametric kernel calculation from data as a *kernel alignment* problem.

Method	**Spectral Transform**	**Distance Name**
Laplacian eigenmaps	$g(\mu) = 1$	-
Diffusion maps*	$g(\mu) = (1-\mu)^{2t}$	Diffusion distance
Heat diffusion	$g(\mu) = \exp(-2\beta\mu)$	-
Resistance kernel	$g(\mu) = 1/(\mu+\epsilon)^2$	Resistance distance
Non-parametric kernel	$\forall i : g(\mu_i) \geq g(\mu_{i+1}) \geq 0$	-

Table 1: Different spectral graph-based methods with different spectral transforms. (*) Unlike other methods, diffusion maps are constructed based on the eigenvalues and eigenvectors of L_{rw} and not L.

In Chapter 2, we implement and study the last three methods in Table 1 to develop a generalized text metric.

1.2.5 The Diffusion Operator

In Section 1.2.3, the diffusion framework over the discrete set $\mathscr{D}$ at the discrete time scale t was discussed. A natural generalization is the extension of the diffusion framework to continuous compact sets with continuous time scales. For this purpose, in this section, we study *diffusion operator*. Interested readers may refer to [Nadler et al., 2006, Lafon and Lee, 2006, Lee and Wasserman, 2010] for more details. The diffusion operator and the convergence of its eigenfunctions form the basis for our discussion and proposed framework in Chapter 4.

Let the dataset $\mathscr{D} = \{x^{(1)}, \ldots, x^{(N)}\}$ be the instances of the random variables $V = \{X_1, \ldots, X_d\}$ sampled iid from the distribution P with compact support $\mathscr{X} \subset \mathbb{R}^d$ with a bounded, non-zero density p. Define the similarity kernel $k_\sigma(x,y) = \exp(-\|x-y\|_2^2/4\sigma)$ on $\mathscr{X}$. The *discrete-time diffusion operator* $A_{p,\sigma} : L^2(\mathscr{X}) \mapsto L^2(\mathscr{X})$ (where $L^2(\mathscr{X})$ is the class of functions f defined on $\mathscr{X}$ s.t. $\int f^2(x)dP(x) < \infty$) is defined as :

$$A_{p,\sigma}[f(x)] = \int_{\mathscr{X}} a_\sigma(x,y)f(y)p(y)dy, \tag{1.17}$$

with asymmetric kernel $a_\sigma(x,y) = k_\sigma(x,y)/\int k_\sigma(x,z)p(z)dz$. $A_{p,\sigma}$ is a compact, positive operator with the largest eigenvalue $\lambda_{\sigma,1} = 1$ corresponding to the constant eigenfunction $\psi_{\sigma,1} = 1$. Alternatively, $A_{p,\sigma}$ can be represented using its eigen decomposition as $A_{p,\sigma} = \sum_{i=1}^{\infty} \lambda_{\sigma,i}\Pi_i$ where Π_i is the orthogonal projection on $\psi_{\sigma,i}$. Moreover, $a_\sigma(x,y)$ can be written as $a_\sigma(x,y) = \sum_{i=1}^{\infty} \lambda_{\sigma,i}\psi_{\sigma,i}(x)\varphi_{\sigma,i}(y)$ where $\varphi_{\sigma,i}$ is the eigenfunction of $A^*_{p,\sigma}$, the adjoint of $A_{p,\sigma}$.

It is known that the principal eigenfunctions of $A_{p,\sigma}$ with the largest eigenvalues encode the manifold structure of data and therefore can be used for low dimensional embedding of the original data [Lafon and Lee, 2006]. In fact, this is the basic motivation for introducing *diffusion map* $\phi_\sigma : \mathbb{R}^d \mapsto \mathbb{R}^r$ for $r < d$:

$$x \mapsto \phi_\sigma(x) = [\lambda_{\sigma,1}\psi_{\sigma,1}(x), \ldots, \lambda_{\sigma,r}\psi_{\sigma,r}(x)]^T \tag{1.18}$$

The underlying structure of data can be studied at different scales (like in hierarchical clustering). This gives rise to the notion of *m-step discrete-time diffusion operator* defined by exponenting the eigenvalues of $A_{p,\sigma}$ to the power of m as $A^m_{p,\sigma} = \sum_{i=1}^{\infty} \left(\lambda_{\sigma,i}\right)^m \Pi_i$. Subsequently, the asymmetric kernel for $A^m_{p,\sigma}$ is derived as $a_{\sigma,m}(x,y) = \sum_{i=1}^{\infty} \left(\lambda_{\sigma,i}\right)^m \psi_{\sigma,i}(x)\varphi_{\sigma,i}(y)$. $A^m_{p,\sigma}$ also induces the diffusion map $\phi^m_\sigma(\cdot)$ which maps the original data to a coarser cluster structure as m increases. Furthermore, one can extend the discrete scale of $A^m_{p,\sigma}$ (i.e. m steps of length σ) to the continuous scale $t = m\sigma$ (with $\sigma \to 0$) by defining the *continuous-time diffusion operator* $A^t_p : L^2(\mathscr{X}) \mapsto L^2(\mathscr{X})$ [Lee and Wasserman, 2010]:

$$A^t_p[f(x)] = \lim_{\sigma \to 0} A^{t/\sigma}_{p,\sigma}[f(x)] = \int_{\mathscr{X}} a_t(x,y)f(y)p(y)dy, \tag{1.19}$$

where, $a_t(x,y) = \lim_{\sigma\to 0} a_{\sigma,t/\sigma}(x,y)$. The eigenvalues and the eigenfunctions of A^t_p are computed as: $\lambda^{(t)}_{p,i} = \lim_{\sigma\to 0}\lambda^{t/\sigma}_{\sigma,i}$ and $\psi^t_{p,i} = \lim_{\sigma\to 0}\psi_{\sigma,i}$, respectively.

1.2.6 Building The Similarity Graph

So far, we have assumed the similarity graph G is given; however, in many real problems, one needs to build the similarity graph from the dataset $\mathscr{D}$ before any further analysis. There are many methods for this task in the literature which depending on the original problem can generate different results [Maiker et al., 2009]. In this section, we review the basic methods for constructing the similarity graph G from dataset $\mathscr{D}$.

The main goal in constructing the similarity graph is to capture the *local* similarities among the members of $\mathscr{D}$. In other words, we ideally want to have an edge with high similarity weight between x_i and x_j if and only if they live in the same *locality* or *neighborhood*. However, this is not a trivial task; almost all the proposed algorithms for this problem in the literature face the following challenges:

(C1) There should exist a local similarity metric or equivalently a notion of locality in the problem in order to define the similarity graph.

(C2) The local similarity metric or equivalently the locality must be meaningful w.r.t. to the problem in question. In other words, if x_i and x_j live in the same locality according to the local similarity metric, they must also be considered similar according to the problem.

(C3) How big is the locality is usually an input parameter to the algorithm. Unfortunately, in many problems, there is no straightforward approach to set this parameter. The general advice is to set this parameter such that the resulted graph is connected or with only a few connected components [von Luxburg, 2007].

(C4) The computational and the memory requirements for constructing the similarity graph for many of the existing algorithms are typically of order $O(N^2)$ where N is the size of $\mathscr{D}$. For large N, the construction of G becomes a serious practical challenge. In Chapter 3 of this proposal, we address this problem in more details.

The graph construction algorithms also depend on the nature of objects in $\mathscr{D}$. A very common type are feature vectors residing in a d-dimensional Euclidean space (i.e. $\mathscr{D} \subset \mathbb{R}^d$). In the following four subsections, we assume that is the case for $\mathscr{D}$; later, we consider cases where the members of $\mathscr{D}$ are not explicitly represented as feature vectors.

Given $\mathscr{D} \subset \mathbb{R}^d$, (C1) can be resolved by using many famous distance metrics in Euclidean spaces with the Euclidean distance being the most popular one. (C2) is also addressed by assuming that being close in the Euclidean space induces similarity w.r.t. the problem in question. (C3) and (C4), however, depend on the specific choice of the algorithm for graph construction.

1.2.6.1 Weighted Fully Connected The first approach for construction of the similarity graph is to connect all the nodes (the elements of $\mathscr{D}$) and have the edges weighted according to their distance in the Euclidean distance. Ideally, we want the similarity weights decrease fast (e.g. exponentially) as the distance between the nodes increases. This is achieved through using of *similarity kernels* which are decreasing functions of the (Euclidean) distance. The most famous similarity kernel is the Gaussian kernel, defined as:

$$w_{ij} = k_\sigma(x_i, x_j) = \exp\big(\frac{-\|x_i - x_j\|_2^2}{2\sigma^2}\big) \tag{1.20}$$

Here, σ is called the *bandwidth* parameter and determines how fast the similarity weight vanishes as the distance increases. In other words, σ controls the locality radius for this method. Unfortunately, there is no intuitive way to tune this parameter. One of the contributions of the proposed framework in Chapter 3 is a systematic way to set the bandwidth parameter.

In terms of CPU and memory, the standard similarity kernel method is in the order of $O(N^2)$ which can be very expensive for large-scale problems. Also, the similarity matrix generated by this approach is not sparse which means that any further operation on this matrix will not be cheap either. In Chapter 3, we see many approximate algorithms in the literature designed to tackle this problem. In particular, we propose a fast and scalable method to compute the transition probability matrix P introduced in Section 1.2.3 directly without computing the similarity matrix.

1.2.6.2 *k*-nearest-neighbor (*k*NN) The basic idea of kNN is to connect each node to its k nearest neighbors according to the distance metric. The edges in the original kNN algorithm are assigned the similarity weight 1; however, they can be weighted using some similarity kernel (e.g. the Gaussian kernel). The parameter k determines the connectivity of the resulting graph. Although, k does not exactly determines the locality radius, it is strongly related to that. Unfortunately, setting k is not an easy task either. The optimal asymptotic rate when $N \to \infty$ is $\log N$ [Brito et al., 1997], however, for the moderate sample sizes there is no "optimal" rule.

kNN tends to generate a connected graph. This is normally considered a plus unless there exist clusters of datapoints with different densities in the input space which are not desired to be connected to each other. In those cases, a variant of kNN called the *mutual* kNN is recommended [von Luxburg, 2007]. In the mutual kNN, two nodes are connected only if both of them are among each other's k nearest neighbors. Note for both kNN and the mutual kNN, the resulted graph can have some nodes with degrees not equal to k. To generate a graph with uniform node degrees, [Jebara et al., 2009] has proposed an optimization setting called *b-matching*.

From the computational aspect, kNN normally takes $O(kN^2)$ CPU time and $O(kN)$ memory. Therefore, building a kNN graph using the standard algorithm can be costly for large-scale problems. In Chapter 3, we will see approximate methods using metric trees to find the kNN graph for large-scale problems. Memory-wise, however, kNN generates an sparse graph which makes many further operations fast.

1.2.6.3 ϵ-neighborhood In ϵ-neighborhood graph, two nodes are connected if their distance is less than the ϵ parameter. Like kNN, the edges can further weighted by some similarity kernel. The notion of locality is explicitly expressed in ϵ-neighborhood graphs such that ϵ exactly encodes the radius of the locality inside which the nodes will be connected. Setting ϵ can be very tricky specially if the datapoints have different densities at different parts of the space. An improper value of ϵ can result in either a fragmented or an overly-connected graph. Unfortunately, for finite sample sizes there is no optimal way of setting ϵ. One rule of thumb is to set ϵ as the average distance of a typical node to its k nearest neighbors where $k \sim log(N)+1$ [von Luxburg, 2007]. For infinite sample size, [Penrose, 1999] has shown that the optimal rate for ϵ that preserves the connectivity of the graph is $O((log(N)/N)^d)$. Computationally, building an ϵ-neighborhood graph takes $O(N^2)$ which is again expensive for large scale problems. In terms of memory, an ϵ-neighborhood graph van be represented via sparse matrices.

1.2.6.4 Minimum Spanning Tree (MST) Minimum spanning trees (MST) [Kleinberg, 2006] are effective tools for building similarity graphs. There are at least three main advantages gained by using MST: (1) the resulted graph is always connected, (2) the resulted graph is sparse, and (3) there is no locality radius parameter to worry about. However, these benefits come with some disadvantages which prevent us from using the standard MST in many real applications. Firstly, the edge connectivity is uniform across the graph and is equal to 1. This is not naturally desirable especially for datasets containing some clusters because one expects not to split one cluster into two only by removing one edge inside that cluster. Secondly and more importantly, MST algorithms can be very sensitive to the noise in data; that is, the end result can be very different if one of the datapoints is slightly perturbed. This is a very crucial issue as in many Machine Learning applications, noisy data is inevitable. To address these problems, [Carreira-Perpinán and Zemel, 2005] have proposed two algorithms based on MST called *Perturbed MST (PMST)* and *Disjoint MST (DMST)* to build the similarity graph. Unlike MST, PMST and DMST generate graphs which are not trees so the edge-connectivity is more than 1. Also, by construction PMST and DMST are very robust to noise. The downside with these algorithms is their computational complexities which is roughly $O(N^2 \log N)$ and make them prohibitive for large-scale problems.

1.2.6.5 Dealing With Non-vector Spaces So far, we have assumed that the elements of $\mathscr{D}$ reside in $\mathbb{R}^d$. However, that is not always the case; in many real-world applications the datapoints are not simply living in some Euclidean space. For instance, [Buchman et al., 2009] addresses the problem of estimating the density of the hurricane trajectories. Each datapoint (i.e. a hurricane trajectory) is a curve in 2D. Or as another example, in many text analysis problems, the datapoints are simply the terms in the lexicon without any feature representation attached to them; e.g. [Jin et al., 2003, Losada and Barreiro, 2003, Caillet et al., 2004]. The major challenge in this kind of problems is one cannot easily find a meaningful local similarity metric to define the notion of locality in these spaces. And we illustrated before, the notion of locality is essential to building similarity graphs. To tackle this problem, we note three general approaches one can take depending on the problem:

(a) One possible solution for non-vector datapoints is to somehow convert them into feature vectors in an Euclidean space where the Euclidean distance is meaningful locally. For example, in Chapter 4 of this proposal, we illustrate a systematic way of converting functional spaces into (possibly high dimensional) vector spaces where we can use the Euclidean distance to define the locality. This is based on the framework proposed in [Rossi et al., 2005]. The other example is extracting SIFT features from image object for various Machine Vision and Image Processing applications (e.g. [Bicego et al., 2006, Zhou et al., 2009, Ledwich and Williams, 2004]).

(b) The other solution is to define problem-specific local distance/similarity metrics that can define the locality in the original space. For instance, in [Buchman et al., 2009], the authors have proposed to use sample geographical locations from each hurricane trajectory and compute the sum of the Euclidean distances of the corresponding sampled locations between two trajectories as their distance. Earth Mover Distance (EMD) is another distance metric commonly used for non-vector objects specially functional objects and have been applied to compute the distance between images [Rubner et al., 2000], histograms [Ling and Okada, 2007], documents [Wan and Peng, 2005], musical signatures [Logan and Salomon, 2001] and etc. The problem with these specific distance metrics is they are not general and therefore may not be usable in a different problem. Also, the computational cost of calculating these metrics might be high; for instance, EMD is known to be computationally expensive [Shirdhonkar and Jacobs, 2008].

(c) A third approach is to go around finding a local similarity metric and directly define the locality in the problem. Note that the sole purpose of the local similarity metric is to capture the locality in the input space; if one can somehow directly define the locality, there would be no need to have a local similarity metric. However, this is only doable in some problems. For example, in Chapter 2, one problem is to build a similarity graph on the space of words that have appeared in a corpus of documents. We have considered the locality in this problem as the scope of a sentence; that is, if two words co-occure in a same sentence, we assume they belong to the same locality and therefore we will add an edge between them in the similarity graph.

2.0 CASE STUDY: CONSTRUCTING GENERALIZED LOCALLY-INDUCED TEXT METRICS

To demonstrate the usage of the non-parametric graph-based methods in practice, in this chapter, we develop a graph-based framework for constructing text metrics. The framework can be used to compare and support inferences among terms and sets of terms representing text components such as sentences, paragraphs, abstracts or whole documents. Our metric is derived from data-driven kernels on graphs, introduced in the previous chapter, that let us capture global relations among terms and sets of terms, regardless of their complexity and size. To compute the metric efficiently for any two subsets of terms, we develop an approximation technique that relies on the precompiled term-term similarities. We demonstrate the benefits of the whole framework on two text inference tasks: prediction of terms in the article from its abstract and query expansion in information retrieval. It should be noted that this case study have been accomplished in collaboration with Shuguang Wang, Ph.D. Candidate at University of Pittsburgh. [Amizadeh et al., 2011]

2.1 INTRODUCTION

A huge number of text documents are published or shared every day in different areas of science, technology, culture etc. Due to the huge volumes, the analysis of these documents and their contents is becoming increasingly hard. This prompts the development of tools that let us better analyze these texts and support various automatic inferences upon their content.

This chapter focuses on the development of a new class of text kernels that let us capture complex relations between terms and sets of terms in a corpus. Such kernels can be very useful for analysis of term relations, or to support inference tasks such as term predictions, term clustering, or more applied tasks such as query expansion. The key challenge in designing a good text kernel is to account for indirect global term relations that are not immediate from relations explicitly mentioned in the text. As an example, there is a logical relevance between 'designer' and 'software' and between 'designer' and 'cloth', while 'software' and 'cloth' may not ever happen together in a document while there is a weak similarity between them as 'artifacts designed by humans'.

At the core of our methodology is the design of term-term similarity metrics (or, in other words, term-term kernels). These metrics aim to capture abstract and often complex relations among terms and their strength as they appear in the document corpus. All the proposed metrics in this paper are derived from a graph of local associations among the terms observed in the document corpus. We call this graph the *association graph*. To cover and account for indirect relations which span multiple direct associations, we define a global term similarity metric based on the spectral decomposition of the association graph and a special class of graph-Laplacian kernels [Zhu et al., 2006] that assure the smoothness of the metric across observed term associations.

The term-term similarity metric is defined on the term space. However, many useful text inferences, e.g. query expansion, work with sets of terms. To address the problem, we need a *generalized* term similarity metric that lets us represent set-set similarities. We show how this new metric can be, in principle, built by expanding the original association graph with N nodes (corresponding to terms) with special auxiliary nodes representing sets of terms. However, computing the distance between any two sets would require a new graph expansion and the recalculation of the $O(N^3)$ spectral decomposition, which is infeasible in practice. We approach the problem by proposing and defining a new method that can efficiently approximate the set-set similarities on-line, whenever they are needed, by computing the spectral decomposition of the underlying graph only once.

The spectral decomposition of the graph Laplacian takes $O(N^3)$ time. This is prohibitive for large N even if computed only once. One way to alleviate the issue is to benefit from the graph structure and its disconnected components. A more principled approach requires a reduction of the number of terms in the graph. In the Chapter 3, we propose and study an approximation approach that first performs the decomposition on a randomly selected sub-graph of the association graph, and then extends the results to the entire graph to approximate the spectral decomposition of the full graph.

As mentioned earlier, a good metric relating terms and their sets can be used to support various text inferences such as text clustering, query expansion, etc. We demonstrate the benefit of our text kernels on two text inference problems: (1) prediction of terms in the full document from terms in its abstract and (2) the query expansion for the retrieval of documents relevant to the search query [G. Cao and Robertson, 2008]. The key question we must ask when designing these tools is: do we need a deep semantic analysis and understanding of the text in order to support well all text related inferences? The lessons learned from the link analysis on document collections have showed us that such analysis is not always necessary and that the structure of references or links can reveal a great deal about each individual document and that the analysis of these structures alone is sufficient to perform well on many document related tasks. In our work we pursue a similar idea, but this time we do not want to work in the document space but in the term space. Also our goal is more ambitions: we want to support inferences among arbitrary text components, which may correspond to individual terms, sentences, paragraphs or documents.

2.2 RELATED WORK

In this section we briefly review the related work defining and using similarities with graph-based methods for text applications. Embedding-based methods for text metric learning have been recently studied in the literature. In [Lebanon, 2006], parametric metrics have been learned for text documents based on Fisher geometry. In [Cai et al., 2005], text documents are mapped into a semantic space using Locally Preserving Indexing. However, in both frameworks, the text metrics only compare documents and not any arbitrary chunks of texts.

Laplacian kernels and their special cases, on the other hand, have been used in various machine learning problems. However, their application in text analysis is still relatively rare. [Dillon et al., 2007] used a specific Laplacian kernel to derive the document similarity for Machine Translation tasks. [Bordino et al., 2010] proposed to estimate the query similarity using the classical spectral projection on the query flow graph. This is in fact a special case of Laplacian embedding using step function transform which is used on the graph over queries instead of terms. Our framework is more general than these two works and also assures the metric is smooth in that it respects directly the observed term relations.

[Collins-Thompson and Callan, 2005] proposed to model term association using the Markov chain (random walk) model defined on the graph over terms. The edges in the graph are defined using multiple external sources such as the synonyms in WordNet [Moldovan and Rus, 2001]. The analysis is performed on the term relevance graph directly instead of its Laplacian matrix. We propose to model term relevance using a general family of graph-Laplacian kernels.

We note that there are different ways of defining term similarity other than graph-based methods. For example, [Wang et al., 2010] propose to apply a PHITS model, originally designed for the link analysis of document networks, to term association graphs. The PHITS model learned from the training data is then used to approximate the probability of a term-term association. However, this method projects terms into a latent low-dimensional space representing clusters of interconnected terms and the similarity for any pair is computed in this space. In our experiments, we show that these methods are outperformed by our graph-based similarity metrics.

For computing the similarity between sets of objects, [Kondor and Jebara, 2003] proposed to compute the Bhattacharyya divergence between the densities of the sets in the embedded space. [Bach, 2008] proposed to use efficient graph kernels to compute the kernel between point clouds. Our set similarity extension is different from these methods in that first we work with sets of terms (and not vectors) and second our work is inspired by short-circuiting in electrical circuits.

2.3 THE GRAPH-BASED TEXT METRIC

We build our framework for generating text distance metrics based on the Laplacian-based graph kernels introduced in the previous chapter. In particular, we first show how a distance metric between the simplest elements of texts, namely terms, can be induced and then generalize it to define a distance metric between the sets of terms.

2.3.1 Term-Term Distance Metrics

First, let us define the key ingredients of our framework.

Term association graph is a weighted graph $\mathscr{A}$ with nodes V corresponding to the terms in the lexicon extracted from an input document corpus. The edges in $\mathscr{A}$ represent the co-occurrence of the two terms corresponding to the edge in the same sentences. Furthermore, each edge is assigned an *association weight* in $\mathbb{R}^+$ expressing the strength of the relation. For the term co-occurrence relation, the strength is the number of different documents in which the two terms co-occur in the same sentence. We note that, in general, the co-occurrence relation and its weight can be replaced with any reasonable term-term relation and corresponding statistics as long as it is easy to extract them from the input corpus.

Relevance function is defined as a vector-valued function $r : V \mapsto \mathbb{R}^k$ such that if two terms t_i and t_j are considered to be *relevant* according to the domain of the desired task, then $\|r(t_i) - r(t_j)\|_2^2$ is small. Thus, knowing $r(\cdot)$ would greatly help to build a reliable term-term distance metric. However, in general, the true $r(\cdot)$ is unknown for a given problem.

Now, the question is how $r(\cdot)$ and $\mathscr{A}$ are related? Our key assumption in this work is: $r(\cdot)$ *is relatively smooth with respect to* $\mathscr{A}$*; that is, the smoothness* $\Delta_{\mathscr{A}}(r)$ *in* (**??**) *is relatively small.* As a result, we can use the Laplacian-based kernels derived from $\mathscr{A}$ to define term-term distance metrics which are able to capture the true relevance among the terms. In particular, we use the resistance, diffusion and non-parametric kernels in the previous chapter to build distance metrics on terms.

As mentioned before, the parameters of these kernels can be optimized based on the task (or equivalently the true $r(\cdot)$). However, since $r(\cdot)$ is unknown, we can use some proxy objective for this optimization. In particular, if we have the binary information whether two terms are relevant or not on a subset of terms as training data, we can maximize the AUC measure between the goal standard and the distance ordering derived from the kernel on the same set of terms. Based on this objective, the optimization of single-parameter kernels, such as the diffusion kernel, can be carried out using a simple line search procedure. For the non-parametric kernel (with spectral transformation parameters θ_i subject to constraints), we define a linear program to find the optimal $\boldsymbol{\theta}$ vector as:

$$\begin{aligned} \max_{\boldsymbol{\theta}=(\theta_1,\ldots,\theta_n)^T} \quad & \sum_{ij} K(t_i,t_j) - b\boldsymbol{\theta}^T\Lambda \\ \text{s.t.} \quad & 0 \le \theta_i \le \theta_{i+1} \le 2 \ \forall i = n-1,\cdots,1 \end{aligned} \tag{2.1}$$

where the sum is over all pairs of terms which are considered relevant according to the training data, $b \ge 0$ is a regularization penalty and Λ is the vector of eigenvalues of the $\mathscr{A}$'s Laplacian. The order constraints over θs assure that smoother eigenvectors are assigned higher weights. Intuitively, the program tries to maximize the kernel values for the observed relevances and at the same time (using the second term) penalize the eigenvalues of the kernel matrix associated with large eigenvalues of Laplacian.

Now that the kernel is specified and its parameters are optimized, one can derive the mapping $\phi(\cdot)$ using Eq. (4.19) to define the distance between terms. In fact, $\phi(\cdot)$ can be seen as an approximation to the true $r(\cdot)$.

2.3.2 Set-Set Distance Metric

To generalize the distance measures derived in the previous subsection to the distance between sets of terms, a straightforward approach is to somehow combine the mutual term-term distances between the two sets. To do so, the natural candidates are the *max* ($d^{max}(\cdot,\cdot)$), the *min* ($d^{min}(\cdot,\cdot)$) and the *average* ($d^{avg}(\cdot,\cdot)$) functions. Here, we develop a more principled approach to compute the distance between two sets of terms. We call this distance $d^{collapse}(\cdot,\cdot)$.

Recall that the resistance kernel in the previous chapter approximates the total resistance between two nodes in the graph if the edge weights are interpreted as reciprocal of resistance. In an actual circuit, in order to compute the total resistance between two sets of nodes S_1 and S_2, one should first short-circuit all the nodes in each set separately to collapse each set to one node. Then, the total resistance between the collapsed nodes is equal to the total resistance between S_1 and S_2. Figures 2(I)&(II) illustrate the idea. Figure 2(I) shows an electrical network; Figure 2(II) is the same network after collapsing the terms (nodes) in the set $S = \{A, E\}$.

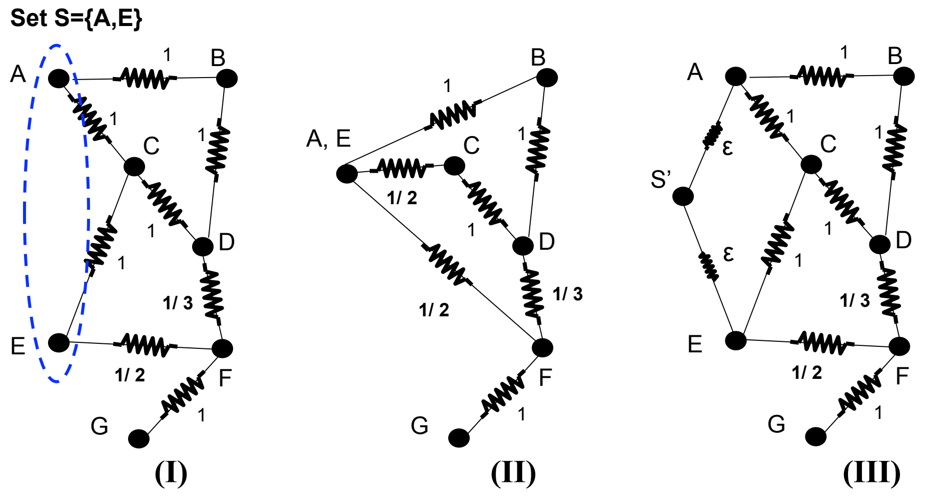

Figure 2: Collapsing nodes in electrical resistance network.

Note that the electrical circuit example is just one analogy and the core idea is more general. In fact, short-circuiting the nodes in a set S in an electrical circuit is equivalent to adding high association (0 resistance) edges between S's members in a more general graph. After doing so, if a candidate node x is similar to *any* of S's members then it will become similar to all the nodes in S (due to the insertion of high association edges). This somehow encodes an 'OR' logic.

We extend the same idea to compute the distance between two sets of terms for any Laplacian-based kernel. That is, to compute the distance between the term sets S_1 and S_2, we collapse the corresponding terms of each set in $\mathscr{A}$ to one super node to obtain a new reduced association graph $\mathscr{A}'$. After that we recompute the Laplacian-based kernel for $\mathscr{A}'$ to get the distance between the collapsed nodes. The main drawback of this approach is for any given two subsets of nodes, we have to reshape the graph, form the Laplacian matrix and compute its eigen decomposition which takes $O(N^3)$ time.

To avoid recalculations, we propose an efficient approximation that does not require us to change the structure in the underlying association graph. The solution is based on the following electrical circuit analogy: short-circuiting the nodes in some set S is equivalent to adding an auxiliary node s' to the network and connecting it to the nodes in S with zero resistance (infinite association weight) links. The total resistance between s' and any other node in the graph is equivalent to the resistance between the collapsed node representing S to these nodes. We apply the same trick on the association graph to build our approximation: instead of collapsing the terms in a set into one node, we add a new node s' to $\mathscr{A}$ and connect it to all terms in S with some high association weights (to approximate infinite weights). This is illustrated in Figure 2(III). Now, we have to compute the eigen decomposition for the Laplacian of $\mathscr{A} \cup s'$ to find the mapping $\phi(s')$ for the new node; however, instead of recalculating everything from scratch, we can just extend the eigenvectors of $\mathscr{A}$'s Laplacian to one more element using the Nyström approximation [Buchman et al., 2009]. More specifically, if node s' was included in the graph, we would have $\sum_j L(s',j)u_k(j) = \lambda_k u_k(s')$, for the k^{th} eigenvector of L. Solving for $u_k(s')$, we will get:

$$\forall k, u_k(s') = \frac{1}{\lambda_k - L(s',s')} \sum_{j \neq s'} L(s',j)u_k(j), \tag{2.2}$$

where $L(s',j)$ is just the negative assigned association weight between node j and the auxiliary node s' (and 0 if $j \notin S$). Also $L(s',s')$ is the degree of s'. Having approximated $u_k(s')$ for all k, we can compute $\phi(s')$ using Eq. 4.19.

Using this approximation, we propose to define and calculate the set-set kernel as follows. First, we compute and store the eigen decomposition for the term-term associations, which takes $O(N^3)$ time and $O(N^2)$ space. This step is performed offline. The metric for any two sets of terms S_1 and S_2 is then calculated online using the stored decomposition. In fact, it only takes $O(|S_1| + |S_2|)$ to find $\|\phi(s_1) - \phi(s_2)\|_2^2$.

2.4 EXPERIMENTS

A good text similarity metric can open door to some interesting applications: predicting term occurrences from text components, clustering of text components, query expansion, etc. In this section, we demonstrate the merits of our framework on two applications; prediction of terms in the document and query expansion in information retrieval. What is interesting is that our methods does not depend on sematic analysis via NLP to acquire these abilities, but simply relies on term co-occurrences. Hence is complementary to these methods.

2.4.1 Term Prediction

The objective of this experiment is to demonstrate that our kernel-based distance metrics can predict the occurrence of terms in a full article from terms in the abstract. Intuitively, for the same document, terms in the full body should be very relevant to the terms mentioned in its abstract. Thus, given the terms in the abstract, a good similarity metric should prefer terms that are mentioned in the full document.

2.4.1.1 Data The documents used in this experiments are from the cancer corpus [Wang and Hauskrecht, 2008] that consists of 6,000 documents related to 10 cancer subtypes that are indexed in PubMed. The articles were randomly divided into the training (80%) and test (20%) sets. Only the abstracts in the training set were used to build the term association network. Although, we could have trained our kernels using the terms in the document bodies as well, they perform well over the entire vocabulary even just using the terms in the abstracts. The terms extracted were the names of genes and proteins occurring in the free text. We used LingPipe[1] to identify genes and proteins. If needed, abstract-document pairs in the training data were used to optimize kernel parameters, such as σ^2 in the heat diffusion kernel, and θs defining a non-parametric Laplacian-based kernel.

2.4.1.2 Evaluation Metric For evaluation, we first compute the distances between terms in the abstracts to all candidate terms and rank them. If our proposed similarity metrics

[1]http://alias-i.com/lingpipe

is good, the terms in the full body of the text should be ranked higher than the rest. We assess this using the Area Under the ROC Curve (AUC) score. More specifically, we assign label 0 to those concepts that were not observed in the full article and 1 to those that were observed. The ranking based on the metric is then combined with the labels and the AUC score is calculated. Note that the optimal term ordering for this document should give a perfect separation of 1s and 0s.

2.4.1.3 Baselines We compare our methods to three baseline methods: TF-IDF, PHITS, and the shortest-path approach. The TF-IDF method predicts the terms in the document body using the TF-IDF score ranking [Salton and McGill, 1983] calculated on the training documents and is independent of the query (the terms in the abstract). The PHITS-based approach [Wang and Hauskrecht, 2008] first learns the PHITS model [Cohn and Chang, 2000] from the term association graph and uses it to approximate the strength of the term and set-to-term relations. The shortest path method uses the term association graph and its reciprocal weights to calculate the shortest paths between terms in the abstract and the rest of the terms. The shortest path lengths are then used to estimate term-term and set-to-term similarities.

2.4.1.4 Results Table 2 summarizes the results of the experiment on the *full* term vocabulary of 1200 test documents. For each method, the table shows the mean AUC scores obtained for test documents and their 95% confidence intervals (CI).

Baselines vs. Kernels: All Laplacian-based metrics were statistically significantly better than baselines when predicting the terms in full documents. This means the derived similarity metrics are very meaningful and model the term relevance better than baselines.

Comparison of Kernels: The parameters of all kernels were optimized using either line search (for the diffusion and the resistance kernels) or the linear program (for the non-parametric kernel). There are small overlaps between confidence intervals of different kernel methods. To examine the differences in the mean AUC scores more carefully, we analyzed the methods using pair-wise comparisons. We found that the resistance kernel performs statistically significantly better than other kernels. The diffusion kernel and the

Methods	AUC	95% CI
TF-IDF	0.782	$[0.767, 0.801]$
PHITS	0.781	$[0.749, 0.805]$
shortest-path	0.745	$[0.729, 0.756]$
$K_{Diffusion}$	0.878	$[0.870, 0.887]$
$K_{Resistance}$	**0.883**	$[0.878, 0.892]$
$K_{Nonpara}$	0.870	$[0.863, 0.878]$

Table 2: AUCs for predicting terms on test documents. The best AUC score is in bold.

non-parametric kernel were not significantly different. We attribute the superiority of the resistance kernel to the fact that it heavily emphasizes the smoother (smaller) eigenvalues of the Laplacian compared to other kernels due to its functional form.

We attribute the superiority of the resistance kernel to its functional form (Section 2.3). In particular, we observed that the first 113 eigenvalues of the Laplacian in this experiment are less than 1e-4; in other words, the first 113 eigenvector are very smooth and encode critical information regarding the structure of the underlying graph. The resistance kernel, according to its functional form, puts much more spectral transformation weights on eigenvectors with small eigenvalues than other kernels. For example, the ratio of two spectral transformation coefficients (for 113th and 114th eigenvalues) for the resistance kernel is 1045.7, while the same ratio for the diffusion kernel and the same set of eigenvalues is only 1.01. This shows that the resistance kernel tends to emphasize the components which are critical for the structure more than other kernels.

2.4.2 Query Expansion

In this experiment, we test our metrics on the query expansion task. Briefly, in the query expansion task, we seek to find a small number of terms that can help us to improve the retrieval of relevant documents if they are added to the original query. Here, we enrich a given query with the terms considered close to it according to the resistance kernel.

2.4.2.1 Datasets We use four TREC datasets[2] to analyze the performance of our method on the query expansion task: Genomic Track 2003, Genomic Track 2004, Ad hoc TREC 7, Ad hoc TREC 8. The key properties of these datasets are summarized in Table 2.4.2.2. Each TREC dataset comes with 50 test queries and the list of relevant documents assessed by human experts for each query.

TREC	**Type**	**# of docs**	**N**	**M**	**Term type**
Genomic-03	abs	500k	349K	5K	gene/protein
Genomic-04	abs	4.5mil	1123K	5K	gene/protein
Ad Hoc 7	doc	550k	469K	20K	words
Ad Hoc 8	doc	550k	469K	20K	words

Table 3: TREC datasets used in for query expansion (abs=abstract, doc=document, N = total # of terms, M = # of terms used).

2.4.2.2 Experimental setup Since there are no query expansion baselines, we use our methods in combination with Terrier search engine [3] to rank the documents and observe its relative performance to the baselines. Terrier is a search engine that parses and indexes the document corpus to build its own vocabulary; its performance can be further improved by doing query expansion first. For baselines, we use: (1) Terrier search engine without query expansion, and (2) Terrier with the PRF-based (pseudo-relevance feedback) [Xu and Croft, 1996] query expansion. PRF methods are the state-of-the-art methods for query expansion that use auxiliary searches to expand original queries. They use all terms in the term vocabulary. We report the best results from the DFR-Bo1 model included in Terrier, which is based on Bose-Einstein statistics [Macdonald et al., 2005]. In contrary to PRF, our methods cannot scale up well due to the huge number of terms in these datasets. To address this

[2]http://trec.nist.gov/data.html
[3]http://www.terrier.org

problem, we have subsampled and work with a subset of the overall terms. Yet the end results are very comparable to those of PRF. We postpone the details of the subsampling technique used to the next chapter. Note that M in Table shows the size of the subsample used.

2.4.2.3 Results Table 26 summarizes the result for the experiment. The statistics used in the evaluation is a widely used document retrieval evaluation metric, the Mean Average Precision (MAP) [Buckley and Voorhees, 2005]. The table shows that our kernel-based query expansion either outperforms or comes close to Terrier's PRF-based expansion baseline which is the state-of-the-art. The difference in Ad Hoc 8 can be explained by applying our method on a reduced term space which includes approximately 25% of the original terms.

Methods	**Genomic 03**	**Genomic 04**	**Ad Hoc 7**	**Ad Hoc 8**
Terrier	0.19	0.31	0.18	0.24
Terrier+PRF	0.22	**0.37**	**0.22**	**0.26**
Terrier+ $K_{resistance}$	**0.24**	**0.37**	**0.22**	0.25

Table 4: MAP of methods on document retrieval tasks on TREC data.

2.5 DISCUSSION

To show the benefits of the graph-based methods in practice, in this chapter, we developed a graph-based framework for constructing text metrics to compare any two arbitrary text components. One important feature of our framework is being *global* meaning that as opposed to the traditional document similarity metrics, the metrics produced by our framework are able to detect the relevance between two text components for which their corresponding terms neither overlap nor co-occur in the same sentence/document across the corpus. This property is due to the fact that our framework is based on the data-driven graph Laplacian kernels on the term space which are capable of inferring global relevance beyond local relations.

The other key feature of our framework is that it produces a consistent distance measure for two input texts regardless of their sizes (e.g., comparing a term vs. an abstract). We achieved this property by generalizing the distance between two terms to the distance between two sets of terms. To avoid recalculations in computing the distance between arbitrary sets, we proposed an efficient approximate technique which uses the results of one-time spectral decomposition to compute the distance between any two given sets in an online fashion.

To show the merits of our framework in practice, we used the metrics constructed by our framework for term prediction and query expansion. In both experiments, our metric outperformed the traditional baselines. These very promising results justify further investigation, refinements and possible deployment of our framework for solving real world problems in the field of text mining.

Although the real-world experiments in this chapter showed the effectiveness of the graph-based framework compared to other popular methods for text analysis, they also revealed one major restriction with the graph-based methods: they do not scale up well as the number of nodes in the graph (e.g. terms in this chapter) increases. The main reason behind this shortcoming lies in the representation of the main element in the graph-based methods, the similarity graph. In order to build and store the similarity graph (the association graph in this chapter), one needs to maintain the similarity matrix which takes $O(N^2)$ time and memory, where N is the number of nodes. This may not be problematic for small problems, but for datasets of size in the order of 10^4 or larger, the CPU and the memory will become serious issues. Even if the similarity matrix is given to us on a machine with infinitely large memory, many of the operations on this matrix still takes at least $O(N^2)$ time. For instance, as we will see later in this proposal, matrix-vector multiplication is a common operator on the similarity matrix (or its Laplacian) that is used in many applications such as Link Analysis and Label Propagation. This operation takes $O(N^2)$ time in general. Or even worse, the full eigen-decomposition of the Laplacian matrix takes $O(N^3)$, which as we saw, is a crucial step in our proposed framework in this chapter. As a result, one needs to address the "large N" challenge in a principled way when facing real-world large-scale problems. This is the focus of the next chapter.

3.0 LARGE N

3.1 INTRODUCTION

At the core of the graph-based methods is the similarity graph that captures the geometry of the datapoints in the input space. Depending on the characteristics of the input space, the local similarity metric and the representation of the datapoints, the similarity graph can be generated in numerous ways as described in Chapter 1. Nevertheless, no matter how the similarity graph is generated, a common representation to store it and operate on it is in the form similarity matrix which, in general, takes $O(N^2)$ time and memory to maintain, where N is the number of datapoints or nodes in the graph. With N in the order of 10^4 or larger, we face a practical challenge as the demand for CPU and memory rapidly grows compared to the computational power of many non-cluster machines. This is not just a hypothetical problem as many of the real-world problems these days contain hundred thousands to millions of records which poses a huge restriction in using the graph-based methods. We refer to this problem as the *Large N* problem.

Unfortunately, the problem does not end here. Imagine the similarity matrix is "magically" computed and given to us on a machine with infinitely large memory. In order to use this matrix for our further purposes, we need to perform certain operations on this matrix. Most of these operations inherently need to visit each element of the matrix at least once. This means that for many operations the computational order is at least $O(N^2)$. As a simple example, imagine one is interested in finding the degrees of the nodes in the similarity graph; this quantity is in fact closely related to the density of the points in the input space. Another example is the matrix-vector multiplication which is a common operator in Link Analysis [Ng et al., 2001c, Ng et al., 2001d] and Label Propagation. A more complicated operation is the full eigen-decomposition which takes $O(N^3)$. In fact, as we saw before, the eigen-decomposition of the Laplacian matrix plays a crucial role in the spectral graph-based methods. Therefore, even in those problems where the similarity matrix is pre-computed and/or the memory is abundant, we are still facing a big computational challenge to do operations on large-scale similarity matrices.

In this chapter, many solutions for the Large N problem are overviewed. The important point to note here is *not all these solutions are applicable in all problems*. Depending on the representation of the datapoints in the input space, the similarity/distance metric in the input space, the quantity of interest in the problem and many other factors the appropriate solution might be very different. However, regardless of all the differences, most of these solutions have one common essence and that is the *reduction* of the similarity matrix representation. Depending on the methodology, the reduced representation can come in different forms: a sub-graph (sub-matrix), a sparse matrix, a cluster tree, etc.

After giving an overview of the existing approaches to the Large N problem in the literature, we focus on two specific methods. In particular, in a flashback to the text metric problem in Chapter 2, Section 3.3 illustrates how the random subsampling method combined with the Nyström-based expansion can effectively overcome the Large N problem in our text analysis case study. In fact, we successfully used this technique in the experiments reported in Chapter 2. In the rest of the chapter, we propose a more elaborate reduction method that is specially useful for the problems where the datapoints are explicitly represented in a metric feature space. In particular, we develop a variational dual-tree framework to effectively represent the transition matrix of the random walk on the graph (or equivalently the random walk Laplacian matrix L_{rw}). The transition matrix is specifically useful in applications such as Diffusion Analysis, Semi-supervised Learning and Link Analysis [Ng et al., 2001c, Ng et al., 2001d]. In particular, using our proposed framework, we demonstrate order of magnitudes speedup without sacrificing accuracy for Label Propagation tasks on real datasets in semi-supervised learning.

3.2 RELATED WORK

Based on the type of reduction, the approximate techniques for graph-based method fall into different categories as follows.

3.2.1 Node Sparsification Methods

The key idea behind node sparsification techniques is to reduce the number of nodes in the similarity graph (i.e. N) to M such that $M \ll N$. The differences arise on how each specific algorithm achieves this goal. The most intuitive approach is taking a *random subsample* of the nodes with size M [Talwalkar et al., 2008, Kumar et al., 2009]. The main challenge with subsampling methods is how to extend the results of the analysis to those nodes that are left out of the sample. One common approach is to use Nystrom approximation [Buchman et al., 2009, Kumar et al., 2009, Williams and Seeger, 2001].

The other popular idea for node reduction is building a *backbone graph* which captures the general structure of the similarity graph via a reduced set of *super nodes* each of which represents a subset of the original nodes. Different algorithms in the literature build the backbone graph differently. In [Zhu and Lafferty, 2005], the authors created a mixture model of the data and used the mixture components as super nodes. [Yu et al., 2005] has used non-negative matrix factorization for building the backbone graph. [Valko et al., 2012] has used the *Doubling Algorithm* for incremental clustering [Charikar et al., 1997] to build a backbone graph from an online stream of data. Gridding (binning) the input space is another method to build the backbone graph [Silverman, 1982, Garcke and Griebel, 2005]. The problem with gridding is the size of the backbone graph increases exponentially with the dimensionality of the space. The other method is to use clustering (e.g. k-means clustering) to pick the representatives as the nodes of the backbone graph (For example [Yan et al., 2009, Liu et al., 2010]).

3.2.2 Edge Sparsification Methods

The second group of reduction methods aim at reducing the number of graph edges instead of its nodes. k-nearest-neighbor graphs [von Luxburg, 2007, Zhu, 2005b] and mutual k-nearest-neighbor graphs [von Luxburg, 2007] are among the most popular methods for building sparse similarity graphs. *b-matching* is a similar technique to the k-nearest-neighbor method which guarantees that all the nodes have the same degree in the sparsified graph [Jebara et al., 2009]. A common property of all these three methods is that the node degrees (i.e. the edge sparsity) are somewhat uniform across the graph regardless of the density of the datapoints in the space. In contrast, ϵ-neighborhood graphs [von Luxburg, 2007] captures the density of the datapoints by having different node degrees depending on the density. However, setting the k parameter in the k-nearest-neighbor method is normally an easier task than setting the ϵ parameter in the ϵ-neighborhood approach. *Rank Modulated Degree (RMD)* [Qian et al., 2011] is another method that takes into account the density of the datapoints while generating a sparse graph. It should also be noted that, although manipulating and storing the edge-sparsed similarity graph can be done efficiently, the construction of such sparse graph might be still computationally expensive. Even with the smart speed-up techniques, such as k-nearest-neighbor graphs [Liaw et al., 2010, Moore, 1991], the actual time for building the full sparse graph can vary in practice.

All the methods mentioned above (except for b-matching) can be considered as the *algorithmic* frameworks that do not aim at optimizing any specific criterion. *Optimization-based* techniques, on the other hand, try to find a (sparse) similarity graph that optimizes an explicit objective function. For instance, *Locality Preserving Projection (LPP)* [Niyogi, 2004] and *Neighborhood Preserving Embedding (NPE)* [He et al., 2005] provide such basic objective functions for similarity graph construction. Furthermore, *Graph-optimized LPP (GoLPP)* [Zhang et al., 2010] is an immediate improvement of LPP by adding an entropy term to the objective function. However, none of these embedding methods explicitly enforces sparsity on the similarity graph. *Sparsity Preserving Projection (SPP)* [Qiao et al., 2010] and *Graph Optimization for Dimensionality Reduction with Sparsity Constraints (GO-DRSC)* [Zhang et al., 2012] explicitly enforce sparsity by introducing an $\ell 1$ penalty term in the objective.

3.2.3 Decomposition-based Methods

Decomposition-based techniques view the problem of constructing the similarity graph as a special case of the *Generalized N-body Problem* [Gray and Moore, 2000, Ram et al., 2009]. This problem, in a nutshell, considers the computation all the *mutual effects* among a set of source and target datapoints. For example, consider the problem of computing the node degrees of a similarity graph which is built using the Gaussian kernel in Eq. (3.1). This problem can be easily reduced to the *sums of Gaussians* problem [Raykar et al., 2005]. Similar to graph construction, this problem has the computational complexity of order $O(N^2)$. Viewing each datapoint (or node) as both a source and a target, the key idea in decomposition-based methods for reduction is to decompose the $O(N^2)$ mutual effects between the sources and the targets into $O(N)$ independent factors for each of the sources and the targets, and therefore reducing the computational order to $O(N)$. The most famous method of this kind is *Fast Gauss Transform (FGT)* which was first introduced in [Greengard and Strain, 1991]. An outstanding feature of FGT is that using FGT, the approximation error can be bounded. However, the constant factor in the computational order exponentially depends on the dimensionality of the space which makes it impractical to apply FGT to high-dimensional problems. *Improved Fast Gauss Transform (IFGT)* [Yang et al., 2003, Yang et al., 2005] addresses this problem by using Tylor expansion and k-center partitioning algorithm. The two main issues with IFGT is (a) its error bound is not tight and (b) there is not practical guideline for setting the parameters of the model. [Lang et al., 2005] proposes a non-optimal technique to set the parameters. [Raykar et al., 2005] has improved the IFGT framework so that the parameters are set optimally and the error bound is tight. Moreover, [Lee and Gray, 2008] proposed a variation of FGT in combination with Monte Carlo Multipole method.

3.2.4 Direct Methods

This class of methods goes around the problem of finding a sparse graph representation by trying to approximate the quantity of interest in the problem directly. For instance, manifold regularization techniques [Belkin et al., 2006, Tsang and Kwok, 2006] directly find a smooth function on the graph. For the task of eigendecomposition, [Fergus et al., 2009, Nadler et al., 2006, Amizadeh et al., 2012b] have proposed methods that try to directly estimate the eigen decomposition of the underlying transition matrix in the limit assuming a factorized underlying distribution. Although these methods can be very effective in practice [Fergus et al., 2009], they completely rely on the assumption that the underlying input space is factorizable into a set of low-dimensional sub-spaces in each the sub-eigenfunctions can be estimated efficiently. Clearly, this assumption may not hold in many problems. Besides, these methods are inherently task specific.

3.2.5 Hierarchical Methods

The hierarchical methods try to take the advantage of the hierarchical cluster structure of the data to efficiently reduce the similarity graph. Using the same notion of source and target as in decomposition-based methods, the first group of hierarchical methods are categorized as *single-tree* algorithms where a single tree is built over sources and the the sum effect is computed for each target node individually. kd-tree methods [Friedman et al., 1977, Moore, 1991], for example, have been developed to compute the k-nearest neighbors for each target node in the logarithmic time. Similar algorithms (e.g. [Karger and Ruhl, 2002, Krauthgamer and Lee, 2004]) provide logarithmic-time per query framework for finding the nearest neighbors. However, none of these method provide any error guarantee. Using other structures like *cover trees* [Beygelzimer et al., 2006], one can control the trade-off between the accuracy and the efficiency [Ram et al., 2009].

The *dual-tree* methods [Gray and Moore, 2000, Gray and Moore, 2003, Ram et al., 2009] are the direct generalization of their single-tree counterparts. Using these methods, in addition to the source nodes, one tree is also built for the target nodes. Using these two trees, one can model the mutual effect between the source and the target nodes at different levels of the hierarchy. More recently, [Lee et al., 2011] have proposed a combination of dual-tree methods with Fast Gauss Transform (FGT). Moreover, [Thiesson and Kim, 2012, Thiesson and Wang, 2010] have used an *Anchor tree-based* dual-tree approach for fast mode-seeking clustering and mixture modeling.

It should be noted that dual-tree methods are closely related to multi-level low-rank approximation techniques [Wang and Dong, 2012]. For computing the transition matrix of the random walk in specific, these methods (e.g. [Chennubhotla and Jepson, 2005, Lin, 2003]) learn a hierarchical low-rank representation of the transition matrix without using an explicit notion of a hierarchical tree. The hierarchical low-rank structure of the resulted matrix can be further exploited for efficient eigen-decomposition [Chennubhotla and Jepson, 2005] which is in the core of the spectral methods.

Our proposed Variational Dual-Tree (VDT) framework in this chapter belongs to the class of dual-tree methods. The VDT framework provides a non-parametric methodology to approximate and represent the transition matrix of the random walk on similarity graphs in $O(N^{1.5}\log N)$. One big advantage of this framework to other tree-based methods is that it directly computes the transition matrix without computing the intermediate similarity matrix $\mathbf{W}$. Another advantage of VDT is that given a distance computation of $O(1)$ between any two datapoints, the overall computational complexity of VDT does not depend on the dimensionality d of the input space.

3.3 CASE STUDY: SUBSAMPLING IN THE TERM SPACE

In Chapter 2, we considered the problem of learning generalized text similarity metrics from the data-driven co-occurrence graph $\mathscr{A}$ (aka the association graph) and showed its merits compared to some well-known techniques in the literature. As mentioned there, although the graph-based methodology introduces an elegant framework to model the text similarity, its performance can highly degrade as the number of terms in the space grows large. In particular, if the number of terms N is large, computing the eigen decomposition over the entire term space (in principle an operation of $O(N^3)$ time) becomes computationally demanding, even if it is done offline and only once. As a result, in order to work with large-scale term spaces, we have no choice other than applying one of the reduction methods described in the previous section. However, in this specific problem, the terms are not explicitly represented in a feature space ruling out some reduction methods such as tree-based methods and Fast Gauss Transform. Therefore, to address the problem, we propose the following two-step solution.

3.3.1 Building The Association Graph

Same as before, we build the association graph $\mathscr{A}$ based on the co-occurrence of the terms in sentences by scanning over the whole corpus once. Note that since the co-occurrence matrix is generally sparse, we use the sparse matrix representation for $\mathscr{A}$ so that we will not have a memory issue. In terms of the computational complexity, due to the specific construction method in this problem, the order will be $O(\bar{d}N)$, where $\bar{d}$ is the average node degree in the association graph and we expect $k \ll N$. As a matter of fact, this form of representation and construction can be put in the edge-sparsification category explained above.

3.3.2 Eigen Decomposition of The Association Graph

Although $\mathscr{A}$ is represented as a sparse matrix, its eigen decomposition might still be computationally demanding. Therefore, we propose to keep only a random subsampled sub-graph of size M for the eigen decomposition purposes. By doing so, we effectively lose the values of the eigenvectors for those nodes (terms) that are not included in the sample; however, these values are essential for mapping those terms into the transformed metric space as well as computing the similarity metric. To recover these values, we need a *(semi-)inductive* method to map the nodes excluded from the graph into the metric space. This is not a trivial task as the graph-based methods are inherently transductive. Nevertheless, we can always come up with some efficient approximations. A good candidate approximation technique is in fact the Nyström approximation as explained in the previous chapter. Using this approximation, the total time reduces to $O(M^3 + \bar{d}N)$, where $\bar{d}$ is again the average node degree in $\mathscr{A}$. This is a significant improvement if $m \ll n$.

3.3.3 Empirical Error for Ranking

Obviously, the eigen decomposition of a smaller graph and its Nyström-based expansion to all nodes define an approximation of the true metric. Now, we have to show how much this approximation affects the final distance metric. However, in many real applications including ranking, it is the ordering over the terms induced by the distance metric that really matters and not the actual distance values. Therefore, we can only measure how many misplacements (in %) are introduced using the proposed approximation compared to the ordering induced in the exact case (i.e. the gold standard order). Table 5 shows the (normalized) number of misplacements introduced in a test set of 100 terms using different sample sizes M for a term space and the graph of size $N = 5000$ we used for the analysis of PubMed articles in Section 2.4. The experiment is repeated 5 times for each sample size and the results are averaged. As the table shows, even for 20% of the terms ($M = 1000$), 16% misplacements are introduced in average which means that the approximation is pretty robust.

m	**Avg % of misplacements**	**std deviation**
20%	16.0%	0.011
40%	13.7%	0.005
60%	8.9%	0.012
80%	4.4%	0.002

Table 5: The average number of misplacements for different sample size m with standard deviation.

3.4 LARGE-SCALE TRANSITION MATRIX APPROXIMATION

In the previous section we showed how the subsampling technique can help to build the similarity graph for large-scale datasets. Nevertheless, in most cases, the similarity graph by itself does not produce the desired solution for the problem in hand. Instead, one usually defines a stochastic process on this graph and/or a secondary quantity derived from this graph to capture the desired data-driven similarity among the datapoints. To this end, one common technique is to define a *random walk* on the similarity graph as described in Section 1.2.3.1. The random walk implicitly encodes the cluster structure of the input space in terms of the data-driven similarity metric [von Luxburg, 2007]. As shown in Chapter 1, many famous methodologies in graph-based data analysis take this approach: Diffusion maps [Nadler et al., 2006, Lafon and Lee, 2006, Coifman et al., 2008, Coifman et al., 2005a, Coifman et al., 2005b, Lee and Wasserman, 2010, Amizadeh et al., 2012b], Laplacian eigenmaps [von Luxburg, 2007, Belkin and Niyogi, 2002, Belkin and Niyogi, 2005], Label Propagation algorithm [Zhou et al., 2003, Belkin et al., 2006] etc. are all examples of such frameworks. To represent a random walk on a graph, one needs to maintain the transition probabilities potentially between any pair of nodes in the graph. This means that if the graph has N nodes (i.e. N datapoints), one would need to compute and maintain roughly N^2 probability numbers. These probabilities are typically stored in the form of a N by N matrix, known as the *transition matrix*. For those problems with N in the the order of 10^4 or larger, computing and storing the transition matrix quickly become infeasible. The most common solution to approach this problem is to impose *sparsity* on the similarity graph. A sparse graph would clearly solve the memory problem, but the computation of a sparse solution can be still as expensive as $O(N^2)$ or even more.

Looking at the big picture, the problem of computing the transition matrix on a large similarity graph is an instance of the classical N-*body* problems. Introduced in the Computational Physics [Barnes and Hut, 1986, Greengard, 1994, Carrier et al., 1988], N-body problems are the problems whose exact solutions involve computing the mutual interactions between *all* pairs of datapoints in a dataset. One classical example in Particle Physics is computing the forces/charges among N particles in the space. In Machine Learning also, many computationally intensive problems such as kernel density estimation, nearest neighbor classification, outlier detection, etc can be seen as special cases of N-body problems [Gray and Moore, 2000, Ram et al., 2009]. To address these problems, many methods have been proposed in Machine Learning and Computational Physics literature, of which tree-based hierarchical methods [Gray and Moore, 2003, Moore, 2000, Thiesson and Kim, 2012, Thiesson and Wang, 2010, Lee et al., 2011, Beygelzimer et al., 2006, Liaw et al., 2010], fast multipole methods [Lee and Gray, 2008, Greengard, 1994, Carrier et al., 1988] and fast Gauss transform [Greengard and Strain, 1991, Yang et al., 2003, Raykar et al., 2005, Yang et al., 2005, Lee et al., 2011] are among the famous ones. A subset of these methods can be applied to N-body problems in graph-based methods which we described in Section 3.2.

In the rest of this chapter, we propose the *Variational Dual-tree (VDT)* framework to efficiently compute the transition matrix of random walk on large-scale similarity graphs. This framework, which is based upon the method proposed by [Thiesson and Wang, 2010, Thiesson and Kim, 2012] for fast kernel density estimation, combines the dual-tree hierarchical partitioning of the input space [Thiesson and Kim, 2012, Thiesson and Wang, 2010, Lee et al., 2011] with variational approximation to approximate the transition matrix with reduced number of parameters. It should be emphasized that our proposed framework is specifically designed to approximate the transition matrix whereas all the aforementioned methods including [Thiesson and Wang, 2010, Thiesson and Kim, 2012] are designed to solve other N-body problems specifically kernel density estimation. To efficiently compute the transition matrix of the random walk, the most obvious solution is to build a sparse k-nearest-neighbor (kNN) similarity graph using a N-body fast method like [Moore, 1991] and then compute the transition probabilities on the sparse graph. Our proposed method, on the other hand, *directly* computes the transition probabilities without the need to build the similarity graph. However, the difference between VDT and kNN frameworks is more fundamental: the underlying reduction idea in the VDT framework is *parameter sharing*, whereas for the kNN method, it is *sparsification*. Both of these ideas work in theory; however, as we have shown in our experiments, sparsification can result in disconnectivity in practice, which in turn causes unstable results for certain end applications such as label propagation. Our method, on the other hand, always produces a connected underlying graph which makes it a robust solution for the mentioned applications. Moreover, we develop a fast multiplication algorithm for our model to be used in further applications of the transition matrix like label propagation and eigen-decomposition.

The other important contribution in this chapter is the extension of the VDT framework to non-Euclidean spaces. In almost all of the methods mentioned above, an unwritten assumption is to use the Euclidean distance as the distance measure between the datapoints and the Gaussian kernel as the kernel for density estimation (or similar tasks). Unfortunately, many of these methods have no other choice because their very speed-up ideas directly depend on the usage of the Euclidean distance and the Gaussian kernel. For instance, the proposed method in [Thiesson and Wang, 2010, Thiesson and Kim, 2012] relies on the special form the Euclidean distance to efficiently solve the variational optimization, or the fast Gauss transform methods [Greengard and Strain, 1991, Yang et al., 2003, Raykar et al., 2005, Yang et al., 2005] are designed to be exclusively used with the Gaussian kernel. However, for many real datasets, these choices are not simply the best ones. One can easily imagine a dataset that is generated through some non-Gaussian process in a non-Euclidean space. To address this problem, we extend our proposed framework to support the class of *Bregman divergences* which covers many well-known distances and divergences in Machine Learning including the Euclidean distance. We call this general framework the *Bregman Variational Dual-tree (BVDT)* framework. Through the connection between the Bregman divergences and the exponential families, we show that BVDT also generalizes the usage of the Gaussian kernel to general exponential family kernels. More importantly, we show that due to the special form of the general Bregman divergences, BVDT also enjoys the speed-up ideas in the original VDT with no further cost and yet dramatically improves the accuracy of the framework for certain problems. In particular, for the task of document classification using random walk on the document graph, the BVDT framework significantly outperforms the original VDT method while maintaining the same computational complexity as shown in our experiments. Finally, we show that VDT is in fact a special case of BVDT.

The rest of this chapter is organized as follows: in Section 3.4.1, we formally define the problem of computing transition matrices (and subsequently random walk) on similarity graphs. Section 3.5 gives a thorough overview of the tree-based hierarchical methods for fast kernel density estimation. In particular, we describe the dual-tree variational approximation method proposed by [Thiesson and Wang, 2010, Thiesson and Kim, 2012] in more details. The reader should note that, for our further purposes, we have slightly changed the presentation of this method compared to that of the original papers [Thiesson and Wang, 2010, Thiesson and Kim, 2012]. In Section 3.6, we present our proposed VDT framework for approximation of the transition matrix. Section 3.7 illustrates the Bregman extension of the proposed VDT framework, namely BVDT. To evaluate our proposed models, we have run multiple experiments on different datasets; Section 3.8 demonstrates the results of these experiments. Finally, we present and discuss some conclusion remarks in Section 3.9.

It should be noted that the proposed framework in this section have been developed in collaboration with Bo Thiesson, Associate Professor at Aalborg University [Amizadeh et al., 2012a, Amizadeh et al., 2013].

3.4.1 The Problem Statement

Let $\mathscr{D} = \{x_1, x_2, \dots, x_N\}$ be a set of i.i.d. datapoints in $\mathscr{X} \subseteq \mathbb{R}^d$. The similarity graph $\mathscr{G} = \langle \mathscr{D}, \mathscr{D} \times \mathscr{D}, \mathbf{W} \rangle$ is defined as a *complete* graph whose nodes are the elements of $\mathscr{D}$ and the matrix $\mathbf{W} = [w_{ij}]_{N\times N}$ represents the edge weights of the graph, where

$$w_{ij} = k(x_i, x_j; \sigma) = \exp(-\|x_i - x_j\|^2 / 2\sigma^2) \tag{3.1}$$

is the similarity weight between nodes x_i and x_j. The closer x_i and x_j are in terms of the Euclidean distance $\|x_i - x_j\|$, the higher the similarity weight w_{ij} will be. The bandwidth parameter σ is a design parameter that essentially determines how sparse the graph should be. By abstracting the input space into a graph, the similarity graph essentially captures the geometry of the data in the input space.

Once the similarity graph is constructed, one can define a random walk on it. The transition probability distributions are in this case given by the transition matrix $\mathbf{P} = [p_{ij}]_{N\times N}$, where

$$p_{ij} = \frac{w_{ij}}{\sum_{k\neq i} w_{ik}} = \frac{k(x_i, x_j; \sigma)}{\sum_{k\neq i} k(x_i, x_k; \sigma)} \tag{3.2}$$

is the probability of jumping from node x_i to node x_j. Note that the elements in each row of $\mathbf{P}$ sum up to 1 so that each row is a proper probability distribution. As mentioned before, the transition matrix is the fundamental quantity used by many graph-based Machine Learning frameworks.

In terms of computational resources, it takes $O(N^2)$ CPU and memory to construct and maintain the transition matrix, which can be problematic when the number of datapoints, N, becomes large. Dealing with large number of datapoints is, in fact, quite typical in many real-world datasets. On the other hand, the problem is not only restricted to the computation of the transition matrix; it is one of the most significant challenges faced by many graph-based methods. To overcome this challenge, the key idea is to somehow *reduce* the representation of the graph (or $\mathbf{P}$ in our problem).

3.5 FAST KERNEL DENSITY ESTIMATION

The tree-based algorithms for fast Kernel density estimation (KDE) form the foundations of our proposed framework in this chapter; therefore, in this section, we review the KDE problem and elaborate on the tree-based approximation methods proposed to solve this problem.

Let $\mathscr{M} = \{m_1, m_2, \ldots, m_M\}$ denote a set of kernel centers in $\mathbb{R}^d$. The kernel density estimate for any data point $x \in \mathbb{R}^d$ is then defined by the following weighted sum or mixture model:

$$p(x) = \sum_j p(m_j) p(x|m_j) \tag{3.3}$$

where $p(m_j) = \frac{1}{N}$, $p(x|m_j) = \frac{1}{Z_\sigma} k(x, m_j; \sigma)$, and k is a kernel profile with bandwidth σ and normalization constant Z_σ. For the commonly used Gaussian kernel, $k(x, m_j; \sigma) = \exp(-\|x - m_j\|^2/2\sigma^2)$ and $Z_\sigma = (2\pi\sigma^2)^{\frac{d}{2}}$.

Now let us assume we want kernel density estimates for the observed datapoints $\mathscr{D} = \{x_1, x_2, \ldots, x_N\}$ in $\mathbb{R}^d$ that also define the kernel centers. That is, $\mathscr{D} = \mathscr{M}$, and we use the $\mathscr{D}$ and $\mathscr{M}$ notation to emphasize the two different roles every example x_i takes - one as a *query* datapoint (denoted by x_i), the other as a *reference* kernel center (denoted by m_i). Let us exclude the kernel centered at m_i from the kernel density estimate at that datapoint. In this case, the estimate at $x_i \in \mathscr{D}$ can be defined as the $N-1$ component mixture model:

$$p(x_i) = \sum_{j \neq i} p(m_j) p(x_i|m_j) = \sum_{j \neq i} \frac{1}{N-1} \frac{1}{Z_\sigma} k(x_i, m_j; \sigma), \tag{3.4}$$

where $p(m_j) = 1/(N-1)$.

In some Machine Learning methodologies such as density estimation [Gray and Moore, 2003, Thiesson and Wang, 2010] and mode-seeking clustering [Cheng, 1995, Thiesson and Kim, 2012], one is interested in computing the log-likelihood of the observed data using KDE. In the mathematical terminology, this will translate to computing

$$\log p(\mathscr{D}) = \sum_i \log p(x_i) = \sum_i \log \sum_{j \neq i} p(m_j) p(x_i \mid m_j) \tag{3.5}$$

The complexity of computing Eq. (3.5) is $O(N^2)$, which is prohibitive for large-scale datasets with large N. There are many approximate techniques proposed in the literature to address this problem. Here, we direct our attention to tree-based methods.

3.5.1 Single-tree Approximation

Let the kernel centers in $\mathscr{M}$ be clustered into K clusters, then the effects of all the kernel centers belonging to the same cluster on computing the KDE for a query point x can be *grouped* together and approximated by one number. This is the idea that, for example, has been used in the Fast Gauss Transform (FGT) method [Raykar et al., 2005]. The computational order in this case is roughly $O(NK)$ where the constant coefficient depends on the dimensionality as well as the desired approximation accuracy. We refer to this method as *flat-clustering approximation (FCA)*. If the number of clusters grows faster than $\log N$, FCA can be still costly.

A direct generalization of the method described above is the *single-tree approximation (STA)* [Gray and Moore, 2003]. In this method, instead of having a flat clustering, one has a cluster binary tree on the kernel centers called the *kernel tree*. The kernel tree can be built using one of the fast approximate hierarchical clustering algorithms such as Anchor Tree method, as we will describe in Section 3.5.3. For the case $\mathscr{D} = \mathscr{M}$ where query points are the same as the kernel centers, we have the query point x to be one of the leaves in the kernel tree. Figure 3(A) demonstrates a simple example of computing KDE for the query point x using 5 kernel points other than x itself. In this case, one needs to compute 5 parameters (or cell). Figure 3(B) shows the STA method for this computation which has reduced the number of parameters down to 3. If we follow the path from the root of the tree down to the given query point x (i.e. the shaded path in the figure) and *cut out* the *other* sibling subtree at each node on the path as shown by the crosses in the figure, we will end up with L *disjoint* subtrees, where L is the length of the path. For our simple example, we have $L = 3$. In effect, this process generates a partitioning of the kernel centers into L disjoint clusters according to x which can be used to reduce the complexity of computing KDE for x from $O(N)$ to $O(L)$. It should be noted that computing the log-likelihood in Eq. (3.5) using STA requires us to repeat the described process for each query point independently which takes $O(NL)$ time in total.

The main advantages of the STA over the FCA are as follows. First, as opposed to the FCA techniques, the kernel center grouping is not fixed in STA and can be different depending on the location of the query point. In particular, for the kernel centers located closer to the query point, the clusters are finer while for the farther centers, the clusters are coarser. This can be easily verified for our simple example in Figure 3(B). This way, the quality of approximation can be improved without increasing the number of clusters. Second, due to the existence of a cluster hierarchy, the quality of approximation can be further improved by splitting the bigger clusters into smaller ones using the tree structure. Third, for a fairly balanced cluster tree, the average length of a path from the root to a leaf node is $\log N$. Therefore, given the tree, Eq. (3.5) can be computed roughly in $O(N \log N)$.

3.5.2 Dual-tree Approximation

In *dual-tree approximation (DTA)* methods, in addition to the kernel centers, the query points are also clustered together. In other words, the effects of a group of kernel centers on a group of query points are approximated by one number. As a result, in DTA techniques, we also need a cluster tree for the query points. In the case of $\mathscr{D} = \mathscr{M}$, we can use one tree for both sets $\mathscr{D}$ and $\mathscr{M}$. Figure 4 shows a simple example of DTA on a dataset of size 6. All the mutual effects between the kernel centers and the query points can be organized into a matrix as shown in Figure 4(A). Consider the shaded submatrix in Figure 4(A); this submatrix contains all the mutual effect between 4 query points and 2 kernel centers; that is, 8 distinct parameters. Now, if the 4 query points form the leaves of a subtree in the query

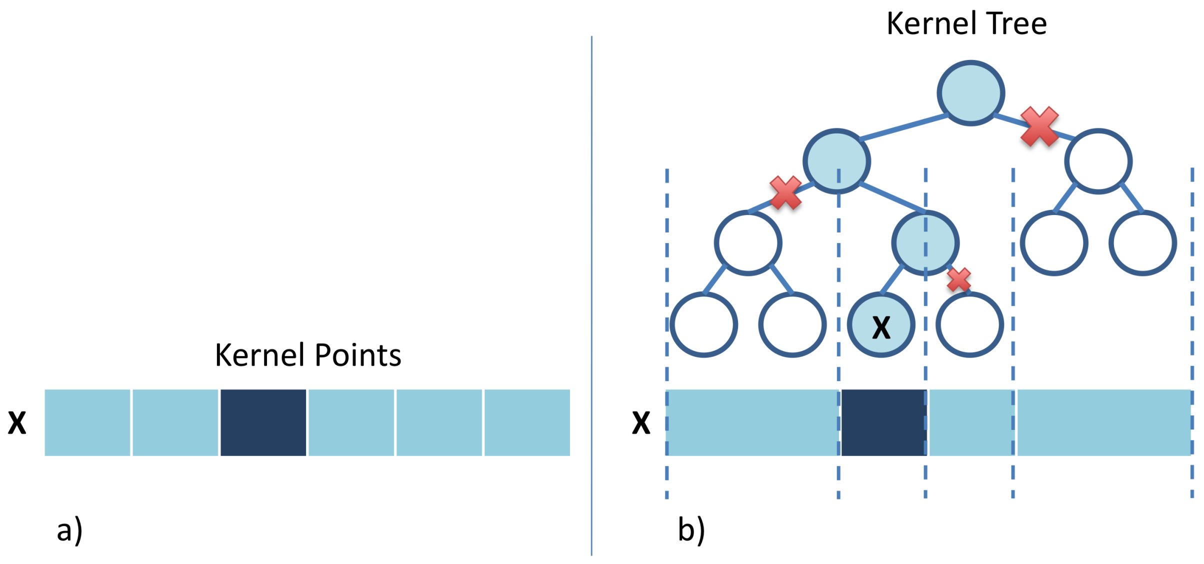

Figure 3: (A) Direct computation of KDE for the query point x (B) Single-tree approximation of KDE for the query point x.

cluster tree (i.e. the subtree A in the figure) and the 2 kernel centers also form the leaves of another subtree in the kernel cluster tree (i.e. the subtree B in the figure), DTA can be used to approximate all those 8 mutual effects between these two groups with only one number as shown in 4(B). This idea can be used for other query points and kernel center as well leading to dramatic reduction in the number of parameters. We will discuss this partitioning idea in more details later in this section.

Back to our problem, for computing Eq. (3.5), two recently proposed dual-tree-based frameworks are Dual-tree Fast Gauss Transform [Lee et al., 2011] which is based on Fast Gauss Transform approximation and Variational Dual-tree [Thiesson and Wang, 2010, Thiesson and Kim, 2012] which is based on Variational Approximation. Our proposed framework in this chapter for transition matrix approximation is closely related to the latter work. Therefore, in this section, we illustrate the basic elements of the variational dual-tree framework. Interested readers may refer to [Thiesson and Wang, 2010, Thiesson and Kim, 2012] for further details.

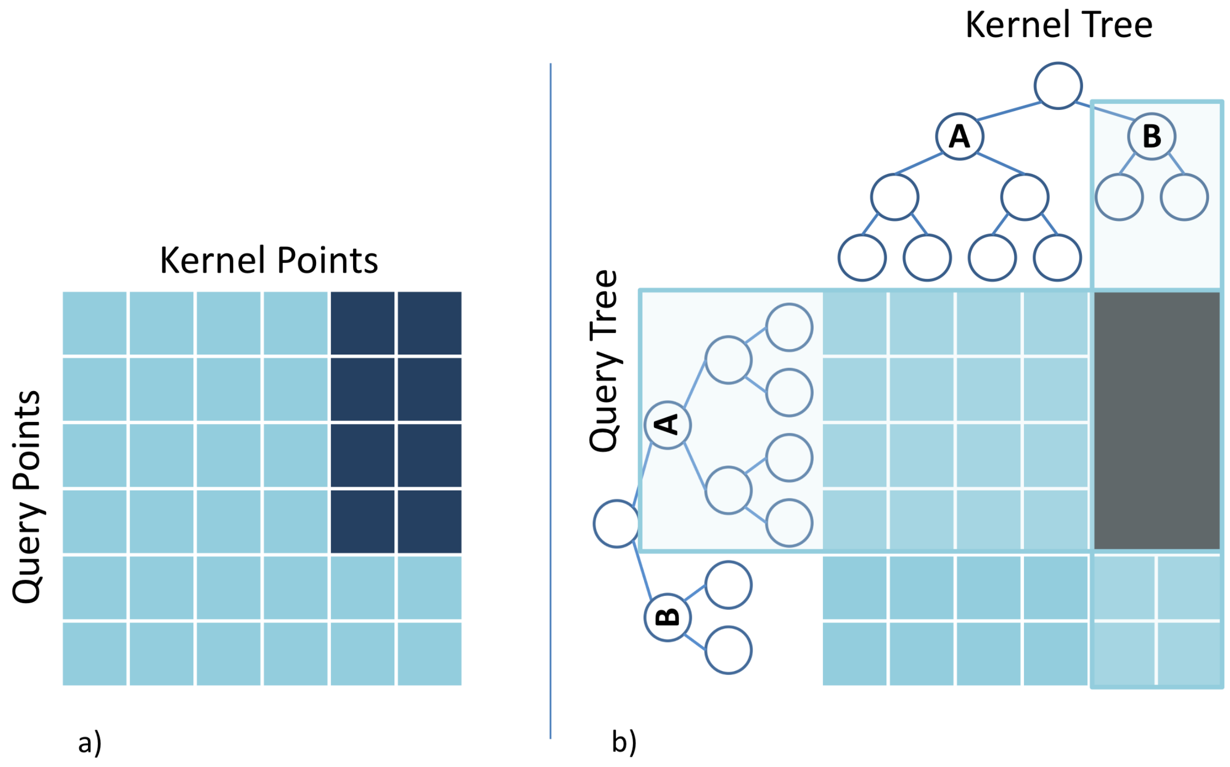

Figure 4: (A) All the mutual interactions between query points and kernel centers in a dataset of size 6 (B) Dual-tree approximation of the mutual effects between the two subtrees.

3.5.2.1 Variational Log-likelihood As mentioned before, in dual-tree-based techniques for computing the log-likelihood in Eq. (3.5), the query points are grouped together as well as the kernel centers. By taking a closer look at Eq. (3.5), we see that the inner sum in the log-function is over the kernel centers and can be easily decomposed into smaller summation terms based on the grouping of the kernel centers. The same is true for the outer sum which is over the query points; that is, one can decompose the outer sum into smaller summation terms based on the grouping of the query points. However, due to the existence of the log-function in between the two summations, the simultaneous decomposition of the *kernel-sum* and the *query-sum* is not possible. This is specifically problematic because in dual-tree methods, we need to be able to decompose both summation terms simultaneously. This problem can be addressed by pulling the inner sum out of the log-function using the Jensen inequality and introducing the *variational parameters*:

$$\begin{aligned}\log p(\mathscr{D}) &= \sum_i \log p(x_i) = \sum_i \log \sum_{j\neq i} p(m_j)p(x_i \mid m_j) \\ &= \sum_i \log \sum_{j\neq i} \frac{q_{ij}}{q_{ij}} p(m_j)p(x_i \mid m_j) \\ &\geq \sum_i \sum_{j\neq i} q_{ij} \log \frac{p(m_j)p(x_i \mid m_j)}{q_{ij}} \triangleq \ell(\mathscr{D}) \end{aligned} \tag{3.6}$$

where, q_{ij}'s $(i,j \in \{1,\dots,N\})$ are free variational parameters with the following constraints:

$$\forall i \in \{1,\dots,N\} : \sum_{j=1}^{N} q_{ij} = 1, 0 < q_{ij} \leq 1, q_{ii} = 0 \tag{3.7}$$

Eq. (3.6) defines a *variational lower-bound* on the log-likelihood. For further purposes, we re-arrange the variational parameters q_{ij}'s $(i,j \in \{1,\dots,N\})$ into an $N \times N$ matrix $\mathbf{Q} = [q_{ij}]_{N\times N}$. By doing some algebra, $\ell(\mathscr{D})$ can be written as:

$$\ell(\mathscr{D}) = \log p(\mathscr{D}) - \sum_i D_{KL}\big(q_{i\cdot} \| p(\cdot|x_i)\big) \tag{3.8}$$

where $p(\cdot|x_i) = \{p(m_j|x_i) \mid 1 \leq j \leq N\}$ is the posterior membership probability distribution for the query point x_i, $q_{i\cdot} = \{q_{ij} \mid 1 \leq j \leq N\}$ is the i-th row of matrix $\mathbf{Q}$ and $D_{KL}(\cdot||\cdot)$ is the KL-divergence between two distributions. Based on Eq. (3.8), the gap between the variational lower-bound and the actual log-likelihood is equal to the difference between the

the posterior membership distributions and the *variational distributions* (defined by the rows of **Q**) in terms of the KL-divergence. In other words, the variational parameter q_{ij} *approximates* the membership posterior $p(m_j|x_i)$. Moreover, if no constraint on q_{ij}'s other than the ones in Eq. (3.7) is enforced, by setting $q_{ij} = p(m_j|x_i)$, the gap will become zero; that is, the lower-bound in Eq. (3.6) is tight.

Because of the problem with the summations in Eq. (3.5) mentioned above, one should compute the lower-bound in Eq. (3.6) instead of the actual likelihood in Eq. (3.5) if a dual-tree method is to be used. However, the variational matrix **Q** introduces $N^2 - N$ [1] distinctive parameters and therefore makes it impossible to simultaneously decompose Eq. (3.6) in terms of the query and the kernel groupings. One simple solution is to set all the parameters to a fixed value while satisfying the constraints in Eq. (3.7); that is, $q_{ij} = 1/(N-1)$. The problem with this solution is the approximation gap can be pretty large depending on the data and the final approximation will *not* be optimum. A smarter solution is to simultaneously group the variational parameters as well based on the query and the kernel groupings and force the parameters in a group to take the same value. This is referred to as the *dual-tree block partitioning* of matrix **Q** and will be discussed in the following section.

3.5.2.2 Dual-tree Block Partitioning The dual-tree-based variational approach introduced in [Thiesson and Kim, 2012] maintains two cluster trees; the *query partition tree* and the *kernel partition tree*, that hierarchically partition the query points and the kernels into disjoint subsets, such that an intermediate node in a tree represents a subset of the query points or the kernels and leaves correspond into singleton sets. In this work we assume the structure of the two trees is identical, leading to the exactly same subsets of query points and kernels represented by the tree.

The main reason for introducing the partition tree is to define relations and permit inferences for groups of related query points and kernels without the need to treat them individually. More specifically, we use the partition tree to induce a *block partition* of the matrix **Q**, where all the parameters q_{ij} within a block are forced to be equal. We also refer to this partition as *block constraints* on **Q**.

[1]Note that the diagonal elements of **Q** are always set to 0.

Formally, a *valid* block partition $\mathscr{B}$ defines a mutually exclusive and exhaustive partition of $\mathbf{Q}$ into blocks (or sub-matrices) $(A,B) \in \mathscr{B}$, where A and B are two *non-overlapping* subtrees in the partition tree. That is, $A \cap B = \emptyset$, in the sense that query-leaves in the subtree under A and kernel-leaves in the subtree under B do not overlap. (To maintain the convenient matrix representation of $\mathbf{Q}$, the singleton blocks representing the neutral diagonal elements in $\mathbf{Q}$ are added to this partition.) Figure 5(A) shows a small example with a valid block partition for a partition tree built for six data points (kernels). This block partition will, for example, enforce the block constraint $p_{13} = p_{14} = p_{23} = p_{24}$ for the block $(A,B) = (1-2,3-4)$ (where $a-b$ denotes a through b).

With the formal definition of a block partition, the block constraints can be mathematically expressed as:

$$q_{ij} = q_{AB} \text{ s.t. } (A,B) \in \mathscr{B}, x_i \in A, m_j \in B \tag{3.9}$$

Where q_{AB} is a single variational parameter associated with the block $(A,B) \in \mathscr{B}$. We refer to the set of q_{AB}'s as the set of *block parameters* and denote it by $Q_{\mathscr{B}} = \{q_{AB} \mid (A,B) \in \mathscr{B}\}$.

To represent the block-constrained $\mathbf{Q}$ matrix more compactly, we utilize the so-called *marked partition tree (MPT)* that annotates the partition tree by explicitly linking query groups to kernel groups in the block partition: for each block $(A,B) \in \mathscr{B}$, the query-node A is marked with the matching kernel-node B. Each node A in the MPT will therefore contain a, possibly empty, list of marked B nodes. We will denote this list of marks as $A_{mkd} \triangleq \{B | (A,B) \in \mathscr{B}\}$. Figure 5(B) shows the MPT corresponding to the partition in Figure 5(A). For example, the mark of kernels $B = 3–4$ at the node representing the data $A = 1–2$ corresponds to the same block constraint $(A,B) = (1-2,3-4)$, as mentioned above. It is the only mark for this data node, and therefore $A_{mkd} = 1-2_{mkd} = \{3-4\}$. As another example, the list of marks for node $A = 5$ has two elements; $A_{mkd} = 5_{mkd} = \{5,6\}$. An important technical observation, that we will use in the next section, is that each path from a leaf to the root in the MPT corresponds to the row indexed by that leaf in the block-partitioned matrix $\mathbf{Q}$. Moreover, the block parameters are stored at the marks in the MPT.

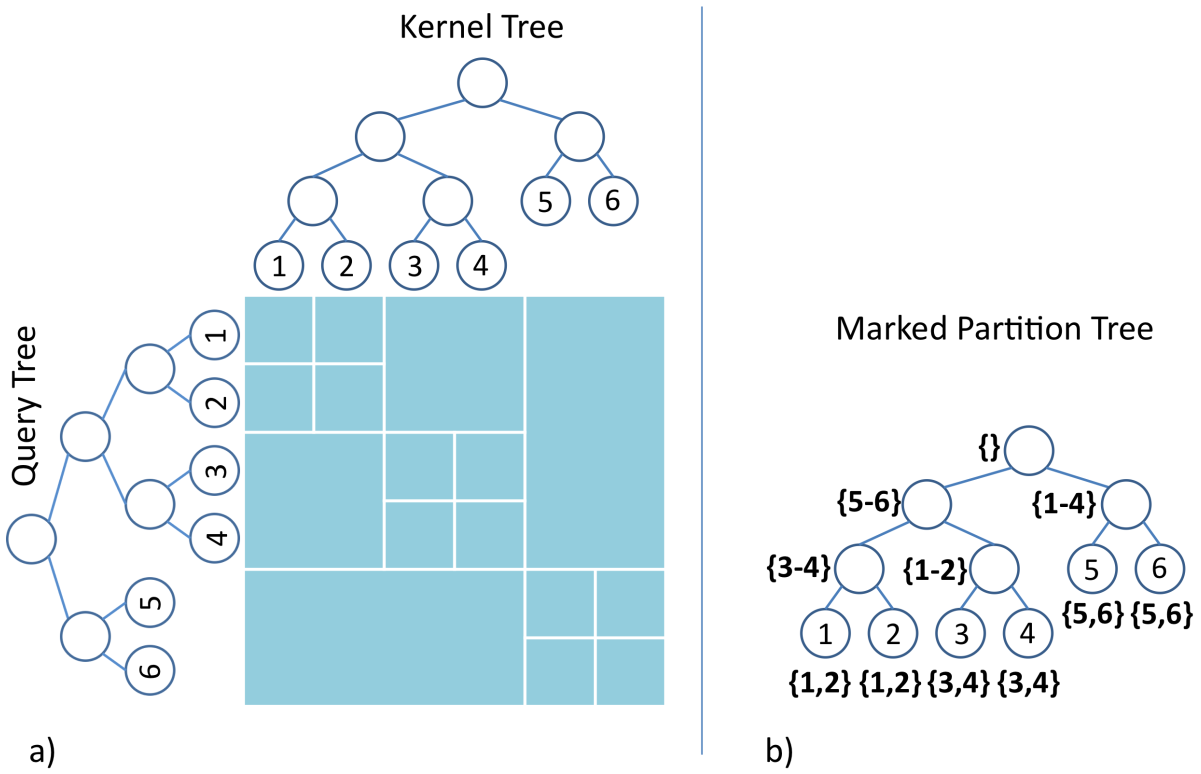

Figure 5: (A) A block partition (table) for query partition tree (left) and the identical kernel partition tree (top). (B) MPT representation of the block partition.

Clearly, there are more than one valid partitions of $\mathbf{Q}$; in fact, any further *refinement* of a valid partition results in another valid partition with increased number of blocks. We postpone the discussion of how to choose an initial valid partition and how to refine it to Section 3.6.4. For now, let us assume that we are given a valid partition $\mathscr{B}$ of $\mathbf{Q}$ with $|\mathscr{B}|$ number of blocks. By enforcing the block constraints in Eq. (3.9) for all parameters in a given block, the number of parameters in $\mathbf{Q}$ is effectively reduced from $N^2 - N$ to $|\mathscr{B}|$ (i.e. the number of blocks or equivalently block parameters).

3.5.2.3 Variational Optimization

Given a block partitioning $\mathscr{B}$ of $\mathbf{Q}$, one needs to set the block parameters $Q_{\mathscr{B}}$ before computing the lower-bound of the log-likelihood. However, these parameters cannot be set arbitrarily; first and foremost, they need to satisfy the constraints in Eq. (3.7). Furthermore, we want these parameters to take values such that the approximation gap (i.e. Eq. (3.8)) between the lower-bound and the actual log-likelihood is minimized. This is equivalent to maximizing the lower-bound in Eq. (3.6) with respect to the block parameters subject to the constraints in Eq. (3.7). To carry out this optimization, first we need to rewrite the lower-bound as well as the constraints in terms of the block parameters. In other words, we need to apply the block constraints in Eq. (3.9) into Eq. (3.6) and Eq. (3.7), resulting in:

$$\ell(\mathscr{D}) = c - \frac{1}{2\sigma^2} \sum_{(A,B)\in\mathscr{B}} q_{AB} \cdot D_{AB} - \sum_{(A,B)\in\mathscr{B}} |A||B| \cdot q_{AB} \log q_{AB}, \tag{3.10}$$

and,

$$\sum_{(A,B)\in\mathscr{B}(x_i)} |B| \cdot q_{AB} = 1, \forall x_i \in \mathscr{D} \tag{3.11}$$

where

$$\mathscr{B}(x_i) \triangleq \{(A,B) \in \mathscr{B} \mid x_i \in A\} \tag{3.12}$$

$$c = -N \log\left((2\pi)^{d/2} \sigma^d (N-1)\right)$$

$$D_{AB} = \sum_{x_i \in A} \sum_{m_j \in B} \|x_i - m_j\|^2. \tag{3.13}$$

Now, one can maximize Eq. (3.10) subject to the constraints in Eq. (3.11) to find the best values for the block parameters Q that minimize the approximation gap. [Thiesson and Kim, 2012][Algorithm 3] has developed a recursive algorithm that solves this optimization for all q_{AB}, $(A,B) \in \mathscr{B}$ in $O(|\mathscr{B}|)$ time. Once the block parameters are computed, the lower-bound in Eq. (3.10) can be computed in $O(|\mathscr{B}|)$ as an approximation to the true log-likelihood.

A very crucial element of the variational dual-tree framework is the way that the coefficients $D_{AB}, \forall(A,B) \in \mathscr{B}$ in Eq. (3.10) are computed; the direct computation of the double-sum for D_{AB} in Eq. (3.13) takes $O(|A| \cdot |B|)$ time. This means that the computation of D_{AB} for all blocks $(A,B) \in \mathscr{B}$ will take $O(N^2)$ time in total! This is clearly the antithesis of the original motivation behind the described approximation framework. Fortunately, this problem can be avoided thanks to the functional form of the Euclidean distance; in particular, D_{AB} can be written as:

$$D_{AB} = |A|S_2(B) + |B|S_2(A) - 2S_1(A)^T S_1(B), \tag{3.14}$$

where, $S_1(A) = \sum_{x \in A} x$ and $S_2(A) = \sum_{x \in A} x^T x$ are the *statistics* of subtree A. These statistics can be incrementally computed and stored while the cluster tree is being built in the Agglomeration phase of the anchor tree building algorithm (Algorithm 4); more specifically, if nodes A and B are being merged to form the parent node C; then, the statistics of node C are computed as: $S_1(C) = S_1(A) + S_1(B)$ and $S_2(C) = S_2(A) + S_2(B)$. Using these pre-calculated statistics, the set of coefficients $D_{AB}, \forall(A,B) \in \mathscr{B}$ are then computed in $O(|\mathscr{B}|)$.

The crux of the reformulation in Eq. (3.14) is the de-coupling of the *mutual interactions* between two clusters A and B so that the sum of mutual interactions can be computed using the sufficient statistics pre-calculated independently for each cluster. This reformulation is possible only because the distance metric incorporated in the framework is the Euclidean distance. Using other distance metrics can potentially introduce a very critical challenge to the variational dual-tree framework. We will discuss this problem in more details in Section 3.7.

Finally, given that we use the anchor tree method to build the cluster tree of the input dataset in $O(N^{1.5} \log N)$ in the worst case, the variational dual-tree framework will take $O(N^{1.5} \log N + |\mathscr{B}|)$ time and $O(|\mathscr{B}|)$ memory in total to approximate the log-likelihood of the data.

3.5.3 Anchor Trees

As mentioned in the previous sections, the tree-based methods need the hierarchical cluster representation of the data at the core of their approximation methodology. We assume this cluster hierarchy is represented as a binary tree $\mathscr{T}$ whose inner nodes represent the data clusters at different granularity scales and its leaves are the actual datapoints. If $\mathscr{T}$ is not given in advance, one needs to build it from the data using some hierarchical clustering algorithm. The exact bottom-up agglomerative clustering algorithm is of $O(N^3)$ time complexity and therefore is automatically omitted from the list of potential candidates. Fortunately, there exist some approximate techniques for cluster tree construction which are quite efficient in practice. Some well-known methods of this type include anchor tree [Moore, 2000], kd-tree [Moore, 1991] and cover tree [Ram et al., 2009, Beygelzimer et al., 2006] construction. In this work, we have used the anchor method, known to work well with (non-Euclidean) distance-related, low-dimensional as well as high-dimensional data with intrinsic structure. Here, we briefly describe the anchor tree algorithm and discuss its computational complexity. For a more detailed description of the algorithm, interested readers may refer to [Moore, 2000].

3.5.3.1 Anchor Tree Construction Algorithm

The anchor tree algorithm consists of three main consecutive steps: *anchor building, anchor agglomeration and recursion*. Anchor building starts by picking one of the datapoints at random (say y). Then, the algorithm sorts the rest of points based on their Euclidean distance to y. By doing so, the algorithm creates an *anchor* (or node) A with the *pivot* datapoint $A_p = y$ and the radius A_r that contains all the datapoints. A_r is set to the distance of the farthest point in A to y. Next, the algorithm creates a new anchor A^{new} by picking the farthest point from the pivot in A as the pivot of the new anchor. That is,

$$A_p^{new} = \arg\max_{x \in A} \|x - A_p\| \tag{3.15}$$

Then A^{new} *steals* all the points in A that are closer to the pivot of A^{new} than that of A; that is $\forall x : \|x - A_p^{new}\| < \|x - A_p\|$. Because the list of elements in an anchor is sorted with respect to $\|x_i - A_p\|$, a significant computational gain is achieved by not evaluating the remaining datapoints in the list once we discover that for the i-th datapoint in the list $\|x_i - A_p\| \leq d_{thr}$, where

$$d_{thr} = \|A_p^{new} - A_p\|/2, \tag{3.16}$$

This guarantees that the elements with index $j \geq i$ in the list are closer to their original anchor's pivot and cannot be stolen by the new anchor. When a new anchor is done stealing datapoints from older anchors, its list of elements is finally sorted. This process is repeated $\sqrt{N}$ times (by picking the point that is farthest from all anchor pivots at each step as the center of the new anchor) to create $\sqrt{N}$ anchors. Note that in the anchor building step, the goal is to cluster the datapoints into $\sqrt{N}$ such that each point is closer to the center of its cluster than to those of the other clusters. The k-means clustering algorithm pursues the same goal by trying to minimize the maximum radius of all anchors (clusters). Given enough number of re-initializations, k-means can find this optimum (if it exists), but very expensively in terms of the computational complexity. The algorithm used in the anchor building step (which is also referred as *farthest-point clustering algorithm* in the literature) is pretty fast but still an approximation. However, [Gonzalez, 1985] has shown that the solution found by farthest-point clustering is bounded:

Theorem 1. *The maximum radius of the partition found by farthest-point clustering is at most twice the optimal radius.*

A simple proof can be found in [Raykar et al., 2005].

Once the $\sqrt{N}$ anchors are created, the anchor tree construction now proceeds to *anchor agglomeration phase* that assigns anchors as leaf nodes and then iteratively merges two nodes that create a parent node with the *smallest* covering radius. This agglomerative bottom-up process continues until a hierarchical binary tree is constructed over the $\sqrt{N}$ initial anchors. Finally, recall that the leaves (i.e. the initial anchors) in this newly constructed tree contain $\sqrt{N}$ datapoints on average each. The whole construction algorithm is now recursively called in the *recursion phase* for each anchor leaf to grow it into a subtree. The recursion ends when the leaves of the tree contain only one datapoint each.

The pivot datapoint of an anchor is supposed to be the representative of that anchor. Because of this reason, we want the pivot to be the actual *mean* of the datapoints in the

Algorithm 1: Anchor Tree Construction

```
1: function CONSTRUCT($A$)
2:   Input: Anchor node $A$, initially contains all the datapoints
3:   Output: Anchor tree $\mathcal{T}$
4:   if $|A| = 1$ then
5:     $\mathcal{T} \leftarrow A$
6:   else
7:     $tList \leftarrow \{\}$
8:     $aList \leftarrow$ ANCHORBUILDING($A$)
9:     for all $B \in aList$ do
10:      $tList \leftarrow tList \cup \{$CONSTRUCT($B$)$\}$
11:    end for
12:    $\mathcal{T} \leftarrow$ AGGLOMERATE($tList$)
13:  end if
14:  return $\mathcal{T}$

15: end function
```

anchor. For this purpose, we alter the original construction algorithm by swapping the anchor agglomeration and the recursion steps. This way, the pivot of the parent anchor C of the anchors A and B can be computed incrementally in the anchor agglomeration phase as:

$$C_p = (|A| \cdot A_p + |B| \cdot B_p)/(|A| + |B|) \tag{3.17}$$

Furthermore, the radius of C is set to the minimum value that covers both anchors A and B; that is,

$$C_r = \max(\|A_p - C_p\| + A_r, \|B_p - C_p\| + B_r) \tag{3.18}$$

Algorithms 1 - 4 show the anchor tree construction algorithm explained above.

3.5.3.2 Time Complexity The construction time for the anchor-tree method is in the order of $O(N^{1.5} \log N)$ for a relatively balanced data set. To see this, we note that the anchor building step starts by creating $v = \sqrt{N}$ anchors, which will have approximately $O(\sqrt{N})$ data points in each. This step involves a sorting of all data points with order $O(N \log N)$ for each anchor resulting in $v \cdot O(N \log N)$ in the worst case where every time a new anchor is created, all the other points are absorbed (stolen) into the new anchor. This worst case can only happen when all the datapoints are equi-distance, typically in a very large dimensional space. However, in many problems where there is some intrinsic structure in the data, the average complexity of the anchor building step is much lower than the worst case.

Algorithm 2: Anchor Building

```
1:  function ANCHORBUILDING($A$)
2:    Input: Anchor node $A$
3:    Output: List of anchors $\mathscr{L}$
4:    $\mathscr{L} \leftarrow \{A\}$
5:    $M \leftarrow \sqrt{|A|} - 1$
6:    for $i = 1$ to $M$ do
7:      $A_p^{new} \leftarrow \arg\max_{B \in \mathscr{L}} \max_{x \in B} \|x - B_p\|$ // Picking the farthest point as the new pivot
8:      for all $B \in \mathscr{L}$ do
9:        STEALDATAPOINTS($A^{new}$,$B$)
10:     end for
11:     Sort all points $x \in A^{new}$ in decreasing order of $\|x - A_p^{new}\|$
12:     $A_r^{new} \leftarrow \max_{x \in A^{new}} \|x - A_p^{new}\|$
13:     $\mathscr{L} \leftarrow \mathscr{L} \cup A^{new}$
14:   end for
15:   return $\mathscr{L}$

16: end function
```

Once the v anchors are created, they are agglomeratively merged into a hierarchy in $O(v^2) = O(N)$ operations. Following, the algorithm is now recursively called for each anchor. Let $t(N)$ denote the time required to build an anchor tree from N points; we then have the following recurrence:

$$t(N) = v \cdot t(\sqrt{N}) + v \cdot O(N \log N) + O(v^2) \tag{3.19}$$

For $v = \sqrt{N}$, the solution of (3.19) is at most $O(N^{1.5} \log N)$. Recall that this guarantee is for a relatively balanced tree construction. The complexity will increase with the degree of unbalance in the tree. In a rare worst case of equi-distant data points (a completely structure-less dataset) a maximal unbalanced tree will force the complexity to $O(N^2)$.

Algorithm 3: Datapoint Stealing

```
1:  function STEALDATAPOINTS($A$,$B$)
2:    Input: Anchor nodes $A$ and $B$
3:    Output: $A$ steals datapoints from $B$
4:    $thr \leftarrow \|A_p - B_p\|/2$
5:    $x \leftarrow$ sorted members of $B$
6:    $i \leftarrow 1$
7:    while $\|x_i - B_p\| > thr$ do
8:      if $\|x_i - A_p\| < \|x_i - B_p\|$ then
9:        $B \leftarrow B \setminus \{x_i\}$
10:       $A \leftarrow A \cup \{x_i\}$
11:     end if
12:     $i \leftarrow i + 1$
13:   end while

14: end function
```

Algorithm 4: Agglomeration

```
1:  function AGGLOMERATE($tList$)
2:    Input: List of subtrees $tList$
3:    Output: Merged binary tree $\mathcal{T}$
4:    while $|tList| > 1$ do
5:      $(A^*, B^*) \leftarrow \arg\min_{A \in tList, B \in tList}(A_r + B_r + \|B_p - A_p\|)/2$
6:      $C.LeftChild \leftarrow A^*$, $C.RightChild \leftarrow B^*$
7:      $C_p = (|A^*| \cdot A_p^* + |B^*| \cdot B_p^*)/(|A^*| + |B^*|)$
8:      $C_r = \max(\|A_p^* - C_p\| + A_r^*, \|B_p^* - C_p\| + B_r^*)$
9:      $tList \leftarrow tList \cup \{C\} \setminus \{A^*, B^*\}$
10:   end while
11:   $\mathcal{T} \leftarrow tList(1)$
12:   return $\mathcal{T}$

13: end function
```

3.6 VARIATIONAL DUAL-TREE TRANSITION MATRIX APPROXIMATION

Having described the variational dual-tree framework for the fast approximation of the log-likelihood in the previous section, we are now ready to show how this framework can be utilized to solve the main problem in this chapter: approximating the transition matrix of the random walk on large-scale graphs.

3.6.1 Variational Random Walk

As we saw in Section 3.5.2.1, the posterior $p(m_j|x_i)$ models the membership probability of the query point x_i to the Gaussian kernel centered on m_j. However, $p(m_j|x_i)$ can be interpreted from a completely different view as the probability of jumping from datapoint x_i to datapoint m_j in the random walk defined on $\mathcal{G}$; that is $p(m_j|x_i)$ is equal to the p_{ij} element of matrix $\mathbf{P}$ defined in Eq. (3.2). This can be easily verified by using the Bayes rule to derive the posterior $p(m_j|x_i)$. On the other hand, as we saw before, the element q_{ij} of matrix $\mathbf{Q}$ approximates $p(m_j|x_i)$. Subsequently, we can conclude that matrix $\mathbf{Q}$ is in fact the approximation of the transition probability matrix $\mathbf{P}$ using only $|\mathcal{B}|$ number of parameters (or blocks); in particular, the block parameter q_{AB} approximates the probability of jumping from a datapoint in A to a datapoint in B. The main implication of this finding is that although the variational dual-tree framework was originally developed to approximate the log-likelihood of data, it can also be use to approximate the transition matrix of the random walk on similarity graphs. This is specifically important because it enables us to compute and store the transition probability matrix in $O(N^{1.5}\log N + |\mathscr{B}|)$ time and $O(|\mathscr{B}|)$ memory for large-scale problems.

In the light of the random walk view of $\mathbf{Q}$, the lower bound log-likelihood $\ell(\mathscr{D})$ in Eq. (3.10) will also have a new interpretation. More precisely, the second term in Eq. (3.10) can be reformulated as:

$$-\frac{1}{2\sigma^2}\sum_{(A,B)\in\mathscr{B}} q_{AB}\cdot D_{AB} = -\frac{1}{2\sigma^2}\sum_{i,j} q_{ij}\cdot \bar{d}_{ij} = -\frac{1}{2\sigma^2} tr(\mathbf{Q}\bar{\mathbf{D}}) \tag{3.20}$$

where, q_{ij} is defined as in Eq. (3.9), $\bar{d}_{ij} = D_{AB}/|A||B|, \forall x_i \in A, m_j \in B$ is the *block-average distance* and $\bar{\mathbf{D}} = [\bar{d}_{ij}]_{N\times N}$. This is, in fact, a common optimization term in similarity-graph learning from mutual distances [Jebara et al., 2009]. In particular, the maximization of Eq. (3.20) w.r.t. q_{ij} under the constraints in Eq. (3.7) will assign lower q_{ij} values to those transition pairs with large block-average distance $\bar{d}_{ij}$. However, there is one problem with the maximization of this term: it will make each point x_i only connected to its closest neighbor x_{j^*} with $q_{ij^*} = 1$ (and $q_{ij} = 0$ for $j \neq j^*$). In other words, the resulted graph will be highly disconnected. This is where the third term in Eq. (3.10) benefits the new interpretation. One can rewrite this term as:

$$-\sum_{(A,B)\in\mathscr{B}} |A||B|\cdot q_{AB}\log q_{AB} = -\sum_{i,j} q_{ij}\log q_{ij} = \sum_{i=1}^{N} H(q_{i\cdot}) \tag{3.21}$$

where $H(q_{i\cdot})$ is the entropy of the transition probability distribution from datapoint x_i. As opposed to Eq. (3.20), the term in Eq. (3.21) is maximized by a uniform distribution over the outgoing probabilities at each datapoint x_i; that is, a fully connected graph with equal transition probabilities. The third term in Eq. (3.10) therefore acts as the *regularizer* in the learning of the transition graph, trying to keep it connected. The trade-off between the second and the third terms is adjusted by the coefficient $1/2\sigma^2$: increasing the bandwidth σ will result in a more connected graph.

3.6.2 Optimizing The Bandwidth

Finding the transition probabilities for a random walk by the variational optimization of the lower-bound Eq. (3.10) has a side advantage too: Eq. (3.10) is a *quasi-concave* function of the bandwidth σ, which means that, given q_{AB}'s fixed, one can find the optimal bandwidth that maximizes the lower bound. By taking the derivative and solving for σ, the closed form solution is derived as:

$$\sigma^* = \sqrt{\frac{\sum_{(A,B)\in\mathscr{B}} q_{AB} \cdot D_{AB}}{Nd}} \tag{3.22}$$

Note that the reason we can learn the bandwidth this way is because we are maximizing the lower bound on the log-likelihood instead of the log-likelihood itself. In fact, the actual log-likelihood increases indefinitely as the bandwidth goes to zero (which indeed will result in overfitting).

In general, due to the dependence of σ^* to q_{AB}'s, we alternate the optimization of q_{AB}'s and σ in our framework. In practice, we have observed that the convergence of this alternate optimization is fast and not sensitive to the initial value of σ. However, we can also tune the bandwidth independent of q_{AB}'s by maximizing a looser lower bound which does not depend on q_{AB}'s. In particular, using the Jensen inequality, one can write:

$$\log p(\mathscr{D}) \geq \sum_i \sum_{j\neq i} p(m_j) \log p(x_i \mid m_j) \tag{3.23}$$

Now, by maximizing the right-hand side w.r.t. σ, we get:

$$\sigma^* = \frac{1}{N}\sqrt{\frac{\sum_i \sum_{j\neq i} \|x_i - x_j\|^2}{d}} \tag{3.24}$$

Of course, the bandwidth value obtained using the latter technique is sub-optimal compared to the one obtained using the former technique due to the fact that the bound in Eq. (3.23) is loose.

3.6.3 Fast Inference

In the previous subsections, we saw how the variational dual-tree based framework can be used to efficiently build and store a variational transition matrix $\mathbf{Q}$ of a random walk. We will now demonstrate that the block structure of this transition matrix can be very useful for further inference in similarity-graph based learning algorithms.

One common operation that is embedded in many algorithms involving transition matrices is the matrix-vector multiplication. For instance, in the *label propagation (LP)* algorithm used for graph-based semi-supervised learning [Zhou et al., 2003], the transition matrix is successively multiplied by intermediate label vectors to propagate the label information from the labeled datapoints to the unlabeled ones. More specifically, given a similarity graph over the data points x_i, the LP algorithm iteratively updates the label matrix $Y = [y_{ij}]_{N\times C}$ (where C is the number of label classes) at time $t+1$ by propagating the labels at time t one step forward according to the transition matrix $\mathbf{P}$:

$$Y^{(t+1)} \leftarrow \alpha \mathbf{P} Y^{(t)} + (1-\alpha) Y^0 \tag{3.25}$$

Here, the matrix Y^0 encodes the initial labeling of data such that $y^0_{ij} = 1$ for label$(x_i) = y_i = j$, and $y^0_{ij} = 0$ otherwise. The coefficient $\alpha \in (0,1)$ determines how fast the label values are updated.

As another example, many spectral graph-based methods (such as Diffusion Maps [Lafon and Lee, 2006, Nadler et al., 2006, Coifman et al., 2008, Amizadeh et al., 2012b] and Laplacian Eigenmaps [Belkin and Niyogi, 2002, Belkin and Niyogi, 2005, von Luxburg, 2007]) require to compute the eigen-decomposition of the transition matrix. The decomposition is usually carried out using some numerical algorithms such as Power Method or Arnoldi Iteration which have the matrix-vector multiplication as their basic operation. More specifically, in order to find the eigen decomposition of $\mathbf{P}$ using Arnoldi or Power methods, one needs to compute the following vector series:

$$b, \mathbf{P}b, \mathbf{P}^2 b, \mathbf{P}^3 b, \mathbf{P}^4 b, \ldots$$

where b is a random initial vector. Again, matrix multiplication is the key component of these algorithms.

The matrix-vector multiplication, in general, has $O(N^2)$ computational complexity and therefore is highly inefficient for large-scale problems, especially if many multiplications need to be performed successively (as expected in all the algorithms mentioned above.) Therefore, when dealing with large-scale problems, it is crucial for these algorithms to speed up this operation. In this section, we propose an algorithm that multiplies the block partitioned transition matrix $\mathbf{Q}$ by an arbitrary vector $Y = (y_1, y_2, \ldots, y_N)^T$ to achieve $Z = \mathbf{Q}Y \simeq \mathbf{P}Y$ in $O(|\mathscr{B}|)$ rather than $O(N^2)$ computations. To do so, our algorithm uses the MPT representation of $\mathbf{Q}$.

Algorithms 5 - 7 describes the $O(|\mathscr{B}|)$ computation of Z. The algorithm assumes that each y_i is stored at the corresponding leaf x_i in the MPT–e.g., by association prior to constructing the MPT. Alternatively, an index to the leaves can be constructed in $O(N)$ time. The algorithm starts with a COLLECTUP phase that traverses the MPT bottom-up and stores incrementally computed *sum-statistics* at each node A, as

$$T_A = \sum_{x_i \in A} y_i = T_{A.leftChild} + T_{A.rightChild}, \tag{3.26}$$

which is an $O(N)$ computation. With each q_{AB} stored at the node A marked with B in the MPT, a DISTRIBUTEDOWN phase now conceptually computes

$$z_i = \sum_{(A,B) \in \mathscr{B}(x_i)} |B| q_{AB} T_A \tag{3.27}$$

by following the path from each leaf to the root in the MPT. A more efficient implementation traverses the MPT top-down, avoiding recalculations of the shared terms for the updates by propagating the value of the shared terms through the variable py. This traversal has complexity $O(|\mathscr{B}|)$. On completion of the algorithm the leaves in the MPT will contain $Z = \{z_i\}_{i=1}^N$.

Algorithm 5: Calculate $Z = \mathbf{Q}Y$

```
1: function MULTIPLY($\mathbf{Q}$, $Y$)
2:   Input: the transition matrix $\mathbf{Q}$ and the operand vector $Y$
3:   Output: $Z = \mathbf{Q}Y$
4:   $A \leftarrow$ ROOT(MPT($\mathbf{Q}$)) // MPT($\mathbf{Q}$) returns the MPT representation of $\mathbf{Q}$
5:   COLLECTUP($A$, $Y$)
6:   DISTRIBUTEDOWN($A$, 0)
7:   $Z \leftarrow [z_i]_{N\times 1}, \forall i \in$ LEAVES(MPT($\mathbf{Q}$))
8:   return $Z$

9: end function
```

Algorithm 6: Collect-up

```
1: function COLLECTUP($A$, $Y$)
2:   Input: node $A$ in the tree and the operand vector $Y$
3:   Output: computes the sum-statistic at node $A$ and its descendents
4:   if ISLEAF($A$) then
5:     $T_A \leftarrow y_A$
6:   else
7:     COLLECTUP($A.leftChild$, $Y$)
8:     COLLECTUP($A.rightChild$, $Y$)
9:     $T_A \leftarrow T_{A.leftChild} + T_{A.rightChild}$
10:  end if

11: end function
```

3.6.4 Partitioning and Refinement

So far, we have assumed that a valid partition $\mathscr{B}$ of $\mathbf{Q}$ with $|\mathscr{B}|$ number of blocks is given. In this section, we illustrate how to construct an initial valid partition of $\mathbf{Q}$ and how to further refine it to increase accuracy of the model. We should note that the methods for partitioning and refinement in this section are different from the ones in [Thiesson and Kim, 2012].

Let $\mathscr{B}_{diag}$ denote the N neutral singleton blocks that appear on the diagonal of $\mathbf{Q}$. The coarsest (with the smallest number of blocks) valid partition $\mathscr{B}_c$ for $\mathbf{Q}$ is achieved when for every block $(A,B) \in \mathscr{B}_c = \mathscr{B} \setminus \mathscr{B}_{diag}$, we have that A and B are sibling subtrees in the partition tree. (Recall that the query and the kernel points are partitioned by the same tree.) Any other partition will either not be a valid partition conforming with the partition tree, or it will have a larger number of blocks. The number of blocks in $\mathscr{B}_c$ in this case, is therefore equal to twice the number of the sibling pairs (or the inner nodes) in the anchor tree; i.e. $|\mathscr{B}_c| = 2(N-1)$. Figure 5 is an example of a coarsest valid block partition. On the other hand, the most refined partition is achieved when $\mathscr{B}_r = \mathscr{B} \setminus \mathscr{B}_{diag}$ contains $N^2 - N$ singleton

Algorithm 7: Distribute-down

```
1:  function DistributeDown($A, py$)
2:    Input: node $A$ in the tree and the partial result $py$
3:    Output: computes $z_i$'s at the leaf descendents of $A$
4:    for all $B \in A_{mkd}$ do
5:      $py \leftarrow py + |B| q_{AB} T_A$
6:    end for
7:    if IsLeaf($A$) then
8:      $z_A \leftarrow py$
9:    else
10:     DistributeDown($A.leftChild, py$)
11:     DistributeDown($A.rightChild, py$)
12:   end if

13: end function
```

blocks. In fact, this partitioning is equivalent to having the exact transition matrix $\mathbf{P}$.

Hence, the number of blocks $|\mathscr{B}|$ in a valid partition can vary between $O(N)$ and $O(N^2)$. As we saw in the previous sections, $|\mathscr{B}|$ plays a crucial role in the computational performance of the whole framework and we therefore want to keep it as small as possible. On the other hand, keeping $|\mathscr{B}|$ too small may excessively compromise the approximation accuracy. Therefore, the rational approach would be to start with the coarsest partition $\mathscr{B}_c$ and split the blocks in $\mathscr{B}_c$ into smaller ones only if needed. This process is called *refinement*. As we refine more blocks, the accuracy of the model effectively increases while its computational performance degrades. Note that a block (A,B) can be refined in two ways: either *vertically* into $\{(A.leftChild,B),(A.rightChild,B)\}$ or *horizontally* into $\{(A,B.leftChild),(A,B.rightChild)\}$. Figure 6 shows these two types of refinement applied on the shaded block shown on the left column of the figure.

After any refinement, we can re-optimize (3.10) to find the new variational approximation $\mathbf{Q}$ for $\mathbf{P}$. Importantly, any refinement loosens the constraints imposed on $\mathbf{Q}$, implying that the KL-divergence from $\mathbf{P}$ cannot increase. From (3.8) we can therefore easily see that a refinement is likely to increase the log-likelihood lower bound $\ell(\mathscr{D})$, and can never decrease it. Intuitively, we want to refine those blocks which increase $\ell(\mathscr{D})$ the most. To find such blocks, one need to refine each block in each direction (i.e. horizonal and vertical) one at a time, re-optimize $\mathbf{Q}$, find the difference between the new $\ell(\mathscr{D})$ and the old one (aka the *log-likelihood gain*), and finally pick the refinements with maximum difference. However, this is an expensive process since we need to perform re-optimization per each possible refinement. Now the question is, whether we can obtain an estimate of the log-likelihood gain for each possible refinement without performing re-optimization. The answer is positive for horizontal refinements, and rests on the fact that each row in $\mathbf{Q}$ defines a (posterior) probability distribution and therefore must sum to one, i.e. Eq. (3.11). Now, consider the horizontal refinement of (A,B) into $\{(A,B.leftChild),(A,B.rightChild)\}$. By this refinement, in essence, we allow a random walk, where the probability of jumping from points in A to the ones in $B.leftChild$ (i.e. $q_{A,B.leftChild}$) is different from that of jumping from A to $B.rightChild$ (i.e. $q_{A,B.rightChild}$). If we keep the block parameters for other blocks fixed, we can still *locally* change $q_{A,B.leftChild}$ and $q_{A,B.rightChild}$ to increase $\ell(\mathscr{D})$. Since the sum of the outgoing probabilities from each point is 1 (see Eq. (3.11)) and the other q's are unchanged, the sum of the outgoing probabilities from A to $B.leftChild$ and $B.rightChild$ must be equal to that of the old one from A to B; that is

$$|B.leftChild|q_{A,B.leftChild} + |B.rightChild|q_{A,B.rightChild} = |B|q_{AB} \tag{3.28}$$

Under this local constraint, we can find $q_{A,B.leftChild}$ and $q_{A,B.rightChild}$ in closed form such that $\ell(\mathscr{D})$ is maximized:

$$C \in \{B.leftChild, B.rightChild\},\ q_{AC} = \frac{|B|\exp(G_{AC})q_{AB}}{\sum_{D\in\{B.leftChild,B.rightChild\}}|D|\exp(G_{AD})} \tag{3.29}$$

where $G_{AB} = -D_{AB}/(2\sigma^2|A||B|)$. By inserting Eq. (3.29) in Eq. (3.10), we can compute the maximum log-likelihood gain for the horizontal refinement of (A,B) as

$$\Delta^h_{AB} = \ell'(\mathscr{D}) - \ell(\mathscr{D}) = |A||B|q_{AB}\cdot\log\left(\frac{\sum_{C\in\{B.leftChild,B.rightChild\}}|C|\exp(G_{AC})}{|B|\exp(G_{AB})}\right) \tag{3.30}$$

where $\ell'(\mathscr{D})$ denotes the log-likelihood lower bound after the refinement. Note that the actual gain can be greater than Δ^h_{AB} because after a refinement, all q-values are re-optimized. In other words, Δ^h_{AB} is a lower bound for the actual gain and is only used to pick blocks for refinements. Unfortunately, for vertical refinements such a bound is not easily obtainable. The reason is that the constraints in Eq. (3.11) will not allow $q_{A.leftChildB}$ and $q_{A.rightChildB}$ to change if we fix the remaining block parameters. Therefore, the local optimization that we applied for the horizontal refinement cannot be applied to estimate the gain of a vertical refinement. On the other hand, we cannot simply avoid vertical refinements just because we cannot compute their approximate gain. To address this issue, whenever we pick a block (A,B) for horizontal refinement, we also apply vertical refinement to its transpose counterpart (B,A) if it also belongs to $\mathscr{B}$. We call this the *symmetric refinement* because it leaves us with a symmetric block-partitioning of the transition matrix.

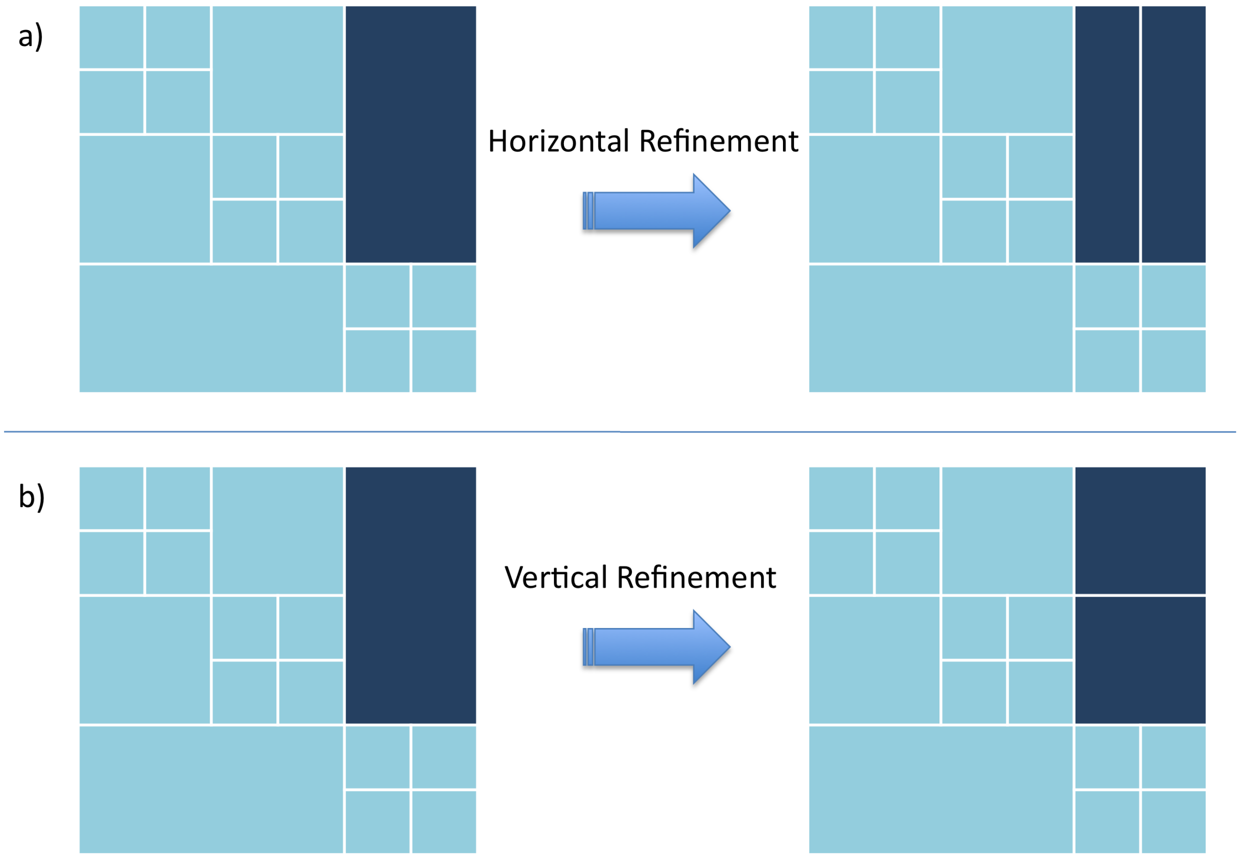

Figure 6: (A) Horizonal refinement for the shaded block (B) Vertical refinement for the shaded block.

By computing Δ^h_{AB} for all blocks, at each step, we greedily pick the block with the maximum gain and apply the symmetric refinement. The newly created blocks are then added to the pool of blocks and the process is repeated until the number of blocks reaches the maximum allowable block number $|\mathscr{B}|_{max}$, which is an input argument to the algorithm. This greedy algorithm can be efficiently implemented using a priority queue prioritized by the log-likelihood gain in Eq. (3.30). Algorithm 8 describes the symmetric refinement process in

Algorithm 8: Symmetric Refinement

```
1: function SymmetricRefinement($Qu, |\mathscr{B}|_{max}$)
2:   Input: The priority queue $Qu$ containing the initial blocks and the desired maximum number of blocks $|\mathscr{B}|_{max}$
3:   Output: Split the existing blocks to increase the log-likelihood lower bound
4:   while $|\mathscr{B}| < |\mathscr{B}|_{max}$ do
5:     $(A, B) \leftarrow Qu.dequeue()$
6:     $Qu.enqueue((A, B.leftChild), \Delta^h_{A,B.leftChild})$
7:     $Qu.enqueue((A, B.rightChild), \Delta^h_{A,B.rightChild})$
8:     // Applying vertical refinement to the transpose block
9:     if $(B, A) \in Qu$ then
10:      $Qu.remove((B, A))$
11:      $Qu.enqueue((B.leftChild, A), \Delta^h_{B.leftChild,A})$
12:      $Qu.enqueue((B.rightChild, A), \Delta^h_{B.rightChild,A})$
13:    end if
14:  end while
15:  Re-estimate the $Q$ parameters
16: end function
```

more details. Note that after the refinement process, the lower bound on the log-likelihood need to be re-optimized in order to estimate the block parameters for the new partitioning of the **Q** matrix. As mentioned before, this operation can be done in $O(|\mathscr{B}|)$.

3.7 BREGMAN VARIATIONAL DUAL-TREE FRAMEWORK

The variational dual-tree (VDT) method described in the previous section proposes an elegant framework for approximating large-scale Markov transition matrices using the reduced hierarchical block structure. One inherent assumption in this framework is the underlying distance metric in the input space should be the Euclidean distance; this is somewhat restrictive. In many real-world problems, the Euclidean distance is simply not the best way to quantify the similarity between datapoints. As an example, for frequency data, where each feature represents the observed counts (or frequency) of a specific event, KL-divergence can outperform the Euclidean distance in some applications [Banerjee et al., 2005].

On the other hand, the formulation of the VDT framework does not seem to depend on the choice of distance metric, which makes it very tempting to replace the Euclidean distance with a general distance metric. However, there is one problem: the de-coupling in Eq. (3.14) was achieved only because of the Euclidean distance's special form which is not the case for a general distance metric. Unfortunately, we cannot compromise on this de-coupling simply because, without it, the overall complexity of the framework is back to $O(N^2)$. One solution is to use some approximation technique similar to Fast Gauss Transform techniques [Yang et al., 2005] to approximately de-couple a general distance metric. Although, this may work well for some special cases, in general, the computational burden of such approximation can be prohibitive; besides, we will have a new source of approximation error.

So, if we cannot extend VDT for general distance metric, is there any sub-class of metrics or divergences which we can safely use to extend the VDT framework? The answer is yes, the family of *Bregman divergences* is a qualified candidate. This family contains a diverse set of divergences which also include the Euclidean distance. By definition, the Bregman divergence has a de-coupled form which makes it perfect for our purpose. From the applied side, Bregman divergences cover some very practical divergences and metrics such as Euclidean distance, KL-Divergence and Logistic Loss that are widely used in many engineering and scientific applications. Furthermore, the natural correspondence of Bregman divergences with the exponential families provides a neat probabilistic interpretation for our framework. In this section, we describe our proposed extension of the VDT framework to the Bregman divergences; we call this framework the *Bregman Variational Dual-Tree (BVDT) framework*.

3.7.1 The Bregman Divergences and The Exponential Families

Before illustrating the BVDT framework, we briefly review the Bregman divergence, its important properties and its connection to the exponential families. Interested readers may refer to [Banerjee et al., 2005] for further details.

Let $\mathscr{X} \subseteq \mathbb{R}^d$ be a convex set in $\mathbb{R}^d$, $ri(\mathscr{X})$ denote the relative interior of $\mathscr{X}$, and $\phi : \mathscr{X} \mapsto \mathbb{R}$ be a strictly convex function differentiable on $ri(\mathscr{X})$, then the Bregman divergence $d_\phi : \mathscr{X} \times ri(\mathscr{X}) \mapsto [0,\infty)$ is defined as:

$$d_\phi(x,y) \triangleq \phi(x) - \phi(y) - (x-y)^T \nabla\phi(y) \tag{3.31}$$

where, $\nabla\phi(y)$ is the gradient of $\phi(\cdot)$ evaluated at y. For different choices of $\phi(\cdot)$, we will get different Bregman divergences; $\phi(\cdot)$ is referred to as the *seed function*. Table 6 lists some famous Bregman divergences along with their corresponding seed functions. It is important to note that the general Bregman divergence is *not* a distance metric: it is not symmetric, nor does it satisfy the triangular inequality. However, we have $\forall x \in \mathscr{X}, y \in ri(\mathscr{X}) : d_\phi(x,y) \geq 0, d_\phi(y,y) = 0$.

Let $\mathscr{S} = \{x_1, \ldots, x_n\} \subset \mathscr{X}$ and X be a random variable that takes values from $\mathscr{S}$ with uniform distribution[2]; the *Bregman information* of the random variable X for the Bregman divergence $d_\phi(\cdot,\cdot)$ is defined as:

$$I_\phi(X) \triangleq \min_{s \in ri(\mathscr{X})} \mathbb{E}\left[d_\phi(X,s)\right] = \min_{s \in ri(\mathscr{X})} \frac{1}{n}\sum_{i=1}^{n} d_\phi(x_i,s) \tag{3.32}$$

The optimal s that minimizes Eq. (3.32) is called the *Bregman representative* of X and is equal to:

$$s^* = \arg\min_{s \in ri(\mathscr{X})} \mathbb{E}\left[d_\phi(X,s)\right] = E[X] = \frac{1}{n}\sum_{i=1}^{n} x_i \triangleq \mu \tag{3.33}$$

That is, the Bregman representative of X is always equal to the sample mean of $\mathscr{S}$ *independent of the Bregman divergence* $d_\phi(\cdot,\cdot)$ *used to compute the divergence.*

The probability density function $p(z)$, defined on set $\mathscr{Z}$, belongs to an *exponential family* if there exists a mapping $g : \mathscr{Z} \mapsto \mathscr{X} \subseteq \mathbb{R}^d$ that can be used to re-parameterize $p(z)$ as:

$$p(z) = p(x;\theta) = \exp(\theta^T x - \psi(\theta))p_0(x) \tag{3.34}$$

where $x = g(z)$ is the *natural statistics* vector, θ is the *natural parameter* vector, $\psi(\theta)$ is the *log-partition function* and $p_0 : \mathscr{X} \mapsto \mathbb{R}_+$ is the base measure. Eq. (3.34) is called the *canonical form* of p. If θ takes values from parameter space Θ, Eq. (3.34) defines the family $\mathscr{F}_\psi = \{p(x;\theta) \mid \theta \in \Theta\}$ parameterized by θ. If Θ is an open set and we have that $\nexists c \in \mathbb{R}^d$ s.t. $c^T g(z) = 0, \forall z \in \mathscr{Z}$, then family $\mathscr{F}_\psi$ is called a *regular exponential family*. [Banerjee et al., 2005] has shown that any probability density function $p(x;\theta)$ of a regular exponential family with the canonical form of Eq. (3.34) can be uniquely expressed as:

$$p(x;\theta) = \exp(-d_\phi(x,\mu))\exp(\phi(x))p_0(x) \tag{3.35}$$

[2]The results hold for any distribution on $\mathscr{S}$.

where $\phi(\cdot)$ is the *conjugate function* of the log-partition function $\psi(\cdot)$, $d_\phi(\cdot,\cdot)$ is the Bregman divergence defined w.r.t. the seed function $\phi(\cdot)$, and μ is the *mean parameter*. The mean parameter vector μ and the natural parameter vector θ are connected through:

$$\mu = \nabla\psi(\theta),\ \theta = \nabla\phi(\mu) \tag{3.36}$$

Moreover, [Banerjee et al., 2005] (Theorem 6) has shown there is a bijection between the regular exponential families and the regular Bregman divergences.[3] The last column in Table 6 shows the corresponding exponential family of each Bregman divergence. Using Eq. (3.33), one can also show that, given the finite sample $\mathscr{S} = \{x_1, \ldots, x_n\}$, the maximum-likelihood estimate of the mean parameter $\hat{\mu}$ for any regular exponential family $\mathscr{F}_\psi$ is always equal to the sample mean of $\mathscr{S}$ regardless of $\mathscr{F}_\psi$.

[3]Interested readers may refer to [Banerjee et al., 2005] (Definition 8) for the technical definition of regular Bregman divergences. The well-known Bregman divergence we have considered in this chapter all belong to this class.

Name	$\mathscr{X}$	$\boldsymbol{\phi}(\mathbf{x})$	$\mathbf{d}_{\boldsymbol{\phi}}(\mathbf{x},\mathbf{y})$	**Exponential Family**
Logistic Loss	$(0,1)$	$x\log x$	$x\log\left(\frac{x}{y}\right)+(1-x)\log\left(\frac{1-x}{1-y}\right)$	1D Bernoulli
Itakura-Saito Dist.	$\mathbb{R}_{++}$	$-\log x-1$	$\frac{x}{y}-\log\left(\frac{x}{y}\right)-1$	1D Exponential
Relative Entropy	$\mathbb{Z}_{+}$	$x\log x-x$	$x\log\left(\frac{x}{y}\right)-x+y$	1D Poisson
Euclidean Dist.	$\mathbb{R}^d$	$\\|x\\|^2/2\sigma^2$	$\\|x-y\\|^2/2\sigma^2$	Spherical Gaussian
Mahalonobis Dist.	$\mathbb{R}^d$	$x^T\Sigma^{-1}x$	$(x-y)^T\Sigma^{-1}(x-y)$	Multivariate Gaussian
KL-Divergence	d-simplex	$\sum_{j=1}^d x(j)\log x(j)$	$\sum_{j=1}^d x(j)\log\left(\frac{x(j)}{y(j)}\right)$	-
-	int. d-simplex	$\sum_{j=1}^d x(j)\log\left(\frac{x(j)}{L}\right)$	$\sum_{j=1}^d x(j)\log\left(\frac{x(j)}{y(j)}\right)$	Multinomial

Table 6: Famous Bregman divergences along with their corresponding $\phi(\cdot)$ function, its domain and the corresponding exponential family distribution.

3.7.2 Bregman Variational Approximation

Having described the basic concepts of Bregman divergences, we are now ready to nail down the BVDT framework. Let $\mathscr{S} = \{z_1, z_2, \ldots, z_N\} \subset \mathcal{Z}$ be a finite sample from the convex set $\mathcal{Z}$ which is not necessarily an Euclidean space. We are interested to approximate the transition matrix $\mathbf{P}$ on the similarity graph of $\mathscr{S}$ where we know the Euclidean distance is not necessarily the best way to encode similarity. In order to do so, we assume $\mathscr{S}$ is sampled according to an unknown mixture density model $p^*(z)$ with K components from the regular exponential family $\mathscr{F}_\psi$. That is, there exists the mapping $g : \mathcal{Z} \mapsto \mathscr{X} \subseteq \mathbb{R}^d$ such that $p^*(z)$ can be re-parametrized in the canonical form as [4]:

$$p^*(x) = \sum_{i=1}^{K} p(\theta_i) \exp(\theta_i^T x - \psi(\theta_i)) p_0(x) = \sum_{i=1}^{K} p(\mu_i) exp(-d_\phi(x, \mu_i)) \exp(\phi(x)) p_0(x) \qquad (3.37)$$

where, $\mu_i = \nabla\psi(\theta_i)$ and $\phi(\cdot) = \psi^*(\cdot)$ is the conjugate of $\psi(\cdot)$.

In order to compute the transition matrix $\mathbf{P}$ over $\mathscr{S}$ using our framework, we need to compute the log-likelihood of $\mathscr{S}$ where the density of each point is modeled via p^*. However, $p^*(x)$ in not known *a priori*. In particular, for a given problem, we assume we only know the underlying regular exponential family $\mathscr{F}_\psi$ in advance; that is, we only know the function $\psi(\cdot)$ (or equivalently $\phi(\cdot)$). But we do *not* know the actual number of components K or the value of the natural parameter vector θ_i (or equivalently the mean parameter vector μ_i) for each component. Therefore, to model this distribution, we turn our attention to the most well-known non-parametric density estimation technique: kernel density estimation. To this end, let $\mathscr{D} = g(\mathscr{S}) = \{x_1, x_2, \ldots, x_N\} \subset \mathscr{X}$ be the natural statistics of sample $\mathscr{S}$ s.t. $x_i = g(z_i)$. We put a kernel belonging to the regular exponential family $\mathscr{F}_\psi$ on each member of $\mathscr{D}$, where $\mathscr{F}_\psi$ is the same family that models the components of the true mixture model p^* which generated the data in the first place. In particular, for each $x_j \in \mathscr{D}$, we assign the kernel $p(x \mid m_j) = \exp(-d_\phi(x, m_j)) \exp(\phi(x)) p_0(x)$ centered on x_j; that is, $m_j = x_j = g(z_i)$. Now, the likelihood of $\mathscr{D}$ using the kernel density estimation is derived as:

$$p(\mathscr{D}) = \prod_{i=1}^{N} \sum_{j \neq i} p(m_j) p(x_i \mid m_j) = \prod_{i=1}^{N} \sum_{j \neq i} p(m_j) \exp(-d_\phi(x_i, m_j)) \exp(\phi(x_i)) p_0(x_i)$$

[4]For some exponential families such as the Gaussian (with known variance) and the Multinomial, the mapping $g(\cdot)$ is identity.

Given a block partitioning $\mathscr{B}$ on $\mathbf{P}$, we follow the similar steps in Eq. (3.6)-(3.13) to derive the block-partitioned variational lower-bound on $p(\mathscr{D})$:

$$\ell(\mathscr{D}) = c - \sum_{(A,B)\in\mathscr{B}} q_{AB} \cdot D_{AB} - \sum_{(A,B)\in\mathscr{B}} |A||B| \cdot q_{AB} \log q_{AB},$$

where

$$\begin{aligned} c &= -N \log(N-1) + \sum_{i=1}^{N} \big(\phi(x_i) + \log p_0(x_i)\big) \\ D_{AB} &= \sum_{x_i \in A} \sum_{m_j \in B} d_\phi(x_i, m_j). \end{aligned} \tag{3.38}$$

Now we can maximize $\ell(\mathscr{D})$ subject to the constraints in Eq. (3.11) to find the approximation $\mathbf{Q}$ of $\mathbf{P}$ using the same $O(|\mathscr{B}|)$-time algorithm in the VDT framework.

The crucial aspect of the BVDT framework is D_{AB} in Eq. (3.38) is de-coupled into statistics of the subtrees A and B using the definition of the Bregman divergence:

$$D_{AB} = |B| S_1(A) + |A| \big(S_2(B) - S_1(B)\big) - S_3(A)^T S_4(B) \tag{3.39}$$

where for any subtree $T \in \mathscr{T}$, we define the statistics of T as,

$$\begin{aligned} S_1(T) &= \sum_{x\in T} \phi(x), & S_2(T) &= \sum_{x\in T} x^T \nabla\phi(x) \\ S_3(T) &= \sum_{x\in T} x, & S_4(T) &= \sum_{x\in T} \nabla\phi(x) \end{aligned} \tag{3.40}$$

Similar to the tree statistics in the original VDT framework, these statistics can be incrementally computed and stored while the cluster tree is being built in Algorithm 4 (in overall $O(N)$ time) such that at the optimization time, D_{AB} is computed in $O(1)$.

Finally, by setting the seed function to $\phi(x) = \|x\|^2/2\sigma^2$ and doing the algebra, the BVDT framework reduces to the Euclidean VDT framework of Section 3.6; that is, the Euclidean VDT framework is a special case of the BVDT framework.

3.7.3 Bregman Anchor Trees

Recall that the approximation in the Euclidean VDT framework is based on the cluster hierarchy $\mathscr{T}$ of the data which is built using the anchor tree method of Section 3.5.3 with the Euclidean distance. For the BVDT framework, we can no longer use this algorithm because the Euclidean distance no longer reflects the similarity in the input space. For this reason, we need to develop an anchor tree construction algorithm for general Bregman divergences. This generalization is not straightforward though, merely because a general Bregman divergence is neither symmetric nor does it hold the triangle inequality. In particular, we need to address two major challenges.

First, in the anchor agglomeration phase of the anchor tree construction algorithm (Algorithm 4), at each step, the chosen anchor nodes A and B for merging are the ones that generates a parent node C with the *minimum radius* in the Euclidean distance sense (i.e. Line 5 of Algorithm 4). However, due to the asymmetry of a general Bregman divergence, this merging criterion is no longer meaningful with Bregman input spaces. To address this issue, we propose to use the criterion suggested in the recent work by [Telgarsky and Dasgupta, 2012]. In particular, at each agglomeration step, anchors A and B with the minimum *merging cost* are picked to merge into the parent anchor C. The cost for merging a pair of anchors A and B is defined as:

$$\Delta(A,B) = |A| \cdot d_\phi(A_p, C_p) + |B| \cdot d_\phi(B_p, C_p) \tag{3.41}$$

where C_p is the parent anchor's pivot which is given by Eq. (3.18). [Telgarsky and Dasgupta, 2012] has shown that the merging cost in Eq. (3.41) can be rewritten as the difference of cluster unnormalized Bregman informations before and after merging. In other words, at each step, the algorithm merges the two clusters that decrease the Bregman information the least.

Second, as shown before, using the halfway Euclidean distance in Eq. (3.16) as the stealing threshold in the anchor construction phase significantly cuts the unnecessary computations (i.e. Line 7 of Algorithm 3). This threshold, however, is meaningless for a general

Bregman divergence simply because a general Bregman divergence is not a metric. Therefore, we need to develop an equivalent threshold for Bregman divergences to achieve a similar computational gain in constructing Bregman anchor trees. The following proposition addresses this problem:

Preposition 1. *Let A^{curr} and A^{new} denote the current and the newly created anchors, respectively, where A^{new} is stealing datapoints from A^{curr}. Define*

$$d_{thr} = \frac{1}{2}\min_{y\in\mathscr{X}}\left[d_{\phi}(y, A_p^{curr}) + d_{\phi}(y, A_p^{new})\right] \tag{3.42}$$

Then for all $x \in A^{curr}$ such that $d_{\phi}(x, A_p^{curr}) \le d_{thr}$, we will have $d_{\phi}(x, A_p^{curr}) \le d_{\phi}(x, A_p^{new})$; that is, x cannot be stolen from A_p^{curr} by A_p^{new}. Furthermore, the minimizer of Eq. (3.42) is equal to:

$$y^* = \nabla\phi^{-1}\left[\frac{1}{2}\left(\nabla\phi(A_p^{curr}) + \nabla\phi(A_p^{new})\right)\right] \tag{3.43}$$

Proof. Proof by contradiction: assume there exists $x \in A^{curr}$, such that:

$$d_{\phi}(x, A_p^{curr}) \le d_{thr}$$
$$d_{\phi}(x, A_p^{curr}) > d_{\phi}(x, A_p^{new})$$

Then immediately, we will have $d_{\phi}(x, A_p^{new}) < d_{thr}$ and therefore,

$$\frac{1}{2}\left[d_{\phi}(y, A_p^{curr}) + d_{\phi}(y, A_p^{new})\right] < \frac{1}{2}(d_{thr} + d_{thr}) = d_{thr}$$

That is, there exists a point x for which the function in Eq. (3.42) will take a value smaller than d_{thr} and this contradicts the assumption that the minimum value of Eq. (3.42) is d_{thr}.

The second part of the proof is simply obtained by differentiating Eq. (3.42) w.r.t. y and solving for y. □

Now by replacing Line 4 in Algorithm 3 by Eq. (3.42), we effectively provide a similar speed-up mechanism for the general Bregman anchor tree construction algorithm. Note that for the special case of Euclidean distance where $\phi(x) = \|x\|^2/2\sigma^2$, Eq. (3.42) reduces to Eq. (3.16). [5]

[5]Note that this reduction is up to power of 2, but since x^2 is a strictly increasing function for $x \ge 0$, we can safely replace all the Euclidean distances in the original anchor tree construction algorithm with the squared Euclidean distance.

3.8 EXPERIMENTS

In this section, we present the experimental evaluation of our framework. In particular, we have evaluated how well our method performs for the semi-supervised learning (SSL) task using Label Propagation (LP) illustrated in Eq. (3.25). The LP algorithm starts with a partial vector of labels and iteratively *propagates* those labels over the underlying similarity graph to unlabeled nodes (see Eq. (3.25)). After T iterations of propagation, the inferred labels at unlabeled nodes are compared against the true labels (which were held out) to compute Area Under Curve (AUC) for 2-class problems and Accuracy (ACC) for multi-class problems.[6]

We also measure the time (in ms) taken to build each model, propagate labels and refine each model. In our experiments, we set $T = 500$ and $\alpha = 0.01$. It should be noted that, here the goal is not to achieve a state-of-the-art SSL algorithm, but to relatively compare our framework to baselines in terms of efficiency and accuracy under the *same* conditions. This means that we have not tuned the SSL parameters to improve SSL; however, we use the same parameters for all competitor methods.

3.8.1 Methods

We have compared our Variational Dual-Tree Framework, *VDT*, with two other methods for building and representing $\mathbf{P}$. The first method is the straightforward computation of $\mathbf{P}$ using Eq. (3.2). We refer to this representation as the *Exact model*. In terms of computational complexity, it takes $O(N^2)$ to build, store, and multiply an arbitrary vector by $\mathbf{P}$ using the Exact method.

The second method is the *k-nearest-neighbor (kNN)* algorithm where each data point is connected only to its k closest neighbors. In other words, the rows of matrix the similarity matrix $\mathbf{W}$ (and subsequently $\mathbf{P}$) will contain only k non-zero entries such that $\mathbf{P}$ can be represented as a sparse matrix for small k's. In other words, kNN zeros out many of the transition probabilities to make $\mathbf{P}$ sparse, as opposed to our VDT method that groups and

[6]For multi-class problems, we concurrently run LP for all classes using the one-vs-all scheme, and then output the class with the maximum label value for each unlabeled datapoint.

shares parameters in $\mathbf{P}$. Note that we still assign weights to the k edges for each data point using Eq. (3.2). This means that as we increase k toward N, the model converges to the Exact model. Therefore, k acts as a tuning parameter to trade off between computational efficiency and accuracy (The similar role $|\mathscr{B}|$ plays in VDT). In this section, we refer to k and $|\mathscr{B}|$ as the *trade-off* parameters.

Using the kNN representation, $\mathbf{P}$ can be stored and multiplied by an arbitrary vector in $O(kN)$. For constructing a kNN graph directly, the computational complexity is $O(rN^2)$ where $r = \min\{k, \log N\}$. That is, building the kNN representation can be even more expensive than building the Exact representation depending on k. For this reason, the exact kNN graph is not a practical choice for large-scale problems. A faster approach to build a kNN graph is to use the cluster tree of the data in order to avoid unnecessary distance computations. [Moore, 1991] proposed a speed-up of the kNN graph construction that utilizes a *kd-tree*. In our implementation of kNN, we have used the same algorithm with the kd-tree replaced by the anchor tree, illustrated in Section 3.5.3. We refer to this algorithm as *fast kNN*. The computational analysis of fast kNN greatly depends on the distribution of data points in the space. In the best case, it takes $O\big(N(N^{0.5}\log N + k\log k)\big)$ to build the kNN graph using fast kNN. However, in the worst case, the computational order is $O\big(N(N^{0.5}\log N + N\log k)\big)$. Table 7 summarizes the computational complexity orders for the models compared in the experiments. Note that the number of parameters for the kNN model is kN; that is, if our VDT framework has $|\mathscr{B}| = kN$ blocks, then the VDT and the kNN will have the same memory and multiplication complexity.

Models	**Construction**	**Memory**	**Multiplication**
Exact	$O(N^2)$	$O(N^2)$	$O(N^2)$
kNN	$O(rN^2)$	$O(kN)$	$O(kN)$
Fast kNN	$O\big(N(N^{0.5}\log N + h\log k)\big)^*$	$O(kN)$	$O(kN)$
VDT	$O(N^{1.5}\log N + \lvert\mathscr{B}\rvert)$	$O(\lvert\mathscr{B}\rvert)$	$O(\lvert\mathscr{B}\rvert)$

Table 7: Theoretical complexity analysis results. (*) h is equal to k in the best case and N in the worst case.

3.8.2 Efficiency and Quality vs. Problem Size

In the first experiment, we compare the computational complexity and the approximation accuracy of the competing methods against the problem size. In particular, the goal in this experiment, as we increase the problem size N, is to study (a) the time needed to build the Exact model, the coarsest VDT model (i.e. $|\mathscr{B}| = 2(N-1)$), and the coarsest kNN model (i.e. $k = 2$), (b) the time needed for vector-matrix multiplication (or equivalently the *propagation time*) in these models, and (c) the AUC of the LP algorithm when each of the outlined models is used to build the transition matrix, given 10% initial labeled data.

3.8.2.1 Data We have performed the first experiment on two UCI datasets [Bache and Lichman, 2013]:

MAGIC Gamma Telescope Dataset: This dataset consists of Monte Carlo simulated gamma shower images recorded by a Cherenkov gamma telescope [Bock et al., 2004]. Each image is typically an elongated cluster of points which is described by 11 image features such as point density, cluster orientation, etc. The task is to distinguish between the true source of each image recording which can be either a high-energy gamma particle (the signal) or the hadronic showers by cosmic rays (the background). There are total number of 19,020 images in this dataset (i.e. $N = 19,020$).

MiniBooNE Dataset: This dataset is related to the task of *Particle Identification (PID)* in High Energy Physics [Roe et al., 2005]. The dataset consists of recorded *events* which can be either *signal* or *background*. Each event is described by 50 distinguishing variables and the dataset contains 130,065 events (i.e. $N = 130,065$).

3.8.2.2 Experimental Setup To study and compare the computational complexity and the accuracy of the competing models, we have built and worked with each model with different sample sizes. In particular, for each sample size s, we draw samples of size s from each dataset and use each sample to construct a model. Once a model is built, we choose 10% of the sample randomly to be fed to the LP algorithm as the labeled partition. After the propagation is complete, we measure the AUC of the inferred labels over the unlabeled datapoints. As for the LP algorithm, the α parameter in Eq. (3.25) is set to 0.01 and the number of iterations T is set to 500. We have repeated this experiment 5 times for each problem size and each dataset and have reported the average results.

3.8.2.3 Results Figure 7(A) shows the construction time (in ms) for the three models over the MAGIC Gamma Telescope Dataset, as the problem size increases from 100 to 15,000. Also, Figure 7(B) shows the propagation time for the LP algorithm run on each of the models. Note that both of the axes in the plot are in the log-scale. Finally, Figure 7(C) depicts the average AUC computed over the unlabeled datapoints based on the true withheld labels as well as the inferred labels by the LP algorithm. Figure 8 illustrates the same plots for the MiniBooNE Dataset as the problem size increases from 1000 to 120,000. Note that for both datasets, we did not run the Exact method for problem sizes greater than 10,000 due to the demanding CPU and memory requirements for this method. Tables 8-13 report the results in the tabular format along with their 95% confidence intervals.

As the results show for both datasets, the VDT and kNN methods are both orders of magnitude faster than the Exact method in terms the construction time. Also, according to these results, it is slightly faster to build the VDT model than the kNN representation. In terms of the propagation time, VDT and kNN models are also orders of magnitude faster than the Exact method; for larger problem sizes, the kNN model is slightly faster than the VDT framework. Note that, for all three models, the propagation time is directly proportional to the number of parameters used to represent the transition matrix; therefore, the propagation time can be also regarded as a proxy for the memory usage of each model. Of course, the computational efficiency of our proposed model as well as the kNN representation comes at a price: the average AUC of both methods are lower than that of the Exact model and the difference is statistically significant in most cases. However, a couple of percentage loss in terms of AUC in exchange for orders of magnitude computational gain is a practical choice that is simply inevitable for large-scale problems. Furthermore, we observed that in most cases our proposed VDT framework statistically significantly outperforms the kNN model in terms of AUC. One possible explanation for this behavior relates to the *over-sparseness* of the kNN model at its coarsest level of approximation (i.e. $k = 2$). As mentioned before, for this experiment, we have constructed and compared the coarsest VDT and kNN models where both models have exactly $2N$ number of parameters. For the kNN model, this corresponds to the 2-NN graph. The problem with the 2-NN graph is it can be fairly disconnected depending on the geometry of data in the input space which, in turn, can degrades the performance of the LP algorithm. Our proposed model, on the other hand, always produces a connected graph even at its coarsest level of approximation. That is, using the VDT framework, the jumping probability between two far clusters of datapoints might be fairly small but is still greater than 0 which means that the propagation of labels from one cluster to another is still possible.

Problem Size	**VDT**	**Fast KNN**	**Exact**
100	4 ± 4.801	6 ± 4.801	18 ± 3.92
500	26 ± 4.801	38 ± 3.92	470 ± 1
1,000	62 ± 3.92	80 ± 1	$1,908 \pm 11.4287$
2,000	124 ± 4.801	178 ± 3.92	$7,654 \pm 15.9231$
5,000	338 ± 11.4287	478 ± 7.33365	$48,286 \pm 558.191$
10,000	710 ± 22.34741	$1,018 \pm 7.33365$	$192,470 \pm 262.01$
15,000	$1,100 \pm 17.5308$	$1,602 \pm 13.0012$	-

Table 8: The construction time for the MAGIC Gamma Telescope dataset in ms for VDT, Fast KNN and the Exact method.

Problem Size	**VDT**	**Fast KNN**	**Exact**
100	14 ± 4.801	16 ± 4.801	288 ± 3.92
500	94 ± 4.801	80 ± 1.0	$7,028 \pm 11.4287$
1,000	210 ± 6.19806	174 ± 4.801	$28,028 \pm 13.0012$
2,000	450 ± 10.7354	356 ± 4.801	$111,930 \pm 17.5308$
5,000	$1,382 \pm 39.8801$	954 ± 39.4929	$694,728 \pm 6860.79$
10,000	$4,274 \pm 182.037$	$2,420 \pm 198.628$	$2.72216e+06 \pm 120.7271$
15,000	$8,166 \pm 222.233$	$4,302 \pm 144.483$	-

Table 9: The propagation time for the MAGIC Gamma Telescope dataset in ms for VDT, Fast KNN and the Exact method.

Problem Size	VDT	Fast KNN	Exact
100	0.742664 ± 0.079751	0.660928 ± 0.0823052	0.780586 ± 0.075803
500	0.739969 ± 0.0264248	0.676194 ± 0.037017	0.797762 ± 0.0064706
1,000	0.735214 ± 0.0257284	0.681188 ± 0.0354759	0.816883 ± 0.025055
2,000	0.77054 ± 0.0220913	0.686289 ± 0.0325593	0.825763 ± 0.0229039
5,000	0.75585 ± 0.0253655	0.700457 ± 0.0132724	0.85324 ± 0.015919
10,000	0.79188 ± 0.0163628	0.741884 ± 0.0127872	0.853419 ± 0.00656938
15,000	0.787266 ± 0.00870219	0.733619 ± 0.014711	-

Table 10: The AUC for the MAGIC Gamma Telescope dataset in ms for VDT, Fast KNN and the Exact method.

Problem Size	VDT	Fast KNN	Exact
1,000	196 ± 7.84	342 ± 11.4287	$5,916 \pm 19.204$
5,000	$1,194 \pm 44.952$	2030 ± 56.8062	$155,096 \pm 104.929$
10,000	$2,654 \pm 249.821$	$4,296 \pm 41.8541$	$603,628 \pm 13,643.6$
20,000	$6,066 \pm 719.259$	$10,160 \pm 1103.78$	-
40,000	$12,070 \pm 305.848$	$20,906 \pm 775.838$	-
60,000	$20,168 \pm 1735.92$	$32,080 \pm 201.794$	-
80,000	$25,838 \pm 658.029$	$47,120 \pm 2970.37$	-
100,000	$33,754 \pm 767.099$	$58,266 \pm 1,112.11$	-
120,000	$38,656 \pm 269.997$	$68,140 \pm 836.119$	-

Table 11: The construction time for the MiniBooNE dataset in ms for VDT, Fast KNN and the Exact method

Problem Size	**VDT**	**Fast KNN**	**Exact**
1,000	216 ± 14.6673	172 ± 7.33365	$27,316 \pm 15.9231$
5,000	$1,410 \pm 132.789$	984 ± 54.95	$680,680 \pm 25.5553$
10,000	$3,488 \pm 554.004$	$2,158 \pm 239.361$	$2.74238e+06 \pm 38,840.7$
20,000	$11,796 \pm 1,368.5$	$6,222 \pm 434.192$	-
40,000	$27,116 \pm 1,089.07$	$13,696 \pm 187.751$	-
60,000	$44,672 \pm 1,221.32$	$22,580 \pm 950.448$	-
80,000	$62,352 \pm 1,519.87$	$32,002 \pm 1,293.92$	-
100,000	$80,668 \pm 1,527.7$	$40,294 \pm 174.275$	-
120,000	$78,600 \pm 8,607.4$	$35,932 \pm 3,025.11$	-

Table 12: The propagation time for the MiniBooNE dataset in ms for VDT, Fast KNN and the Exact method.

Problem Size	**VDT**	**Fast KNN**	**Exact**
1,000	0.816038 ± 0.0409891	0.713576 ± 0.0403268	0.882088 ± 0.0221806
5,000	0.857159 ± 0.0134527	0.787423 ± 0.0358348	0.920793 ± 0.00474857
10,000	0.827425 ± 0.0264457	0.799448 ± 0.0270987	0.924427 ± 0.0046791
20,000	0.848092 ± 0.025813	0.81865 ± 0.0124414	-
40,000	0.850231 ± 0.0105098	0.830827 ± 0.00928963	-
60,000	0.849074 ± 0.0181264	0.828904 ± 0.00443242	-
80,000	0.863991 ± 0.0129552	0.845533 ± 0.00670458	-
100,000	0.836672 ± 0.0541057	0.845337 ± 0.00819934	-
120,000	0.820719 ± 0.0411865	0.842622 ± 0.0126979	-

Table 13: The AUC for the MiniBooNE dataset for VDT, Fast KNN and the Exact method.

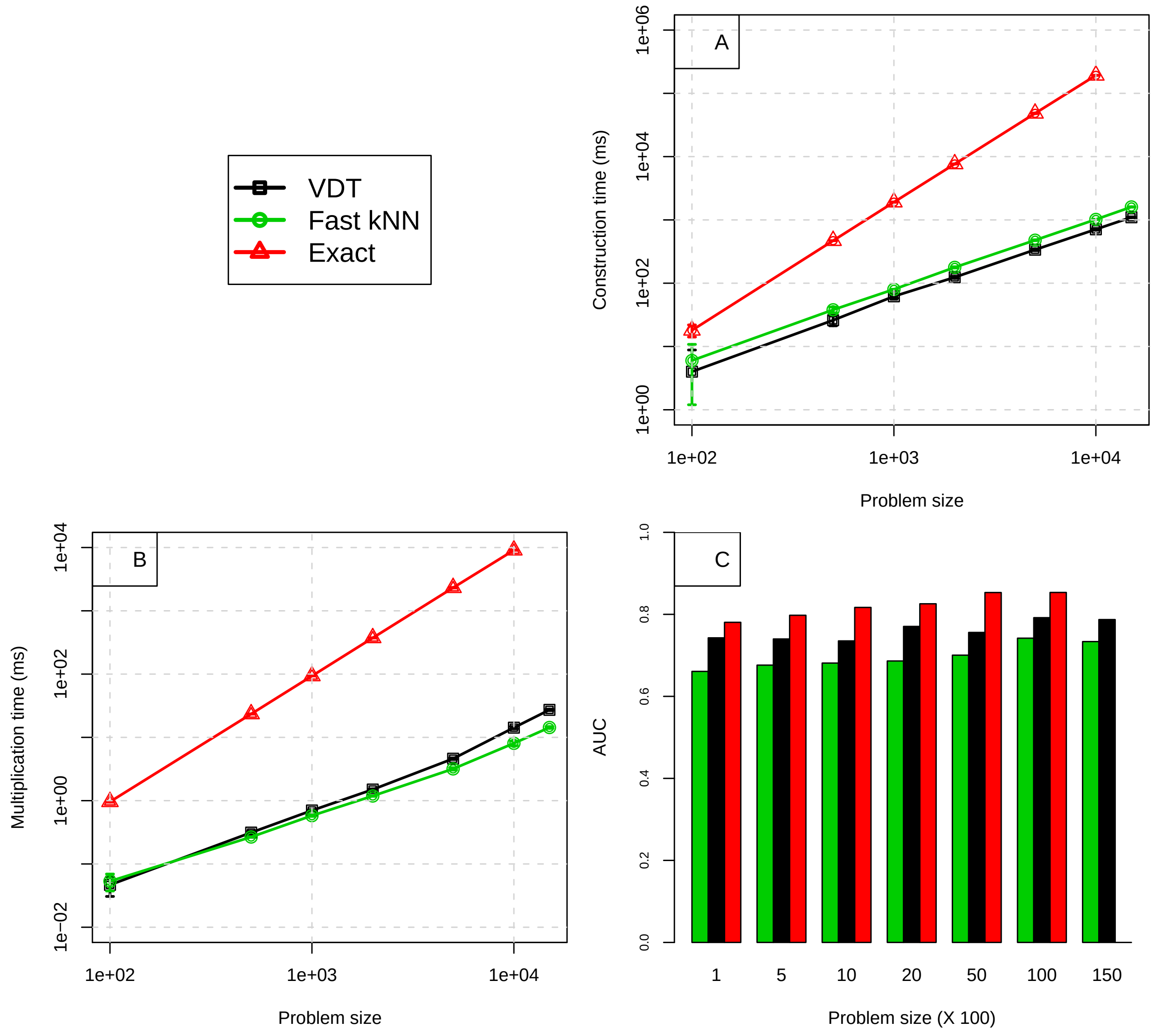

Figure 7: The efficiency and quality results for the VDT, the kNN and the Exact methods on MAGIC Gamma Telescope Dataset: (A) construction time vs. problem size, (B) propagation time for the LP algorithm vs. problem size, and (C) average AUC for different problem sizes.

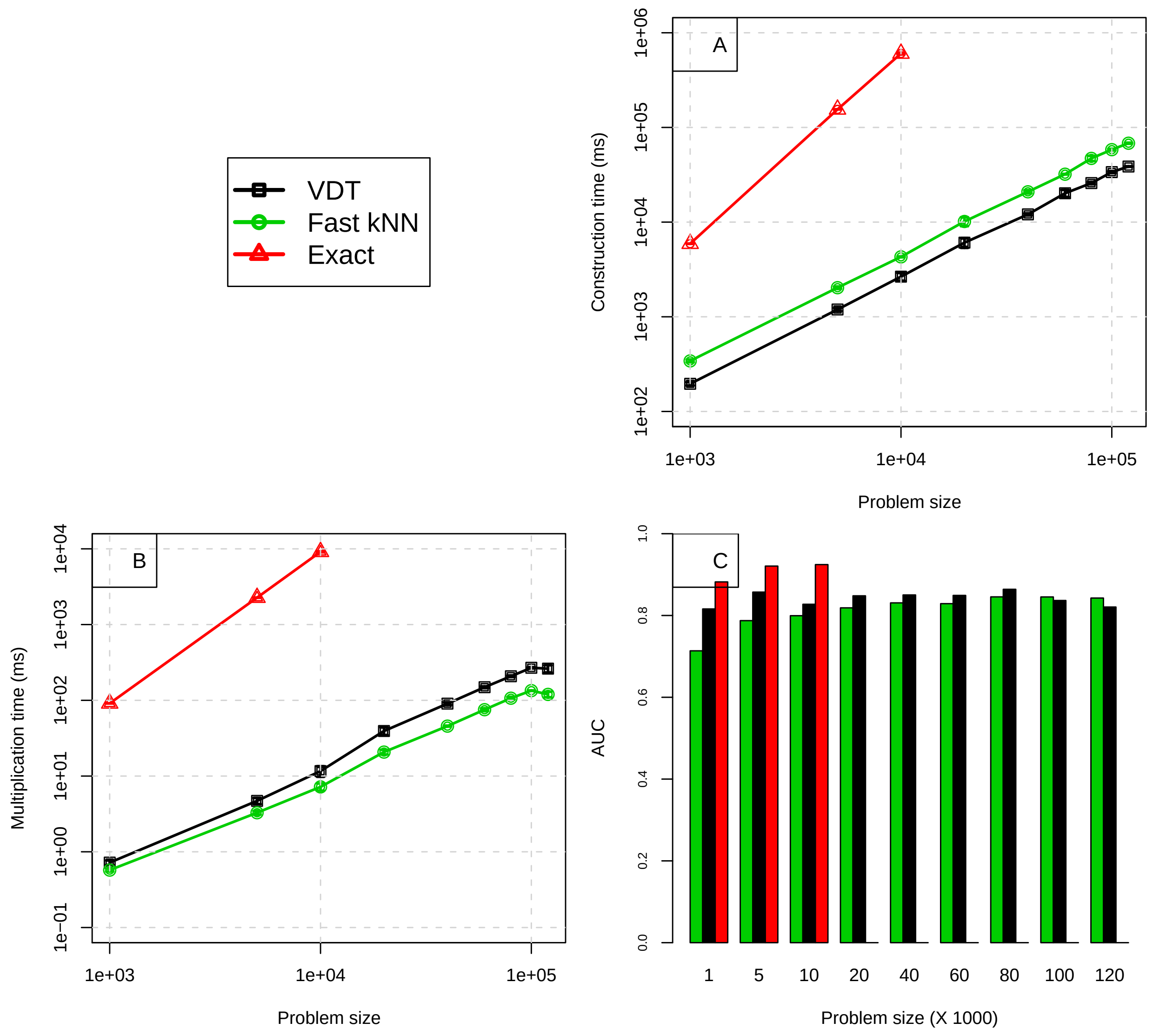

Figure 8: The efficiency and quality results for the VDT, the kNN and the Exact methods on MiniBooNE Dataset: (A) construction time vs. problem size, (B) propagation time for the LP algorithm vs. problem size, and (C) average AUC for different problem sizes.

3.8.3 The Effect of Refinement

In the second experiment, we study the efficiency and the effectiveness of the refinement process for the kNN and the VDT models. As illustrated in Section 3.6.4, through the refinement process, the number of blocks (or parameters) in the block-partitioned representation of the transition matrix is increased by splitting the existing blocks into smaller ones. By doing so, one effectively increases the number of variational parameters, hoping to improve the approximation accuracy at the cost of increased computational and memory complexity.

The refinement process, however, is not clear for the kNN model as there is no notion of block representation for this model. By defining the refinement process as augmentation of the parameter set used for representing the transition matrix, the refinement for the kNN model boils down to adding more edges to the kNN graph. Subsequently, one can employ different schemes to refine the kNN model; for example, at each refinement step, one can add the shortest edge that does not exist in the edge set to the graph. Of course, this simple algorithm is very costly since it involves sorting of *all* possible edges in the graph and therefore is impractical for large-scale problems. Here, we have adopted a simpler strategy of incrementing k by 1 at each refinement step. In other words, at each step, the number of parameters for the kNN model is increased by N. To augment a kNN graph to a k'NN graph where $k' > k$, we experimentally observed that it would be faster to build the k'NN graph from scratch using the Fast kNN algorithm than augmenting the old graph.

To perform a fair comparison between VDT and kNN in this section, we need to make sure they have the same number of parameters during the whole process; that is, $|\mathscr{B}| = kN$ at all times. To ensure this, we set the input argument $|\mathscr{B}|_{max}$ to Algorithm 8 such that after every call to this function, the total number of blocks in the VDT model is equal to kN. Finally, note that both the VDT and the kNN models converge to the Exact model at their finest level of approximation with the maximum N^2 number of parameters.

3.8.3.1 Experimental Setup To compare the efficiency and the effectiveness of the refinement process for the VDT and the kNN models, first we build the coarsest kNN ($k = 2$) and VDT ($|\mathscr{B}| = 2(N-1)$) models and then gradually refine each model to higher levels of refinement. At each level of refinement (where both methods have the same number of parameters; that is, $|\mathscr{B}| = kN$), we run the LP algorithm using 1% initial labeled datapoints and compute the AUC over the unlabeled data using the withheld labels. The label propagation is repeated 5 times, each time with a different random initial labeled set, and the average AUC is reported. Moreover, we have reported the time needed to refine each model as well as the initial model construction time. The refinement for both models is stopped when the number of parameters roughly reaches to $O(N \log N)$. This experiments has been performed on the same datasets introduced in Section 3.8.2.1 with the same parameter settings for the LP algorithm as in the first experiment. Unlike the first experiment, the two competing models are built over the whole datasets and are gradually refined afterwards.

3.8.3.2 Results Figure 9(A) shows the construction time (in ms) for the two models over the entire MAGIC Gamma Telescope Dataset. Also, Figure 9(B) shows the time needed for refinement for each of the models as the a function of the current parameter number (note that the Y-axis for both (A) and (B) plots is in the log-scale.) Finally, Figure 9(C) depicts the average AUC after the execution of the LP algorithm as a function of the parameter number; the margins in this plot show the 95% confidence intervals. Figure 10 illustrates the same plots for the MiniBooNE Dataset. Tables 14-15 show the AUC results in the tabular format along with their 95% confidence intervals.

As expected based on the results of the previous experiment, the VDT framework is faster than the Fast kNN algorithm in terms of the initial construction time. Also, due to the efficient implementation of the Symmetric Refinement algorithm (i.e. Algorithm 8) using the priority queue data structure, the time spent for refinement in the VDT framework is consistently lower than that of the kNN method. On the other hand, the average AUC has the general trend of increasing as the two models get more refined. This is specially the case for the MiniBooNE Dataset. For the MAGIC Gamma Telescope Dataset, the kNN model does not seem to behave consistently as the number parameters is increased (or the model gets more refined.) One possible explanation for such an inconsistent behavior can be the special geometry of this dataset in the space. In particular, imagine two clusters in the data from two different classes which are geometrically located close to each other. By adding more edges to the kNN graph (i.e. the refinement process), at some point, these two clusters will be directly connected to each other while one of them might be completely disconnected from the rest of the graph. In this case, the label propagation results can be misleading if one of these clusters is not represented in the initial labeled set, which in turn, leads to a lower average AUC with larger variation depending on the initial labeled set. On the other hand, this problem seems to be less issue if the graph is completely connected (with all the edge weighted according to their length) as in the Exact model. That is why the VDT model exhibits more robust behavior; because like the Exact model, the VDT framework produces a completely connected graph. In other words, due to the connected nature of the similarity graph representation in the VDT model, it is less sensitive to the geometry of data in the space as the refinement process progresses.

The other important observation in this experiment is the existence of some *internal structure* in both of the datasets. As the AUC plots suggest, despite some initial jumps, the AUC curves quickly converge as the number of parameters increases. In other words, the accuracy does not dramatically improve after a certain point which suggests that we do not really need that many number of parameters to represent the transition matrix for these datasets. This observation signifies the existence of some internal manifold structure in these problems which, in turn, advocates the application of the reduction-based approximation techniques such as the VDT and the kNN models.

Number of Parameters	**VDT**	**Fast KNN**
$38,038$	0.691988 ± 0.0330575	0.590564 ± 0.0274645
$95,098$	0.696513 ± 0.0319579	0.678875 ± 0.0435759
$133,138$	0.715664 ± 0.0165224	0.706024 ± 0.0169628
$171,178$	0.681347 ± 0.0314138	0.676082 ± 0.0661018
$209,218$	0.723785 ± 0.0214224	0.614568 ± 0.0366122
$247,258$	0.725393 ± 0.0127685	0.643211 ± 0.0548255
$285,298$	0.72133 ± 0.0157333	0.703146 ± 0.0322503

Table 14: The AUC during refinement for VDT and Fast KNN on the MAGIC Gamma Telescope dataset.

Number of Parameters	**VDT**	**Fast KNN**
$260,126$	0.708232 ± 0.0323947	0.762606 ± 0.0139435
$650,318$	0.847817 ± 0.0110448	0.836029 ± 0.00851859
$910,446$	0.848593 ± 0.017566	0.838004 ± 0.019564
$1,170,574$	0.841181 ± 0.0126945	0.865347 ± 0.0102216
$1,430,702$	0.858068 ± 0.0213771	0.883521 ± 0.00427182
$1,690,830$	0.86359 ± 0.0128275	0.88284 ± 0.00363055

Table 15: The AUC during refinement for VDT and Fast KNN on the MiniBooNE dataset.

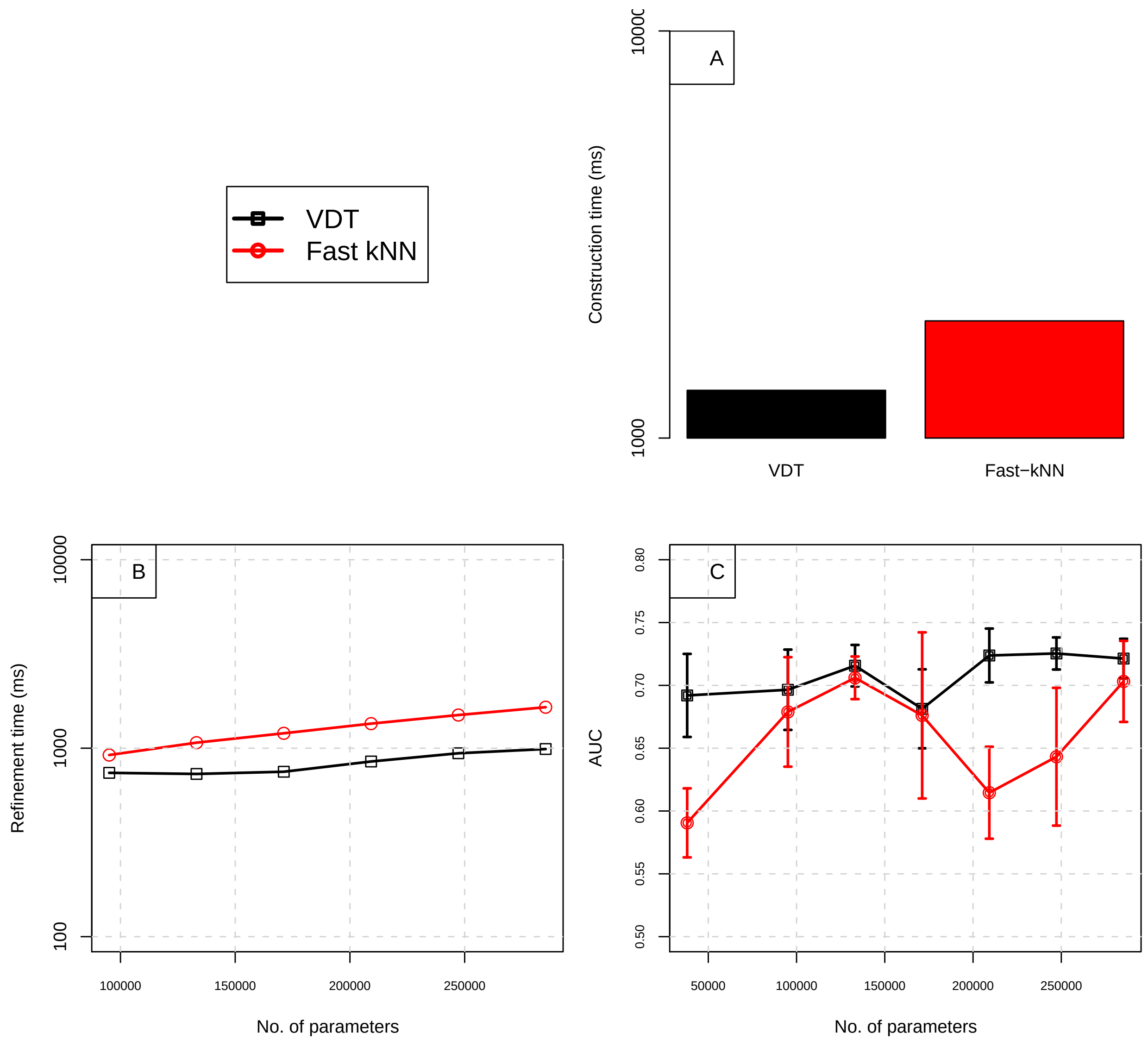

Figure 9: The refinement results for the VDT and the kNN methods on MAGIC Gamma Telescope Dataset: (A) the initial construction time, (B) refinement time vs. the number of parameters for each model, and (C) average AUC vs. the number of parameters for each model.

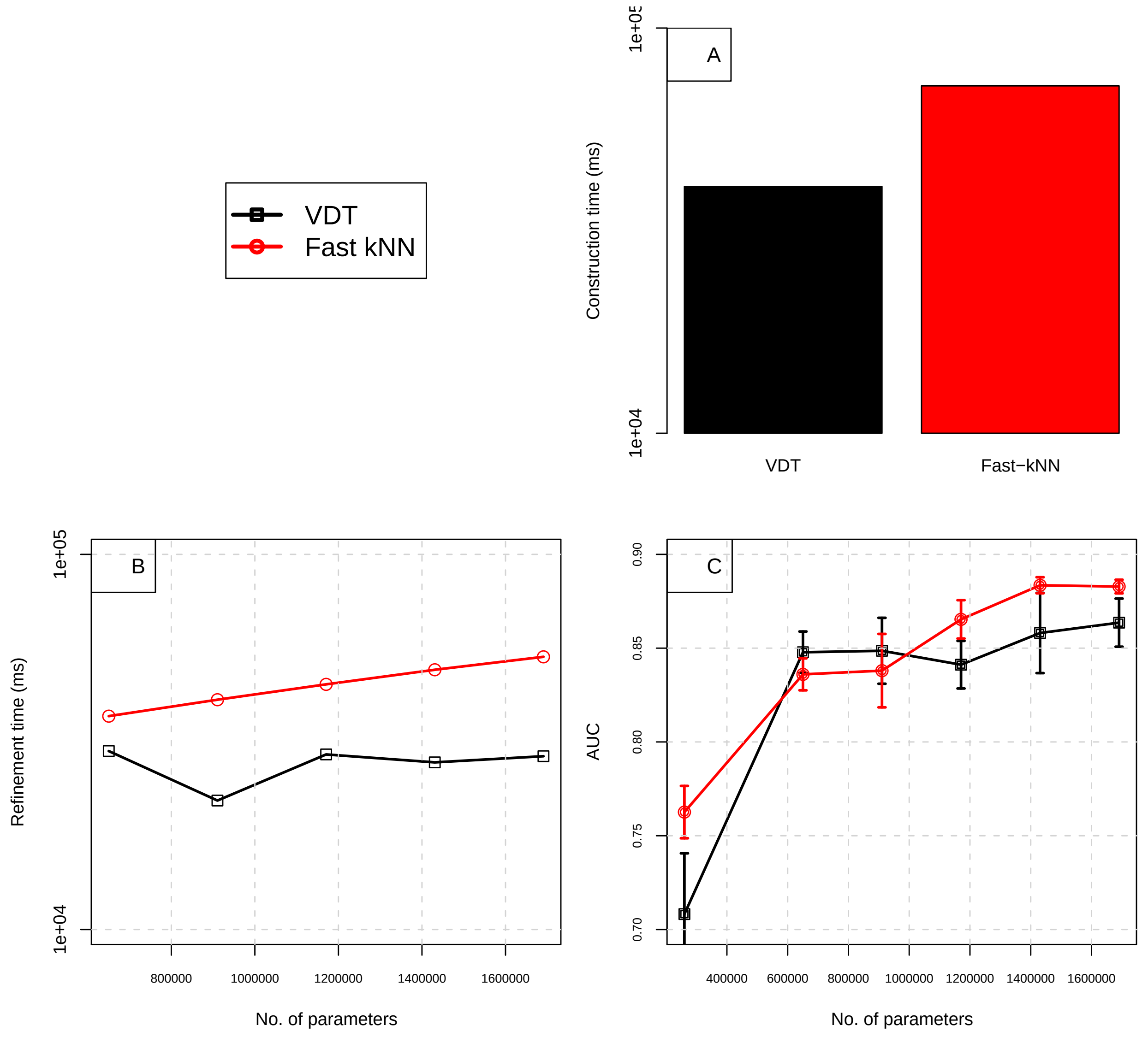

Figure 10: The refinement results for the VDT and the kNN methods on MiniBooNE Dataset: (A) the initial construction time, (B) refinement time vs. the number of parameters for each model, and (C) average AUC vs. the number of parameters for each model.

3.8.4 Computational Scalability

In the third experiment, we explore the applicability and the scalability of the proposed framework for very large datasets. The two data sets used in this experiment are taken from the Pascal Large-scale Learning Challenge.[7] The first data set, *alpha*, consists of 500,000 records with 500 dimensions. The second data set, *ocr*, is even larger with 3,500,000 records of 1156 features. Both data sets consist of 2 balanced classes. Table 16 shows the construction and propagations times as well as the number of model parameters when the VDT algorithm is applied. We could not apply other baseline methods for these data sets due to their infeasible construction times. Nevertheless, by extrapolating the graphs in Figure 8(A), we guess the Exact method, for example, would be roughly 3-4 orders of magnitude ($10^3 - 10^4$ times) slower. It should also be stressed that this experiment is specifically designed for showing the applicability of our framework for gigantic data sets and not necessarily showing its accuracy. According to these results, as a serial algorithm, our framework takes a reasonable time to construct and operate on these gigantic datasets. Furthermore, due to its tree-based structure, the VDT framework has the great potential to be parallelized which will make the algorithm even faster.

Data set	N	**Param#**	**Const.**	**Prop.**
alpha	0.5 M	1 M	4.5 hrs	11.7 min
ocr	3.5 M	7 M	46.2 hrs	93.3 min

Table 16: Very large-scale results.

3.8.5 BVDT For Frequency Data

In this section, we study the applicability and the performance of the Bregman extension of the Variational Dual-Tree framework (aka BVDT). In particular, we compare BVDT with the original Euclidean Variational Dual-Tree framework (VDT) for problems involving frequency data.

[7]http://largescale.ml.tu-berlin.de/

3.8.5.1 The Bregman Divergence For Frequency Data As seen in Section 3.7, in order to apply the BVDT framework to a real problem, the crucial step is to identify the appropriate Bregman divergence and its corresponding $\phi(\cdot)$ function in that specific problem. Knowing the corresponding exponential family that generated the data, on the other hand, is *not* necessary. However, in many problems, the choice of the appropriate Bregman divergence is not clear; instead, we know that the underlying data generation process belongs to *or* can be accurately modeled with exponential families. In such cases, one can systematically derive the appropriate Bregman divergence from the data generation process. This is exactly the case for the frequency data. As a result, in this section, we use this procedure to derive an appropriate Bregman divergence for the frequency data. However, before that, let us start with the formal definition of the frequency data.

Given a set of d events $\{e_j\}_{j=1}^{d}$, the frequency dataset $\mathscr{D}$ consists of N feature vectors where the j-th element of each vector represents the number of times that event e_j happens in that case for all $j \in \{1,\dots,d\}$. The length of each vector is defined as the total number of events happened in that case, i.e. $L_i = \sum_{j=1}^{d} x_i(j)$. For example, in text analysis, events can represent all the terms that appeared in a text corpus, while the data cases represent the documents. The frequency dataset in this case is the famous bag-of-words. We adopt the term-document analogy for the rest of this section.

We start the Bregman divergence derivation procedure by constructing a generative model for document generation. If the length of all documents was equal to constant L, one could model the document generation process with a mixture of Multinomials, where each mixture component had different term generation probabilities while sharing the same length parameter [Banerjee et al., 2005]. However, having the same length for all documents is not the case for most real datasets. To address this issue, we also model the length of document L as a positive discrete random variable with Poisson distribution. In particular, we propose a mixture model with K mixtures for document generation whose k-th component

($k \in [1..K]$) is modeled by the following generative model:

$$
\begin{aligned}
p_k(x, L; \alpha_k, \lambda_k) &= p(x \mid L; \boldsymbol{\alpha}_k) p(L; \lambda_k) \\
p(x \mid L; \boldsymbol{\alpha}_k) &= \frac{L!}{\prod_{j=1}^{d} x(j)!} \prod_{j=1}^{d} \alpha_k(j)^{x(j)} \\
p(L; \lambda_k) &= \frac{1}{L!} e^{-\lambda_k} (\lambda_k)^L
\end{aligned}
\tag{3.44}
$$

where, the length of a document, L, has a Poisson distribution $p(L;\lambda_k)$ with mean length λ_k and given L, document x has a Multinomial distribution $p(x \mid L; \boldsymbol{\alpha}_k)$ with the term probabilities $\boldsymbol{\alpha}_k = [\alpha_k(j)]_{j=1}^{d}$. Figure 11 shows the plate diagram for the proposed generative model.

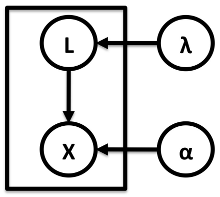

Figure 11: The plate diagram for the proposed document generation process: X is the unigram representation of a document, L is the document length, $\boldsymbol{\alpha}$ is the term probability vector and λ is the average document length.

By doing the algebra, $p_k(x, L; \alpha_k, \lambda_k)$ can be written in the form of Eq. (3.34), where we have:

$$
\begin{aligned}
\theta_k &= [\theta_k(j)]_{j=1}^{d} = \left[\log(\lambda_k \alpha_k(j))\right]_{j=1}^{d}, \\
\psi(\theta_k) &= \sum_{j=1}^{d} \exp(\theta_k(j)), p_0(x) = \left(\prod_{j=1}^{d} x(j)!\right)^{-1}
\end{aligned}
\tag{3.45}
$$

That is, our generative model also belongs to an exponential family. By deriving the conjugate function of the log-partition function $\psi(\cdot)$, we get the $\phi(\cdot)$ function and subsequently its corresponding Bregman divergence as:

$$\begin{aligned}\phi(x) &= \sum_{j=1}^{d} x(j)\log x(j) - \sum_{j=1}^{d} x(j)\\ d_\phi(x,y) &= \sum_{j=1}^{d}\left[x(j)\log\left(\frac{x(j)}{y(j)}\right) - x(j) + y(j)\right]\end{aligned} \tag{3.46}$$

Moreover, the mean parameter vector $\boldsymbol{\mu}_k$ can be derived using Eqs. (3.36) and (3.45) as:

$$\boldsymbol{\mu}_k = \nabla\psi(\boldsymbol{\theta}_k) = \left[\exp(\theta_k(j))\right]_{j=1}^{d} = \left[\alpha_k(j)\lambda_k\right]_{j=1}^{d} = \lambda_k\boldsymbol{\alpha}_k \tag{3.47}$$

In other words, our proposed generative model can be reformulated in the form of Eq. (3.35) with $\phi(\cdot)$ and $d_\phi(\cdot,\cdot)$ are given by Eq. (3.46) and $\boldsymbol{\mu}_k$ is given in Eq. (3.47). Furthermore, the divergence in Eq. (3.46) is called the *Generalized I-Divergence* (GID) which is a generalization of the KL-Divergence [Dhillon and Sra, 2005].

The key finding of the derivations in this section is that the GID is a more natural way of encoding distance between documents than the popular Euclidean distance if the documents are indeed generated according to the proposed generative model. The practical implication of such conclusion is that BVDT framework customized by the GID should outperform the traditional Euclidean VDT on the frequency data. To evaluate this hypothesis, we have run both methods on simulated data as well as real text datasets.

It should be emphasized that the sole purpose of the probabilistic modeling in this section is to derive the appropriate Bregman divergence for the frequency data; however, having found this divergence, the BVDT framework does not explicitly need the proposed probabilistic model to operate. This can also be corroborated by the fact that the distribution hyper-parameters λ_k and $\boldsymbol{\alpha}_k$ do not appear in the derived $\phi(\cdot)$ and $d_\phi(\cdot,\cdot)$ in Eq. (3.46).

3.8.5.2 Simulation The first step to evaluate the Bregman divergence derived from the generative model in Eq. (3.44) is to run the BVDT framework equipped with this divergence over the simulated data that has been actually generated from this generative model. To this end, we have setup a simulation experiment as follows:

Data: Each datapoint in our simulated dataset is a dense vector of bag-of-words (unigram) over 30 artificial terms (i.e. $d = 30$). The generative process is a mixture model with 5 components (i.e. $K = 5$) where each component represents one class and mixture weights are uniform ($= 0.2$). Furthermore, each mixture component is modeled with Eq. (3.44). The average length parameters are set as $\lambda_k = 100$ ($k = 1..5$). The term frequency vector $\boldsymbol{\alpha}_k$ for each class is depicted in Figure 12. The X-axes in this figure represent different terms while the Y-axes show their (normalized) frequencies. As the plots show, the 5 classes are highly overlapping with respect to the terms, yet each class has a distinct term frequency signature. Also note that since the average length of documents in this simulation is relatively short compared to the vocabulary size, the classes are highly overlapping in the Euclidean space which is already a sign that the Euclidean distance may not be the best way to express distance between the generated documents.

Methods: We have compared two methods in this experiment: (1) VDT-EUC: the original Euclidean VDT framework, and (2) BVDT-GID: the BVDT framework customized with the GID. It should be noted that, for this experiment, VDT-EUC and BVDT-GID are both kept at their coarsest level of approximation with the minimal number of variational parameters $2(N-1)$.

Experimental Setup: The setup for this experiment is similar to that of the experiment is Section 3.8.2; that is, we compute the construction and propagation times as well as the performance of the LP algorithm for the competing methods over different problem sizes. One difference here is since we have 5 classes in this problem, we compute and report the average classification accuracy (ACC) instead of the average AUC. Also, in order to make the problem more challenging, for each problem size, we only provide 1% labeled datapoints to the LP algorithm.

Results: Figure 13(A) shows the construction time (in ms) for the two models over the simulated dataset, as the problem size increases from 100 to 200,000. Also, Figure 13(B) shows the propagation time for the LP algorithm run on each of the models. Note that both of these axes in the plot are in the log-scale. Finally, Figure 13(C) depicts the average ACC computed over the unlabeled datapoints based on the true withheld labels as well as the inferred labels by the LP algorithm. Tables 17-19 show these results in the tabular format along with their 95% confidence intervals.

In terms of construction time, the two methods are very similar with VDT-EUC slightly faster for larger sample sizes. This difference can originate from the different tree structures resulted from different distance metrics (i.e. the Euclidean distance vs. the GID). The propagation time, however, is exactly the same for both methods. These results corroborate the fact that the BVDT framework has the same computational and memory complexity as the original VDT framework. However, in terms of accuracy, the story is completely different. The BVDT-GID method consistently and significantly outperforms the VDT-EUC method by over 10% margin in most cases. For the smallest sample size (i.e. 100) both methods have the same ACC around 20% which is basically the performance of the random classifier with 5 balanced classes suggesting a poor performance for both methods. However, as the sample size increases, both methods start to do better with the BVDT-GID method doing much better than the VDT-EUC model. These results show how the correct choice of distance for a given problem can make a huge difference in terms of the accuracy of the approximation made by the varitional dual-tree framework. Moreover, as we observed in this simulation experiment, BVDT-GID shows a consistent behavior as the sample size increases whereas the accuracy of VDT-EUC significantly degrades for larger sample sizes.

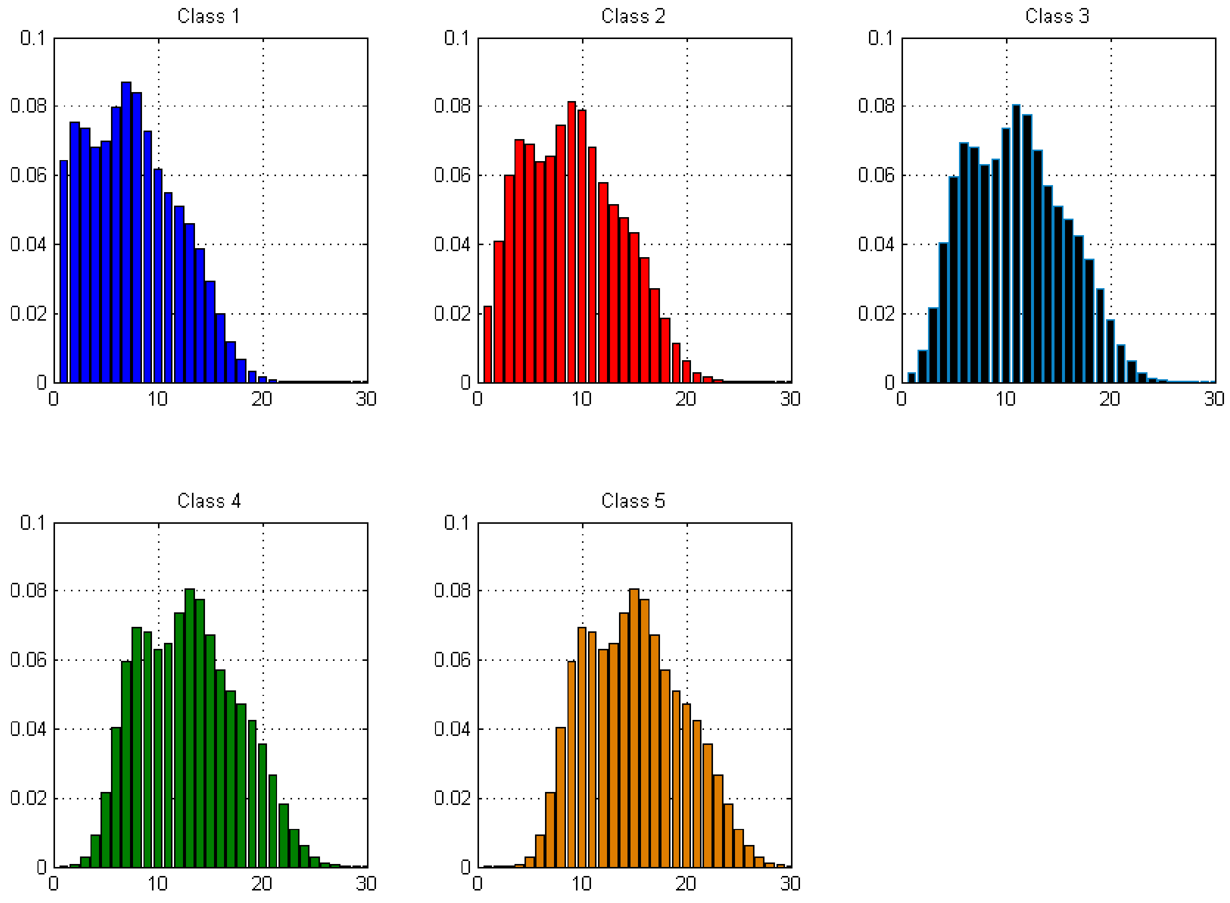

Figure 12: The term frequencies for each class in the Bregman simulation experiment for the frequency data: the X-axes represent different terms while the Y-axes shows their (normalized) frequencies for each class.

Problem Size	**VDT-EUC**	**BVDT-GID**
100	10 ± 0	10 ± 4.801
500	60 ± 0	88 ± 7.33365
1,000	136 ± 7.84	239 ± 51.3355
5,000	884 ± 18.1763	$2,027 \pm 213.397$
10,000	$2,032 \pm 105.44$	$5,046 \pm 236.617$
50,000	$13,310 \pm 696.834$	$46,914 \pm 3,157.26$
100,000	$30,072 \pm 722.477$	$120,275 \pm 3,861.28$
200,000	$69,696 \pm 2,692.53$	$317,332 \pm 23,729.9$

Table 17: The construction time for the simulated frequency data in ms for VDT-EUC and BVDT-GID.

Problem Size	**VDT-EUC**	**BVDT-GID**
100	$1.22000e+02 \pm 7.33365e+00$	$1.20000e+02 \pm 0.00000e+00$
500	$8.22000e+02 \pm 1.68606e+01$	$8.16000e+02 \pm 9.99408e+00$
$1,000$	$1.83200e+03 \pm 2.09271e+01$	$1.83600e+03 \pm 1.81763e+01$
$5,000$	$9.57000e+03 \pm 2.44882e+02$	$9.67400e+03 \pm 1.44376e+02$
$10,000$	$2.37200e+04 \pm 6.97441e+02$	$2.39240e+04 \pm 6.78018e+02$
$50,000$	$2.67990e+05 \pm 1.16214e+04$	$2.72276e+05 \pm 7.82356e+03$
$100,000$	$6.08816e+05 \pm 1.99214e+04$	$6.03774e+05 \pm 1.45048e+04$
$200,000$	$1.46436e+06 \pm 2.26842e+05$	$1.56729e+06 \pm 2.24165e+05$

Table 18: The propagation time for the simulated frequency data in ms for VDT-EUC and BVDT-GID.

Problem Size	**VDT-EUC**	**BVDT-GID**
100	$0.22 \pm 3.09903e-02$	$0.22 \pm 3.09903e-02$
500	$0.42 \pm 9.00150e-02$	$0.5856 \pm 1.04675e-01$
$1,000$	$0.6414 \pm 3.67407e-02$	$0.79888 \pm 3.64205e-02$
$1,0000$	$0.6653 \pm 4.94123e-02$	$0.82624 \pm 2.71510e-02$
$50,000$	$0.741852 \pm 1.76850e-02$	$0.869204 \pm 1.20239e-02$
$100,000$	$0.207320 \pm 1.00920e-03$	$0.865700 \pm 7.85178e-03$
$200,000$	$0.503117 \pm 6.42527e-03$	$0.881183 \pm 5.29533e-03$

Table 19: The accuracy time for the simulated frequency data in ms for VDT-EUC and BVDT-GID.

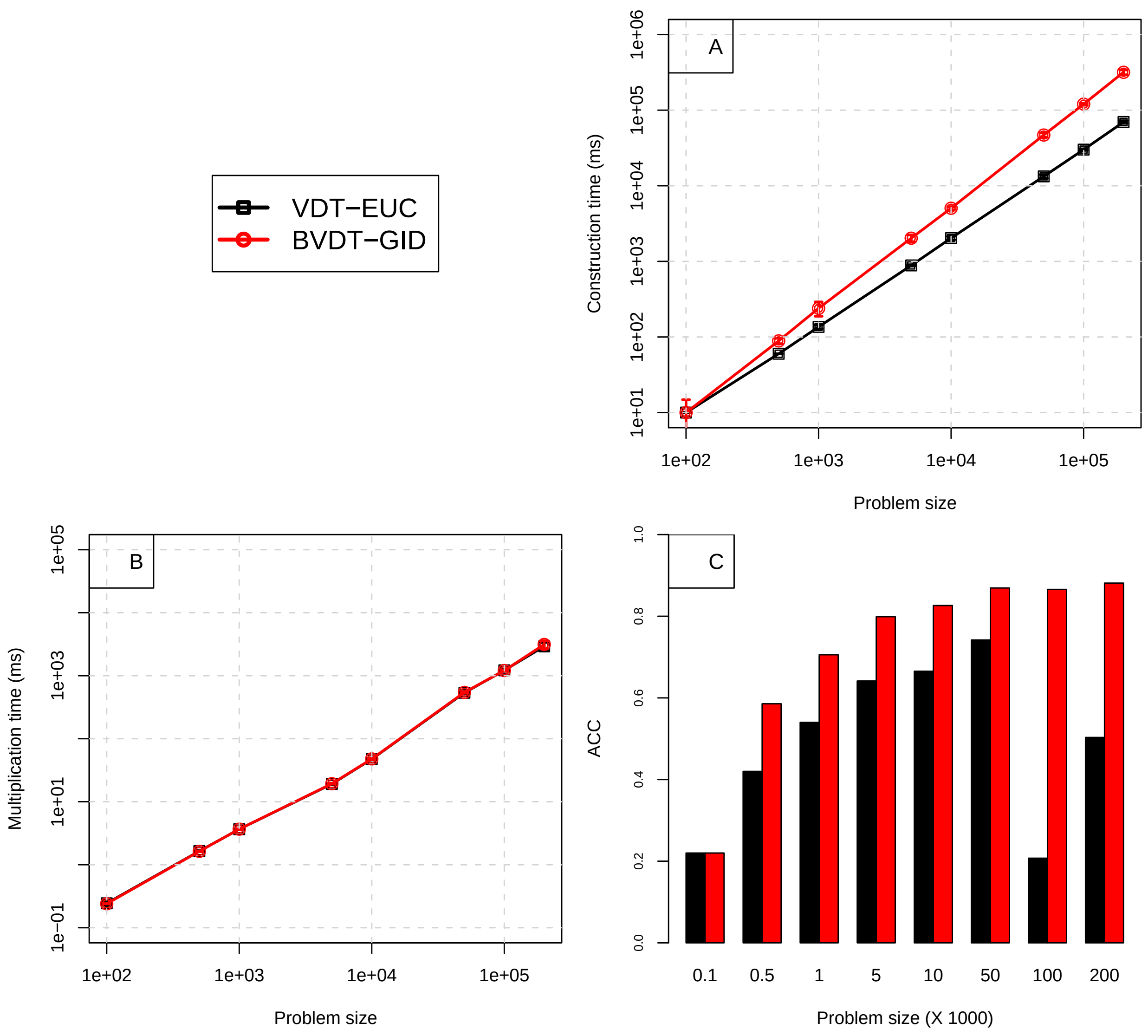

Figure 13: The efficiency and quality results for the BVDT-GID and the VDT-EUC methods on simulated frequency data: (A) construction time vs. problem size, (B) propagation time for the LP algorithm vs. problem size, and (C) average ACC for different problem sizes.

3.8.5.3 Text Data The simulation results of the previous section clearly justifies the importance of going beyond the Euclidean distance for certain problems involving the estimation of transition matrix. Yet, these results are not strikingly surprising: the data used in the previous section was exactly generated according the process which admits the GID as the true distance measure between the generated datapoints. The real question here is: can the BVDT-GID model still outperform the VDT-EUD model on frequency datasets which are *not* exactly generated according to the process in Section 3.8.5.1, in particular real text datasets? To answer this question, we have performed experiments on several real text datasets and reported the results in this section. In particular, the datasets used are:

(A) **BBC-Sport News Articles:** This dataset contains 737 sports news articles from the BBC Sport website from 2004-2005. Each document is represented as a unigram of 4,613 distinct terms. The main task on this data is document categorization based on the 5 topics (classes) of athletics, cricket, football, rugby and tennis [Greene and Cunningham, 2006].

(B) **BBC News Articles:** This dataset contains 2,225 news articles from the BBC News website from 2004-2005. Each document is represented as a unigram of 9,636 distinct terms. The main task on this data is document categorization based on the 5 topics (classes) of business, entertainment, politics, sport and tech [Greene and Cunningham, 2006].

(C) **20 Newsgroup:** This dataset is the train section of the well-known 20 Newsgroup dataset and contains 11,269 news documents each of which is represented by a unigram over 61,188 terms. The main task on this data is document categorization based on the 20 topics (classes) [Lang, 1995].

(D) **NSF Research Abstracts:** This dataset is a subset of the original NSF Research Award Abstracts. The dataset consists of 16,405 abstracts describing NSF awards for basic research from 1990 to 2003. Each abstract is represented as a unigram of 18,674 terms. The main task on this data is document categorization based on the 10 research programs (classes) [Deng et al., 2011].

(E) **Large Movie Reviews:** This dataset consists of 50,000 movie reviews where each review is represented as a unigram over 89,527 terms. The task on this data is sentiment classification of each review into either positive or negative sentiment. The two classes in this dataset are uniformly distributed among the reviews [Maas et al., 2011].

Table 20 summarizes the main properties of each dataset.

Dataset	**N**	**d**	**C**
BBC-Sport News Articles	737	4,613	5
BBC News Articles	2,225	9,636	5
20 Newsgroup	11,269	61,188	20
NSF Research Abstracts	16,405	18,674	10
Large Movie Reviews	50,000	89,527	2

Table 20: Summary of the text datasets used: N = number of documents, d = number of terms, C = number of classes.

Ratio	**VDT-EUC**	**BVDT-GID**	**Exact-EUC**	**Exact-GID**
0.05	0.411669 ± 0.0877717	0.657531 ± 0.0641116	0.440977 ± 0.0847812	0.832564 ± 0.0298591
0.1	0.544912 ± 0.0238994	0.746811 ± 0.0340012	0.451832 ± 0.0884955	0.879512 ± 0.00937991
0.2	0.678155 ± 0.0110102	0.820624 ± 0.00566653	0.640434 ± 0.0852056	0.903392 ± 0.00855162
0.3	0.769064 ± 0.0073508	0.880597 ± 0.00901856	0.719132 ± 0.0407943	0.933243 ± 0.00463687
0.4	0.795929 ± 0.0184749	0.897693 ± 0.005489	0.824695 ± 0.0157648	0.953053 ± 0.00813628

Table 21: The accuracy curves vs. labeled data ratio for the BBC Sport News dataset.

Ratio	**VDT-EUC**	**BVDT-GID**	**Exact-EUC**	**Exact-GID**
0.05	0.452854 ± 0.0123345	0.670652 ± 0.0167843	0.368899 ± 0.048935	0.784 ± 0.00464124
0.1	0.569708 ± 0.0177772	0.731685 ± 0.0135412	0.441798 ± 0.0162788	0.835056 ± 0.0111773
0.2	0.696449 ± 0.012843	0.80036 ± 0.00597714	0.56791 ± 0.0154009	0.873708 ± 0.00644925
0.3	0.77636 ± 0.0123408	0.841618 ± 0.00659557	0.703371 ± 0.0070582	0.90373 ± 0.00474705
0.4	0.834247 ± 0.00861933	0.879101 ± 0.00403679	0.773573 ± 0.0183945	0.920449 ± 0.00327239

Table 22: The accuracy curves vs. labeled data ratio for the BBC News dataset.

Ratio	**VDT-EUC**	**BVDT-GID**
0.05	0.125548 ± 0.00317266	0.205449 ± 0.00538628
0.1	0.189476 ± 0.00687521	0.281498 ± 0.00486172
0.2	0.294596 ± 0.00345168	0.393824 ± 0.00306084
0.3	0.400089 ± 0.000445476	0.488224 ± 0.00283637
0.4	0.495235 ± 0.00182368	0.577833 ± 0.00206427

Table 23: The accuracy curves vs. labeled data ratio for the 20 Newsgroup dataset.

Ratio	**VDT-EUC**	**BVDT-GID**
0.05	0.337446 ± 0.0100813	0.535069 ± 0.00470248
0.1	0.451667 ± 0.00961632	0.60239 ± 0.00374731
0.2	0.585675 ± 0.00687468	0.69007 ± 0.00392903
0.3	0.679086 ± 0.00398567	0.754648 ± 0.00502325
0.4	0.759037 ± 0.0048186	0.805401 ± 0.0024182

Table 24: The accuracy curves vs. labeled data ratio for the NSF Research Abstracts dataset.

Ratio	**VDT-EUC**	**BVDT-GID**
0.05	0.518916 ± 0.00294696	0.591148 ± 0.00376547
0.1	0.562752 ± 0.0015761	0.63286 ± 0.00299815
0.2	0.636304 ± 0.00148879	0.685188 ± 0.00265278
0.3	0.68694 ± 0.000796429	0.722364 ± 0.00135987
0.4	0.733676 ± 0.00141039	0.756976 ± 0.00036655

Table 25: The accuracy curves vs. labeled data ratio for the Large Movie Reviews dataset.

Experimental Setup: In addition to the two competing methods from the previous section, we have run two Exact methods on the text datasets described above: (1) Exact-EUC: the Exact method of Section 3.8.1 and (2) Exact-GID: the Exact method with the Euclidean distance replaced by the GID. For each dataset, we have built the four models on the entire dataset and then run the LP algorithm given a subset of labeled documents. We have gradually increased the percentage of labeled data from 5% to 40% and measured the ACC over the unlabeled documents. For each labeled sample size, we have repeated the experiment 5 times and have reported the average ACC. As for the LP algorithm, the α parameter in Eq. (3.25) is set to 0.01 and the number of iterations T is set to 300.

Results: Figures 14(A-E) show the average classification accuracy vs. the percentage of labeled data for the described datasets, respectively. The plots show the average of 5 trials with 95% confidence intervals. Although, none of the actual datasets is generated by the generative model proposed in Eq. (3.44), the BVDT with GID method consistently and significantly outperforms the Euclidean VDT. In particular, these results show that (a) the Generalized I-Divergence derived from the proposed generative model for text data captures the document similarity much better than the Euclidean distance does, and (b) the BVDT framework provides a straightforward mechanism to extend the variational dual-tree method beyond the Euclidean distance to use Bregman divergences such as GID. Tables 21-25 show the accuracy results in the tabular format along with their 95% confidence intervals.

We have also applied the Exact methods to the two smallest datasets. Not surprisingly, the Exact method with GID has the best performance compared to other methods. However, the Exact method with a wrong distance metric (the Euclidean distance in this case) can do even worse than the VDT method with Euclidean distance, as observed for the second BBC dataset. We conjecture the reason for such a behavior as the existence of block regularization in the VDT framework which compensates for the improper distance to some degree.

Finally, we note the computational complexity of the aforementioned methods vs. the dataset size (i.e. the number of documents) shown in Figure 14(F). Both X an Y axes in this plot are in the log-scale. As the plot shows while Euclidean VDT and BVDT have the same order complexity, they both are orders of magnitude faster than the Exact methods. In other words, while significantly improving on learning accuracy, BVDT still enjoys the same computational benefits as VDT.

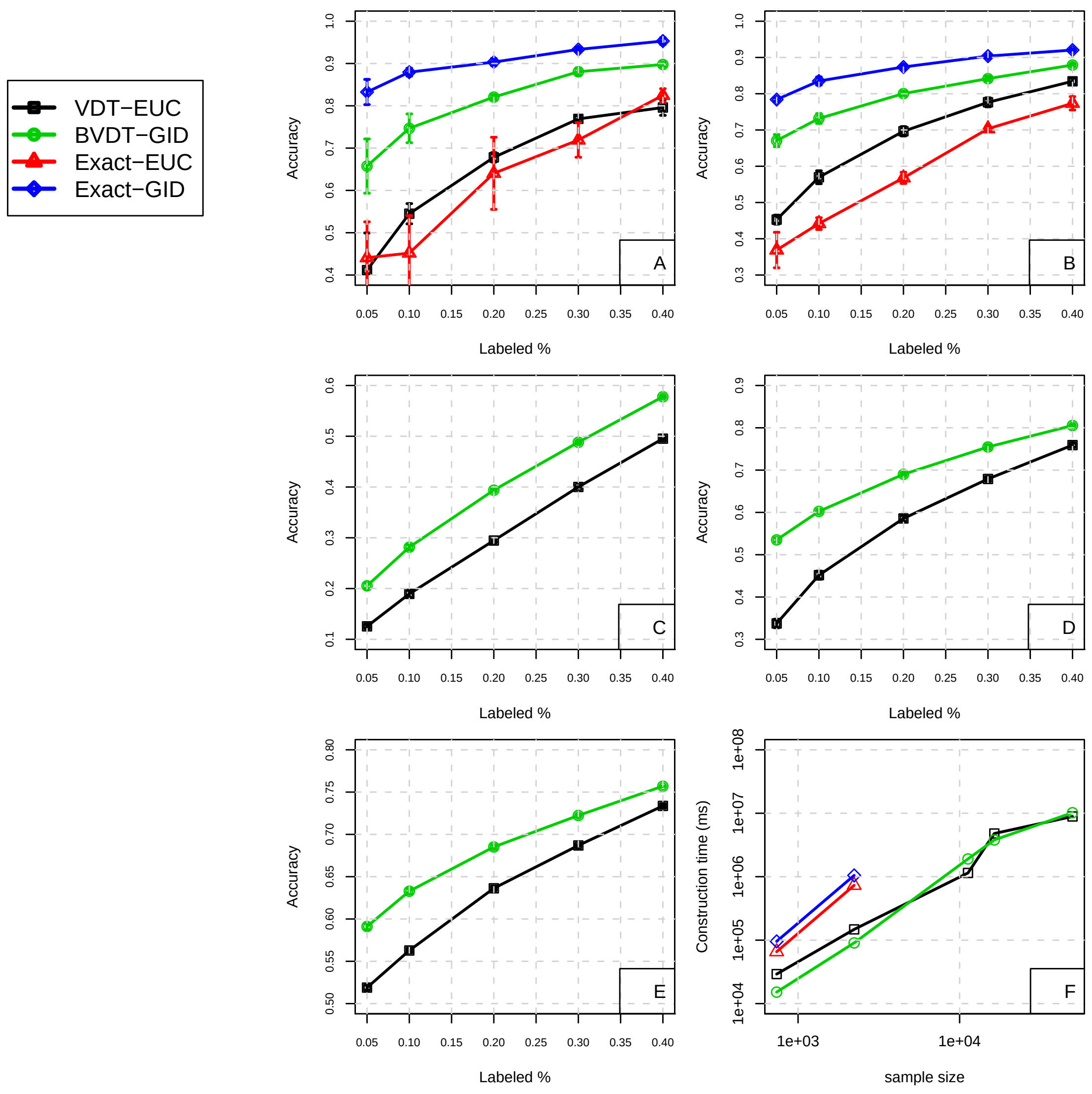

Figure 14: The accuracy curves vs. labeled data % for (A) BBC Sport News (B) BBC News (C) 20 Newsgroup (D) NSF Research Abstracts (E) Large Movie Reviews. (F) The computational complexity of four methods vs. the dataset size.

3.9 DISCUSSION

In this chapter, we proposed a scalable approximation framework based on a variational dual-tree method to compute large-scale transition matrices of random walk on large-scale data graphs. As we showed, using our framework, the transition matrix of random walk can be constructed in $O(N)$ if a hierarchical clustering of data is given. If the hierarchy is not given, one can build the cluster hierarchy using some fast approximate methods. In this work, we used anchor trees which take $O(N^{1.5}\log N)$ construction time in average. In terms of the memory complexity, as we discussed, our proposed method can take as low as $O(N)$ space to maintain the transition matrix. Compared to $O(N^2)$ computational and memory complexity of the direct method for computing transition matrices, our framework makes a huge difference for large-scale problems. Furthermore, we developed an unsupervised optimization technique to find the bandwidth for the Gaussian similarity kernel used in building the transition matrix. Algorithmically, we extended the variational dual-tree framework with a fast multiplication algorithm which is the crucial operation in using random walk on graphs. As we showed, using our framework, the transition matrix can be multiplied by an arbitrary vector as cheap as $O(N)$ compared to the $O(N^2)$ complexity of the direct multiplication. This fast multiplication algorithm then can be used in applications such as large-scale label propagation and eigen-decomposition of the transition matrix.

The key idea behind scalability of our framework is hierarchical parameter sharing. The other common technique for the same scalability purpose is sparsification. To compare these two general ideas, we compared our framework with the well-known k-nearest-neighbor (kNN) method which represents the latter idea in the domain of graphs. To this end, first we provided a theoretical complexity comparison between the two methods. Then, in the experiments, we demonstrated that while both the kNN based method and our method come close in terms of the construction and inference (propagation) times (with our method slightly better in terms of the construction time), our method significantly outperforms the kNN method in terms of the quality of approximation. We also experimentally showed that our method exhibits more robust behavior in the refinement process compared to the kNN model. We attributed the success of our proposed framework to the fact that its underlying graph representation is always connected. However, for the kNN framework, depending on the data, the resulted graph can be disconnected which in turn produces unstable results for certain applications such as label propagation. For such applications, our parameter sharing framework has a clear advantage over the sparse graphs which are inherently prune to disconnectivity. Furthermore, to verify its practicality, we showed that our framework is able to scale up to gigantic data sets with millions of records.

In the last part of this chapter, we extended our proposed framework to non-Euclidean spaces to use the general class of Bregman divergences (of which the Euclidean distance is a member). We call this general framework the Bregman Variational Dual-Tree framework (BVDT). The key advantage of the BVDT framework is it covers a large class of distances and divergences used in Machine Learning and therefore makes the variational dual-trees accessible to many non-Euclidean large-scale datasets. The crucial aspect of our generalization to Bregman divergences is, unlike generalizing VDT to an arbitrary distance metric, it comes with no extra computational cost; that is, its computational order is the same as that of the VDT framework. This is very important to the development of whole framework since the variational dual-trees are originally designed to tackle large-scale problems. To achieve this, we utilized the functional form of the general Bregman divergence to design a bottom-up mechanism to cut unnecessary distance computations similar to that of the Euclidean VDT framework.

Furthermore, by exploiting the connection between the Bregman divergences and the exponential families, we provided a probabilistic view of our model. By a walk-through example, we showed that this probabilistic view can be used to derive the appropriate Bregman divergence for those domains where the best choice of distance in not apparent at the first glance. This example also provides us with a powerful construction procedure to develop the appropriate Bregman divergence for a given problem. Specifically, we used this procedure to derive the Generalized I-Divergence (GID) for the frequency data. We showed that by incorporating GID in the BVDT framework, our model significantly improved the accuracy of learning for semi-supervised learning on various text datasets as well as simulated data, while maintaining the same order of complexity as the original VDT framework. Although in our experiments, we used the BVDT framework with only one type of Bregman divergence, the proposed model is general and can be customized with any member of the Bregman divergence family.

Finally, it should be noted that although the focus of this work is the approximation of transition matrices, our results also open the door to fast solutions for other large-scale problems in Machine Learning. For example, one immediate byproduct of the proposed Bregman Variational Dual-tree framework is that it can be used for fast kernel density estimation and/or approximation of the log-likelihood for large-scale non-Euclidean problems. Also, the Bregman extension of anchor trees can be used for building partition-trees over the large-scale non-Euclidean data for many applications which require a fast algorithm for hierarchical clustering.

4.0 LARGE D

4.1 INTRODUCTION

As mentioned in the previous chapter, the high dimensionality can hurt many of the methods proposed for the Large N problem. The complexity of some methods such as kd-trees directly depends on the dimensionality while for other methods like cover-trees and anchor-trees, this dependence is of more indirect nature through the intrinsic (manifold) structure of the data. In particular, if there exists a low-dimensional structure in the data, many of the approximation techniques proposed for the Large N problem can result in an accurate approximation while keeping the computational costs low. Conversely, if such structure does not exist or is high-dimensional, getting a good approximation may require costly computational resources.

However, the problem is even more fundamental and in fact has a *statistical* nature. More specifically, assuming the datapoints are randomly sampled from some *true* population, the structure represented by the finite sample in high dimension may not be the true representative of the population's structure. One way to see this intuitively is as follows: as the number of features grows the (Euclidean) distances in the input space become larger and as a result the difference between them becomes relatively less significant. This means that with the fixed number of datapoints, as we increase the dimension, we start losing the *true* (manifold) intrinsic structure in the data. Therefore, if one is interested to estimate the quantities of the true population structure (e.g. the clusters in spectral clustering), one needs to (exponentially) increase the number of datapoints as the dimensionality increases, and this points to the well-known *curse of dimensionality* problem which we refer to as the "Large d" problem in this chapter.

The very first solution to the Large d problem is to reduce the dimensionality. Many classical methods for dimensionality reduction, however, are not suitable for our problem just because they do not preserve the intrinsic structure in the data. Graph-based methods which preserve the intrinsic structure do not seem to be practical either. The reason is that given a fixed sample size in high dimension, the similarity graph does not represent the true manifold structure of the data based on the argument presented above. In fact, trying to naively use the graph-based methods in this case, creates a *chicken and egg* problem. As a result, the Large d problem for graph-based methods seems hopeless unless we have some *extra* information about the input space that can help us defeat the curse of dimensionality. In particular, as we see in this chapter, knowing the independence relations among the covariates in the input space can be statistically very helpful dealing with graph-based methods in high dimension.

In this chapter, we restrict our analysis and discussion to a specific class of the graph-based methods called *diffusion maps* (previously introduced in **??**). Diffusion maps are among the most powerful Machine Learning tools to analyze and work with complex high-dimensional datasets. In a nutshell, a diffusion map defines a lower dimensional embedding of the data that preserves the cluster structure of the data at different resolutions. This methodology has been successfully applied to a variety of clustering and semi-supervised learning tasks [von Luxburg et al., 2008, Bach and Jordan, 2004, Ng et al., 2001a, Shi and Malik, 2000, Zelnik-Manor and Perona, 2005, Valizadegan et al., 2008, Zhou et al., 2003, Zhu et al., 2003]. Unfortunately, the estimation of these maps from a finite sample is known to suffer from the curse of dimensionality. Motivated by other machine learning models for which the existence of structure in the underlying distribution of data can reduce the complexity of estimation, we study and show how the factorization of the underlying distribution into independent subspaces can help us to estimate diffusion maps more accurately. Building upon this result, we propose and develop an algorithm that can automatically factorize a high dimensional data space in order to minimize the error of estimation of its diffusion map, even in the case when the underlying distribution is not decomposable. Experiments on both the synthetic and real-world datasets demonstrate improved estimation performance of our method over the standard diffusion-map framework.

As a side step, later in this chapter, we discuss the graph-based methods for the case of functional spaces where the dimensionality is infinite. In particular, we consider the problem of clustering the human brain fibers where each datapoint (i.e. a fiber) is a 3D curve with a variable length that is not described by a fixed-length feature vector.

4.2 THE FACTORIZED DIFFUSION APPROXIMATION

4.2.1 Motivation

Recent theoretical studies have shown that under some reasonable regulatory conditions, the data-based diffusion (Laplacian) matrix asymptotically converge to certain operators defined based on the true distribution of data [Lee and Wasserman, 2010, von Luxburg et al., 2008, Singer, 2006, Gine and Koltchinskii, 2006]. In fact, the eigenfunctions of these asymptotic operators encode the cluster structure of the underlying density of data. An important question is how well we can estimate these eigenfunctions from a finite sample. Unfortunately it has turned out that the rate of convergence of the finite-sample approximations to true eigenfunctions is exponential in the dimension of the data, hence the problem suffers from the curse of dimensionality [Lee and Wasserman, 2010].

One possible way to alleviate the curse of dimensionality problem is based on the factorization of the input space into independent subspaces. Nadler et al. [Nadler et al., 2006] showed that when the underlying distribution is decomposable, it leads to the decomposition of the diffusion eigenfunctions. This idea was later used by Fergus et al. [Fergus et al., 2009] to implement scalable semi-supervised learning in large-scale image datasets. However, the scope of that work is somewhat limited. First, it assumes fully decomposable distributions. Second, it does not provide any insight on the quality of eigenfunctions built using the decomposition.

In this section, we study how the factorization of the underlying distribution into independent subspaces can help us to approximate the true eigenfunctions from a finite sample more accurately. We show that if the underlying distribution is factorizable, we can get significant reductions in the error bound for estimating the major eigenfunctions. In fact, this is analogous to machine learning criteria and models that rely on the underlying distribution structure to compensate for the insufficient sample size. It should be noted that the proposed framework in this section have been developed in collaboration with Hamed Valizadegan, Postdoc at University of Pittsburgh [Amizadeh et al., 2012b].

To clarify this idea, consider the synthetic 3D dataset shown in Figure 19 with four clusters. The first row in the figure shows the four major diffusion eigenfunctions of a sample with 1000 points where the color code shows the sign of the eigenfunctions. Each eigenfunction effectively separates the clusters from a different angle so that as a whole these eigenfunctions discovers the overall cluster structure in the data. Now, if we decrease the sample size to a half, the eigenfunctions and therefore the clustering result will be perturbed (as shown in the second row). However, this specific dataset is generated in a way that the Z coordinate is independent of X and Y. If we incorporate this information using the framework proposed in this paper, we get the same result as the first row but now only with 500 points (as shown in the third row).

As the above example shows, having a factorizable underlying distribution can speed up the convergence of the empirical eigenvectors to true eigenfunctions. However, the distributions of real-world data may not factorize into independent subspaces. In this case, the key question is how much error on diffusion eigenfunctions is introduced if we *impose* such independence assumptions upon non-decomposable distributions. This idea is similar to imposing structural assumptions for model selection in order to decrease the parameter learning complexity. In this chapter, we study the trade-off between the estimation error of diffusion eigenfunctions and the approximation error introduced by imposing the independence assumptions on the underlying distribution. We propose and develop a greedy algorithm which considers different independence assumptions on the distribution in order to find a factorizable approximation that minimizes the error in estimates of diffusion eigenfunctions. To show the merits of our framework, we test it on clustering tasks using both synthetic and real-world datasets.

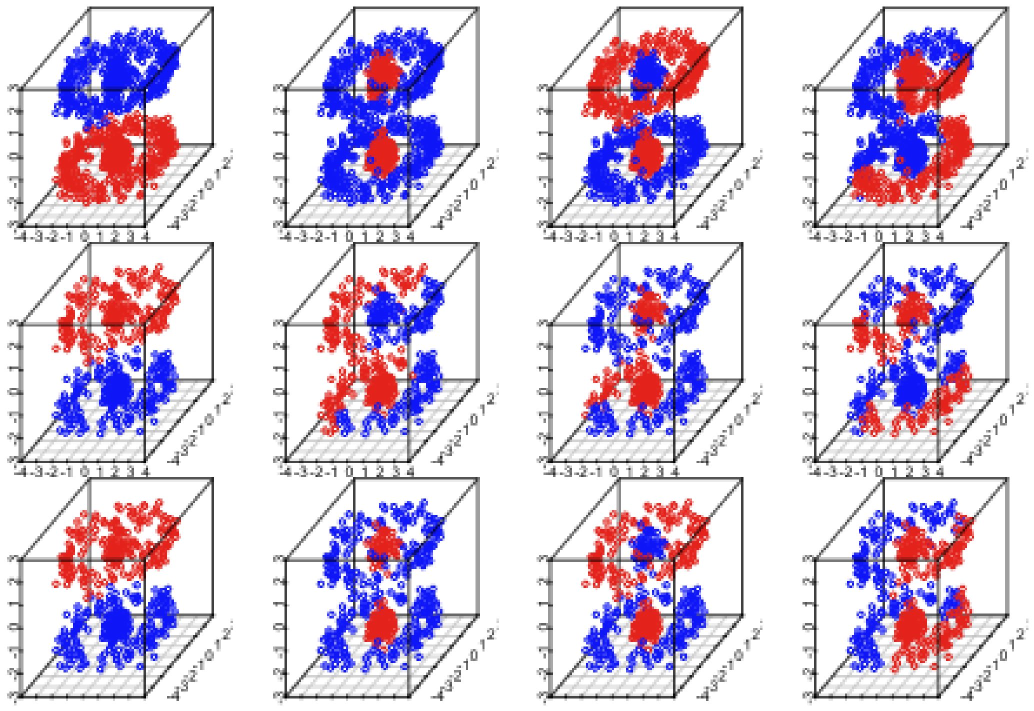

Figure 15: The synthetic 3D dataset with coordinate Z independent of X and Y.

4.2.2 Related Work

The framework presented in this chapter is related to a large body of existing work that utilize the Laplacian-based spectral analysis [Chung, 1997] of the similarity matrix of the data to find a low dimensional embedding of data. Such methods cover a wide range of learning tasks such as dimensionality reduction [Belkin and Niyogi, 2002, Lafon and Lee, 2006], data clustering [von Luxburg et al., 2008, Bach and Jordan, 2004, Jin et al., 2006, Ng et al., 2001a, Shi and Malik, 2000, Zelnik-Manor and Perona, 2005], and semi-supervised learning [Valizadegan et al., 2008, Zhou et al., 2003, Zhu et al., 2003]. While the focus of early works was more on developing new algorithms based on the spectral analysis of similarity metric [von Luxburg et al., 2008, Bach and Jordan, 2004, Jin et al., 2006, Ng et al., 2001a, Shi and Malik, 2000, Zelnik-Manor and Perona, 2005, Valizadegan et al., 2008, Zhou et al., 2003, Zhu et al., 2003], more recent works study the theoretical aspects of such analysis [Belkin and Niyogi, 2005, Lee and Wasserman, 2010, Nadler et al., 2006, Subramanya and Bilmes, 2009, Gine and Koltchinskii, 2006, Singer, 2006], leading to the development of the new algorithms [Belkin et al., 2006, Fergus et al., 2009].

Laplacian-based methods can be categorized into two major groups: locality-preserving methods [Belkin and Niyogi, 2002] and diffusion maps [Nadler et al., 2006], both of them are based on a similarity graph of data. These two groups differ in how they use the similarity metric. Locality-preserving methods aim at finding a mapping of data points to real values by minimizing the local variations of the mapping around the points. These methods can be considered as special cases of kernel PCA [Lee and Wasserman, 2010]. Diffusion maps are based on the random walk on the similarity graph of data and can be considered as non-linear version of Multidimensional Scaling (MDS) [Cox and Cox, 1994] technique. Many of these methods are built upon a similarity graph on the datapoints in the input space which has to be constructed based on the distance metric in the input space. One popular method to build the similarity graph is to transform the Euclidean distances into the similarity weights using the Gaussian similarity kernel with bandwidth σ.

Several authors study the convergence rate of the laplacian-based methods, either by assuming that data lie exactly on a lower dimensional Riemannian manifold in the original space [Gine and Koltchinskii, 2006, Singer, 2006, Belkin and Niyogi, 2005], or considering a general underlying distribution that generates data [von Luxburg et al., 2008, Lee and Wasserman, 2010, Fergus et al., 2009]. Our framework is built upon the result of the later group. For a fixed kernel bandwidth σ, von Luxburg et. al. [von Luxburg et al., 2008] showed that the normalized Laplacian operator converges with rate $O(1/\sqrt{N})$. Lee and Wasserman [Lee and Wasserman, 2010] study the rate of convergence for $\sigma \to 0$ and large sample size N and showed that the optimal rate is dependent on d, the dimension of data. This result, i.e. the dependency of the convergence rate to d, is the main inspiration for our work.

4.2.3 Convergence of Diffusion Operator

In this section, we study the convergence of the diffusion operator previously introduced in Section 1.2.5 of Chapter 1. As mentioned before, the diffusion map induced by the eigen decomposition of A_p^t is a powerful tool for data embedding at different continuous scales t. However, in practice, we have to estimate the eigen decomposition of A_p^t from the finite sample $\mathscr{D}$ by computing the matrix $[\hat{A}_\sigma]_{N\times N} = [k_\sigma(x^{(i)}, x^{(j)})/\sum_{l=1}^n k_\sigma(x^{(i)}, x^{(l)})]_{N\times N}$ with the eigen decomposition $\hat{A}_\sigma \hat{u}_{\sigma,i} = \hat{\lambda}_{\sigma,i}\hat{u}_{\sigma,i}$. The empirical eigenfunctions are then computed from the eigenvectors $\hat{u}_{\sigma,i}$ using the Nyström approximation [Lee and Wasserman, 2010]:

$$\hat{\psi}_{\sigma,i}(x) = \frac{\sum_{j=1}^N k_\sigma(x, x^{(j)})\hat{u}_{\sigma,i}(x^{(j)})}{\hat{\lambda}_{\sigma,i}\sum_{j=1}^N k_\sigma(x, x^{(j)})} \tag{4.1}$$

The eigenvalues and eigenfunctions of $\hat{A}_\sigma$ estimate their counterparts for $A_{p,\sigma}$ by estimating P with the empirical distribution $\hat{p} = 1/n$ (denoted by $\hat{A}_\sigma \to A_{p,\sigma}$ as $N \to \infty$) [Lee and Wasserman, 2010]. Since $\hat{A}_\sigma^{t/\sigma} \to A_{p,\sigma}^{t/\sigma}$ as $N \to \infty$, we can estimate the eigenspaces of A_p^t by those of $\hat{A}_\sigma^{t/\sigma}$. An important concern is how fast the rate of convergence is as $N \to \infty$. To answer this question, von Luxburg et. al. [von Luxburg et al., 2008] showed that the normalized Laplacian operator converges with rate $O(1/\sqrt{N})$ given that σ *is fixed*. This result can be easily extended to the diffusion operator as well. The good thing about this rate is that it does not depend on the dimension d. However, to find the optimal trade-off between bias and variance, we need to let $\sigma \to 0$ as the sample size N increases. Lee and Wasserman [Lee and Wasserman, 2010] showed that the optimal rate for σ is $(\log N/N)^{2/(d+8)}$ and therefore the eigenfunctions converge as [Lee and Wasserman, 2010, Giné and Guillou, 2002]:

$$\|\psi_{p,i}^t - \hat{\psi}_{\sigma,i}\|_2^2 = O_P\Big(\frac{t\sqrt{d}}{\mu_i^{(t)}}\Big[\frac{\log N}{N}\Big]^{2/(d+8)}\Big) \tag{4.2}$$

where, $\mu_i^{(t)} = \min_{2\leq l\leq i}\log(\lambda_{p,l-1}^{(t)}/\lambda_{p,l}^{(t)})$ is the multiplicative eigengap of A_p^t and we have $\|f\|_2^2 = \int_{\mathscr{X}} f^2(x)p(x)dx$. Unfortunately, this rate depends on the dimension exponentially which makes it a hard problem to estimate the eigenfunctions of A_p^t from a finite sample. Throughout the rest of this section, we drop the subscript σ for the empirical operators assuming it is implicitly computed using the optimal rate above.

4.2.4 Factorized Diffusion Maps

4.2.4.1 The Factorized Approximation Let $\mathscr{T}_{\boldsymbol{k}} = \{T_1, T_2, \dots, T_k\}$ be a partition of the variables in V into k disjoint subsets. Each T_i defines a subspace of V with dimension $d_i = |T_i|$. With a little abuse of notation, we also use T_i to refer to the subspace induced by the variables in T_i. We define the *marginal diffusion operator* $A^t_{p_i} : L^2(\mathscr{X}) \mapsto L^2(\mathscr{X})$:

$$A^t_{p_i}[g_z(x)] = \int_{T_i} a_t(x,y) g_z(y) p_i(y) dy,$$
$$x, y \in T_i, z \in V \setminus T_i \tag{4.3}$$

where $g_z(x) = f([x\ z]^T)$ assumes the variables in z are constants and p_i is the marginal distribution over the subspace defined by T_i. In other words, $A^t_{p_i}$ treats the input variables of $f(\cdot)$ which do not belong to T_i as constants. Furthermore, the partition $\mathscr{T}_{\boldsymbol{k}}$ defines the *factorized* distribution $q_{\mathscr{T}_{\boldsymbol{k}}} = \prod_{i=1}^k p_i$. To simplify the notation, for a fixed $\mathscr{T}_{\boldsymbol{k}}$, we refer to $q_{\mathscr{T}_{\boldsymbol{k}}}$ simply by q. We also define the *factorized diffusion operator* A^t_q the same as Eq. (1.19) with the true distribution p is replaced by the factorized distribution q. We have:

Lemma 1. *Let $\boldsymbol{\Lambda}^{\boldsymbol{t}}_{\boldsymbol{i}} = \{\lambda^{(t)}_{i,m} \mid 1 \le m \le \infty\}$ and $\boldsymbol{\Psi}^{\boldsymbol{t}}_{\boldsymbol{i}} = \{\psi^t_{i,m} \mid 1 \le m \le \infty\}$ be the set of eigenvalues and eigenfunctions of $A^t_{p_i}$, respectively. Then the sets:*

$$\boldsymbol{\Lambda}^{\boldsymbol{t}}_{\boldsymbol{q}} = \left\{\prod_{i=1}^k \xi_i \mid \xi_i \in \boldsymbol{\Lambda}^{\boldsymbol{t}}_{\boldsymbol{i}}\right\}, \boldsymbol{\Psi}^{\boldsymbol{t}}_{\boldsymbol{q}} = \left\{\prod_{i=1}^k \varphi_i \mid \varphi_i \in \boldsymbol{\Psi}^{\boldsymbol{t}}_{\boldsymbol{i}}\right\}$$

are respectively the eigenvalues and eigenfunctions of the factorized diffusion operator A^t_q.

Proof. Let $\mathscr{T}_{\boldsymbol{k}} = \{T_1, T_2, \dots, T_k\}$ be a partition of the variables in V and $q(V) = \prod_{i=1}^k p_i(T_i)$. Now assume $x = [x_1, x_2, \dots, x_k]^T$ and $y = [y_1, y_2, \dots, y_k]^T$ are two random vectors partitioned according to $\mathscr{T}_{\boldsymbol{k}}$ (note that x_i's and y_i's are sub-vectors). Then we will have:

$$a_\varepsilon(x,y) = \prod_{i=1}^k a_\varepsilon(x_i, y_i)$$

and therefore,

$$a_t(x,y) = \lim_{\varepsilon \to 0} a_{\varepsilon, t/\varepsilon}(x,y) = \lim_{\varepsilon \to 0} \prod_{i=1}^k a_{\varepsilon, t/\varepsilon}(x_i, y_i) = \prod_{i=1}^k a_t(x_i, y_i)$$

Now let $\psi^t(x) = \prod_{i=1}^k \psi_{i,m_i}^t(x_i)$ and $\lambda^{(t)} = \prod_{i=1}^k \lambda_{i,m_i}^{(t)}$ where $A_{p_i}^t[\psi_{i,m_i}^t(x_i)] = \lambda_{i,m_i}^{(t)} \psi_{i,m_i}^t(x_i)$, then we will have:

$$\begin{aligned} A_q^t[\psi^t(x)] &= \int a_t(x,y)\psi^t(y)q(y)dy \\ &= \prod_{i=1}^k \int a_t(x_i,y_i)\psi_{i,m_i}^t(y_i)p_i(y_i)dy_i \\ &= \prod_{i=1}^k \lambda_{i,m_i}^{(t)}\psi_{i,m_i}^t(x_i) = \lambda^{(t)}\psi^t(x) \end{aligned}$$

Therefore, $\psi^t(x)$ is an eigenfunction of A_q^t with eigenvalue $\lambda^{(t)}$. □

Lemma 1 explicitly relates the eigenvalues and the eigenfunctions of the factorized diffusion operator A_q^t to eigenvalues and the eigenfunctions of the marginal diffusion operators $A_{p_i}^t$. We refer to the eigenvalues and the eigenfunctions based on the factorization as *multiplicative* eigenvalues and eigenfunctions.

The above decomposition also gives a recipe for computing the eigenfunctions of A_q^t from eigenfunction estimates in each subspace T_i. In particular, we can estimate the eigenfunctions in each subspace T_i independently and then multiply the results over all subspaces. This construction procedure is of special practical significance if p, in fact, factorizes according to $\mathscr{T}_{\boldsymbol{k}}$; that is, $p = q$. In that case, the principal eigenfunctions of A_p^t (with largest eigenvalues) can be estimated from a finite sample $\mathscr{D}$ (more accurately), if we make use of the fact that p is factorizable.

The multiplicative eigenvalue and eigenfunction estimates on the full variable space using q come with the following properties. First, the largest eigenvalue $\hat{\lambda}_{i,1}^{(t)}$ in each subspace T_i is $\hat{\lambda}_{i,1}^{(t)} = 1$ and is associated with the constant eigenfunction $\hat{\psi}_{i,1}^t = 1$. Therefore, the largest multiplicative eigenvalue of A_q^t (according to Lemma 1) will be $\hat{\lambda}_{q,1}(t) = \prod_{i=1}^k 1 = 1$ with a constant eigenfunction $\hat{\psi}_{q,1}^t = \prod_{i=1}^k 1 = 1$. Next, the second largest multiplicative eigenvalue of A_q^t will be $\hat{\lambda}_{q,2}^{(t)} = \hat{\lambda}_{j,2} \times \prod_{i\neq j} 1$ with the eigenfunction $\hat{\psi}_{q,2}^t = \hat{\psi}_{j,2}^t \times \prod_{i\neq j} 1$ where $j = \arg\max_r \hat{\lambda}_{r,2}^{(t)}$. That is, the second eigenfunction of A_q^t can be obtained from only one subspace (i.e. T_j) with a reduced dimensionality (d_j). Finally, the m-th multiplicative eigenvalue and eigenfunction $\hat{\lambda}_m(t)$ and $\hat{\psi}_m^t$ will be estimated using at most $\lg m$ marginal eigenfunctions on subspaces.

Lemma 2. *Suppose p factorizes according to $\mathcal{T}_{\boldsymbol{k}}$ and the eigen decomposition of A_p^t is constructed using the procedure suggested by Lemma 1 then the m-th eigenfunction of A_p^t associated with its m-th largest eigenvalue is the multiplication of the marginal eigenfunctions from "at most" $\min(k, \lceil \lg m \rceil)$ subspaces in $\mathcal{T}_{\boldsymbol{k}}$.*

Proof. Suppose ψ is the m-th eigenfunction of A_p^t associated with the m-th largest eigenvalue λ constructed using Lemma 1. That is, we have that $\psi = \prod_{i=1}^k \psi_i$ and $\lambda = \prod_{i=1}^k \lambda_i$ where ψ_i is an eigenfunction of $A_{p_i}^t$ in the subspace T_i associated with eigenvalue λ_i. Now suppose, ψ consists of eigenfunctions from only $\ell < k$ subspaces; that is, only ℓ of the eigenfunctions in the product above are non-constant (non-trivial) with eigenvalues strictly less than 1, while the rest of them are constant with eigenvalues equal to 1. Now if any of these ℓ eigenfunctions is replaced by the constant eigenfunction (and its corresponding eigenvalue with 1) we will have a new valid pair of eigenvalue and eigenfunction $\langle \lambda', \psi' \rangle$ for $A_{p_i}^t$ where $\lambda' > \lambda$. Using this replacement method, we can generate 2^ℓ new pairs with eigenvalues all greater than λ. However, since λ is the m-th largest eigenvalue of A_p^t, we must have $m \geq 2^\ell$ or equivalently $\ell \leq \lceil \lg m \rceil$. On the other hand, the number involved subspaces ℓ cannot be greater than k which means that $\ell \leq \min(k, \lceil \lg m \rceil)$. □

From the estimation point of view, this result has a significant implication in that the estimation error of the m-th eigenfunction over the whole space can be reduced to the estimation error from at most $\lg m$ subspaces, each of which has a smaller dimension. To illustrate this, consider the second principal eigenfunction $\hat{\psi}_{q,2}^t$ described above. Since it depends only on one of the subspaces (with a reduced dimensionality), its rate of convergence, according to [Lee and Wasserman, 2010], should be faster. Hence its estimation error bound is reduced and equal to the error bound for that subspace. This observation further motivates the analysis of error for estimating the factorized diffusion map.

4.2.4.2 Error Analysis In the previous subsection, we saw that if the underlying distribution p is factorizable, then we can decrease the estimation error bound of the m-th principal eigenfunction using the factorization to independent subspaces. However, in reality, p may not factorize at all; then, the question is how much error is introduced if we *approximate* p with q, and under which problem settings we get smaller error bounds by enforcing such a factorization. Suppose $\hat{A}_q^t$ is the estimated factorized diffusion operator from a sample of size N with the factorization according to $\mathscr{T}_{\boldsymbol{k}}$. Let $\psi_{p,m}^t$, $\psi_{q,m}^t$ and $\hat{\psi}_{q,m}^t$ represent the m-th eigenfunction of A_p^t, A_q^t and $\hat{A}_q^t$, respectively. We want to approximate $\psi_{p,m}^t$ with $\hat{\psi}_{q,m}^t$ and to study the error $\|\psi_{p,m}^t - \hat{\psi}_{q,m}^t\|_2^2$. We have the following inequality:

$$\mathscr{E}_{total}(q,m,t) \triangleq \|\psi_{p,m}^t - \hat{\psi}_{q,m}^t\|_2^2 \leq 2\|\psi_{p,m}^t - \psi_{q,m}^t\|_2^2 + 2\|\psi_{q,m}^t - \hat{\psi}_{q,m}^t\|_2^2 \tag{4.4}$$

The first term on the right-hand side is the *approximation error or bias* which is due to approximating $\psi_{p,m}^t$ (i.e. p) with $\psi_{q,m}^t$ (i.e. q). Clearly, in case $p = q$, the approximation error is 0 and the inequality becomes an equality. Note that the approximation error is also a lower bound on $\mathscr{E}_{total}(q,m,t)$. The second term on the right-hand side is the *estimation error* of the factorized eigenfunction from the finite sample. We can bound these errors from above as follows:

Theorem 2. ***Upper bound on the approximation error:*** *Let* $\sup_{f:\|f\|_2 \leq 1} \|f\|_\infty = \ell < \infty$, $\sup_{x,y} a_t(x,y) = \jmath < \infty$ *and* $\delta_m = \lambda_{p,m}^{(t)} - \lambda_{p,m+1}^{(t)}$ *then*

$$\mathscr{E}_{app}(q,m,t) \triangleq \|\psi_{p,m}^t - \psi_{q,m}^t\|_2^2 \leq C \cdot D_{KL}(p\|q) \triangleq U_{app}(q,m,t) \tag{4.5}$$

where $C = 32\jmath^2\ell^2 \ln 2/\delta_m^2$ *and* $D_{KL}(\cdot\|\cdot)$ *denotes the Kullback-Leibler divergence.*

Proof. From [Zwald and Blanchard, 2005], we have that:

$$\|\psi_{p,m}^t - \psi_{q,m}^t\|_2^2 \leq \frac{16}{\delta_m^2}\|A_p^t - A_q^t\|^2 \tag{4.6}$$

where

$$\begin{aligned}
\|A_p^t - A_q^t\|^2 &= \sup_{\|f\|\le 1} \|A_p^t[f(x)] - A_q^t[f(x)]\|_2^2 \\
&= \sup_{\|f\|\le 1} \left\| \int a_t(x,y)f(y)p(y)dy - \int a_t(x,y)f(y)q(y)dy \right\|_2^2 \\
&= \sup_{\|f\|\le 1} \left\| \int a_t(x,y)f(y)[p(y)-q(y)]dy \right\|_2^2 \\
&= \sup_{\|f\|\le 1} \int \left(\int a_t(x,y)f(y)[p(y)-q(y)]dy \right)^2 p(x)dx \\
&\le \sup_{\|f\|\le 1} \int \left(\int |a_t(x,y)||f(y)||p(y)-q(y)|dy \right)^2 p(x)dx \\
&= \sup_{\|f\|\le 1} \int \left(\int |p(y')-q(y')|dy' \times \int |a_t(x,y)||f(y)| \frac{|p(y)-q(y)|}{\int |p(y')-q(y')|dy'} dy \right)^2 p(x)dx \\
&\le \sup_{\|f\|\le 1} \int \left((\int |p(y')-q(y')|dy')^2 \times \int a_t^2(x,y)f^2(y) \frac{|p(y)-q(y)|}{\int |p(y')-q(y')|dy'} dy \right) p(x)dx \\
&\le \int \left((\int |p(y')-q(y')|dy')^2 \times \int \jmath^2 \ell^2 \frac{|p(y)-q(y)|}{\int |p(y')-q(y')|dy'} dy \right) p(x)dx \\
&= \int \jmath^2 \ell^2 \|p-q\|_1^2 p(x)dx = \jmath^2 \ell^2 \|p-q\|_1^2
\end{aligned} \tag{4.7}$$

On the other hand we have the following inequality [Cover and Thomas, 2000]:

$$\|p-q\|_1 \le \sqrt{2\ln 2 \cdot D_{KL}(p\|q)} \tag{4.8}$$

Therefore, we have:

$$\|\psi_{p,m}^t - \psi_{q,m}^t\|_2^2 \le \frac{16}{\delta_m^2}\|A_p^t - A_q^t\|^2 \le \frac{16}{\delta_m^2}\jmath^2\ell^2\|p-q\|_1^2 \le \frac{32\ln 2}{\delta_m^2}\jmath^2\ell^2 \cdot D_{KL}(p\|q) \tag{4.9}$$

□

Theorem 2 translates the distance between the true and the approximated eigenfunction to the distance between the true underlying distribution and its factorized approximation.

Theorem 3. ***Upper bound on the estimation error:*** *Let q factorizes according to $\mathscr{T}_{\boldsymbol{k}}$ and $\sup_{f:\|f\|_2\le 1}\|f\|_\infty = \ell < \infty$. Define $S_m = \{(j_1,\ldots,j_k) \mid \forall i \in [1..k] : 1 \le j_i \le m \text{ and } \prod_{i=1}^k j_i \le m\}$ and the multiplicative eigengap $\mu_{i,j}^{(t)} = \min_{2\le l\le j}\log(\lambda_{i,l-1}^{(t)}/\lambda_{i,l}^{(t)})$ then we have:*

$$
\begin{aligned}
&\mathscr{E}_{est}(q,m,t) \triangleq \|\psi_{q,m}^t - \hat{\psi}_{q,m}^t\|_2^2 \\
&\le \max_{(j_1,\ldots,j_k)\in S_m} \ell^{2(k-1)} \sum_{i=1}^k 2^i \|\psi_{i,j_i}^t - \hat{\psi}_{i,j_i}^t\|_2^2 \\
&= O_P\Big(\max_{(j_1,\ldots,j_k)\in S_m} \ell^{2(k-1)} \sum_{\substack{i=1\\ j_i\ne 1}}^k \frac{2^i t\sqrt{d_i}}{\mu_{i,j_i}^{(t)}} \Big[\frac{\log N}{N}\Big]^{2/(d_i+8)} \Big) \triangleq U_{est}(q,m,t) \qquad (4.10)
\end{aligned}
$$

where n is the sample size and d_i is the dimensionality of the subspace T_i. Furthermore, the sum in the last equality is over at most $\min(k, \lceil \lg m \rceil)$ *sub-spaces.*

Proof. Let $\mathscr{T}_{\boldsymbol{k}} = \{T_1, T_2, \ldots, T_k\}$ be a partition of the variables in V into k subspaces. Also, let $\boldsymbol{\Lambda_i^t} = \{\lambda_{i,m}^{(t)} \mid 1 \le m \le \infty\}$ be the set of eigenvalues of the marginal diffusion operator $A_{p_i}^t$ on subspace T_i for all $1 \le i \le k$. Assume the members of $\boldsymbol{\Lambda_i^t}$ are sorted in the decreasing order with the first (the largest) eigenvalue $\lambda_{i,1}^{(t)} = 1$ associated with the constant eigenfunction $\psi_{i,1}^t = 1$.

Using Lemma 1, the eigenvalues (and their associated eigenfunctions) of A_q^t are constructed by picking one eigenvalue from each set $\boldsymbol{\Lambda_i^t}$ for all $1 \le i \le k$ and multiply them together; that is, the $\lambda_{q,m}^{(t)} = \prod_{i=1}^k \lambda_{i,j_i}^{(t)}$ is the m-th largest eigenvalue of A_q^t. For each m, we can find the index tuple $(j_1,\ldots,j_k)$ indicating which eigenvalue is exactly picked in each subspace to construct the m-th largest eigenvalue of A_q^t. If we know the index tuple $(j_1,\ldots,j_k)$ for the m-th eigenfunction, we can find the upper bound on the estimation error as follows:

$$
\begin{aligned}
\|\psi_{q,m}^t - \hat{\psi}_{q,m}^t\|_2^2 &= \|\prod_{i=1}^k \psi_{i,j_i}^t - \prod_{i=1}^k \hat{\psi}_{i,j_i}^t\|_2^2 \\
&\le 2\Big\|\prod_{i=1}^k \psi_{i,j_i}^t - \psi_{1,j_1}^t \prod_{i=2}^k \hat{\psi}_{i,j_i}^t\Big\|_2^2 + 2\Big\|\psi_{1,j_1}^t \prod_{i=2}^k \hat{\psi}_{i,j_i}^t - \prod_{i=1}^k \hat{\psi}_{i,j_i}^t\Big\|_2^2 \\
&= 2\Big\|\psi_{1,j_1}^t \big(\prod_{i=2}^k \psi_{i,j_i}^t - \prod_{i=2}^k \hat{\psi}_{i,j_i}^t\big)\Big\|_2^2 + 2\Big\|(\psi_{1,j_1}^t - \hat{\psi}_{1,j_1}^t)\prod_{i=2}^k \hat{\psi}_{i,j_i}^t\Big\|_2^2 \\
&\le 2\ell^2 \cdot \|\prod_{i=2}^k \psi_{i,j_i}^t - \prod_{i=2}^k \hat{\psi}_{i,j_i}^t\|_2^2 + 2\ell^{2(k-1)} \cdot \|\psi_{1,j_1}^t - \hat{\psi}_{1,j_1}^t\|_2^2 \qquad (4.11)
\end{aligned}
$$

Using the above derivation recursively, we get:

$$\|\psi^t_{q,m} - \hat{\psi}^t_{q,m}\|_2^2 \leq \ell^{2(k-1)} \sum_{i=1}^{k} 2^i \|\psi^t_{i,j_i} - \hat{\psi}^t_{i,j_i}\|_2^2 = \ell^{2(k-1)} \sum_{\substack{i=1\\ j_i \neq 1}}^{k} 2^i \|\psi^t_{i,j_i} - \hat{\psi}^t_{i,j_i}\|_2^2 \tag{4.12}$$

The equality in Eq. (4.12) comes from the fact that for $j_i = 1$, $\psi^t_{i,j_i} = \hat{\psi}^t_{i,j_i} = 1$. Since we do not know the true eigenvalues in each subspace, we cannot identify the index tuple $(j_1, \dots, j_k)$ for a given index m. As a result the above bound is replaced by the worst case scenario across all possible index tuples, that is:

$$\|\psi^t_{q,m} - \hat{\psi}^t_{q,m}\|_2^2 \leq \max_{(j_1,\dots,j_k)} \ell^{2(k-1)} \sum_{\substack{i=1\\ j_i \neq 1}}^{k} 2^i \|\psi^t_{i,j_i} - \hat{\psi}^t_{i,j_i}\|_2^2$$

However, because $\psi^t_{q,m}$ is associated with the m-th largest eigenvalue of A^t_q (i.e. $\lambda^{(t)}_{i,1}$), not all combinations for the index tuple should be considered in taking the maximum. More precisely, if we replace any of the indices j_i in the index tuple $(j_1, \dots, j_k)$ with a smaller index $j'_i < j_i$, the resulted multiplicative eigenvalue will become larger; this is because of the fact that smaller indices in each set $\boldsymbol{\Lambda^t_i}$ correspond to larger eigenvalues. The total number of such replacements for the index tuple $(j_1, \dots, j_k)$ is $\prod_{i=1}^k j_i$. This means that if the index tuple for the m-th largest eigenvalue of A^t_q is $(j_1, \dots, j_k)$, m must be greater than $\prod_{i=1}^k j_i$. In other words, the valid index tuples for the m-th largest eigenvalue must satisfy $\prod_{i=1}^k j_i < m$. If S_m denotes the set of such tuples, we can improve the bound as:

$$\|\psi^t_{q,m} - \hat{\psi}^t_{q,m}\|_2^2 \leq \max_{(j_1,\dots,j_k)\in S_m} \ell^{2(k-1)} \sum_{\substack{i=1\\ j_i \neq 1}}^{k} 2^i \|\psi^t_{i,j_i} - \hat{\psi}^t_{i,j_i}\|_2^2$$

Now, using Eq. (4.2), we get:

$$\|\psi^t_{q,m} - \hat{\psi}^t_{q,m}\|_2^2 = O_P\Big(\max_{(j_1,\dots,j_k)\in S_m} \ell^{2(k-1)} \sum_{\substack{i=1\\ j_i \neq 1}}^{k} \frac{2^i t \sqrt{d_i}}{\mu^{(t)}_{i,j_i}} \Big[\frac{\log N}{N}\Big]^{2/(d_i+8)}\Big)$$

Moreover, using Lemma 2, there at most $\min(k, \lceil \lg m \rceil)$ non-constant eigenvectors contributing in constructing $\psi^t_{q,m}$ which means the sum in the above bound has at most $\min(k, \lceil \lg m \rceil)$ terms. □

Roughly speaking, the above result states that in estimating the m-th eigenfunction of the factorized operator A_q^t, the error is bounded by sum of the estimation errors in *at most* $\lceil \lg m \rceil$ subspaces each of which has a reduced dimensionality from d to d_i.

The main implication of the above theorems can be summarized as follows: suppose the underlying distribution p is equal or close to the factorized distribution q. If the procedure in Lemma 1 is used to estimate the principal eigenfunctions of A_p^t, the upper bound on the approximation error of these eigenfunctions will be small because p and q are close (Theorem 2). Moreover, the upper bound on the estimation error will involve only a few independent subspaces induced by q each of which has a reduced dimensionality and therefore has an exponentially faster convergence rates (Theorem 3). As a result, we get smaller total error upper bound $U_{total}(q,m,t) = U_{app}(q,m,t) + U_{est}(q,m,t)$ compared to the error bound for the standard diffusion map (Note that using the trivial partition $\mathscr{T}_{\mathbf{1}} = \{V\}$ is equivalent to the standard diffusion map).

4.2.4.3 Finding The Best Partition So far, we have assumed that for the given problem a good partition of variables is known. This is a reasonable assumption in those problems where the (unconditional) independencies among the variables are known in advance. For instance, in object recognition problem, one may consider the edge and the texture features of the input images to be almost independent. However, in many other problems, the independencies and week dependencies among variables (and therefore the optimal partitioning) are not a priori known and need to be discovered from the data. To this end, we need an optimization criterion to evaluate the goodness of different partitions w.r.t. the task in hand. In this section, we use the *estimated* total error for estimation of the major eigenfunctions to find a nearly optimal partition of the variables for factorized diffusion mapping. More formally, given an unlabeled dataset $\mathscr{D}$, we want to find the partition that minimizes $\mathscr{E}_{total}(q,m,t)$. However, we face the following two big challenges to solve this optimization problem. First, since we do not know the true eigenfunctions, we cannot directly compute the total error and need to estimate it. One approach to estimation of $\mathscr{E}_{total}$ is to use the upper bound U_{total} as a proxy for $\mathscr{E}_{total}$. However, the problem with this solution is we need to estimate the constants for error bounds in Theorems 2 and 3 as well as the true multi-

Algorithm 9: Greedy Partitioning

```
1:  input: dataset $\mathscr{D}$ with features $V$
2:  output: the optimal partitioning $\mathscr{T}^*$
3:  $k \leftarrow 1, \mathscr{T}_{\boldsymbol{1}} \leftarrow V$
4:  loop
5:    for all $T_i \in \mathscr{T}_{\boldsymbol{k}}$ do
6:      $\{\tilde{T}_{i1}, T_i \backslash \tilde{T}_{i1}\} \leftarrow Qu(\mathscr{D}, T_i)$
7:      $\Delta_i \leftarrow \Delta_{total}(\{\tilde{T}_{i1}, T_i \backslash \tilde{T}_{i1}\} \mid \mathscr{T}_{\boldsymbol{k}})$
8:    end for
9:    $j \leftarrow \mathrm{argmax}_{1 \leq i \leq k} \Delta_i$
10:   if $\Delta_j > 0$ then
11:     $\mathscr{T}_{\boldsymbol{k}} \leftarrow \mathscr{T}_{\boldsymbol{k}} \backslash \{T_j\} \cup \{T_{j1}, T_{j2}\}$
12:     $k \leftarrow k + 1$
13:   else
14:     $\mathscr{T}^* \leftarrow \mathscr{T}_{\boldsymbol{k}}$; stop
15:   end if

16: end loop
```

plicative eigengaps which is not easy in general for real problems; let alone the fact that these bounds are not tight anyway. To get around these problems, in our framework, we use a bootstrapping algorithm to estimate $\mathscr{E}_{total}(q, m, t)$. More precisely, from the given sample $\mathscr{D}$ of size N, we draw b bootstrap subsamples $\mathscr{D}_1, \ldots, \mathscr{D}_b$ of size $N/2$ each. Then the total error for the given partition $\mathscr{T}_{\boldsymbol{k}}$ is estimated as:

$$\hat{\mathscr{E}}_{total}(q, m, t) = \frac{1}{b} \sum_{i=1}^{b} \| \hat{u}^t_{p,m,\mathscr{D}} - \hat{u}^t_{q,m,\mathscr{D}_i} \|_2^2 \tag{4.13}$$

Here, $\hat{u}^t_{p,m,\mathscr{D}}$ is the estimated eigenvector over the sample $\mathscr{D}$ using no partitioning whereas $\hat{u}^t_{q,m,\mathscr{D}_i}$ denotes the estimated multiplicative eigenvector over the bootstrap subsample $\mathscr{D}_i$ if the partitioning $\mathscr{T}_{\boldsymbol{k}}$ is applied.

Second, even after estimating the total error, we still need to find the optimal partition that minimizes the estimated error which is an NP-hard problem. To address this issue, we develop a greedy algorithm that recursively splits the variable set V into disjoint subsets and and stops when $\hat{\mathscr{E}}_{total}(q, m, t)$ cannot be decreased anymore. Let us start with the following definitions:

Definition 1. *Denoted by $\mathscr{T}'_{\boldsymbol{k+1}} \succ_i \mathscr{T}_{\boldsymbol{k}}$, $\mathscr{T}'_{\boldsymbol{k+1}}$ is defined to be an immediate refinement of $\mathscr{T}_{\boldsymbol{k}}$ on the subset $T_i \in \mathscr{T}_{\boldsymbol{k}}$ if*

$$\mathscr{T}'_{\boldsymbol{k+1}} = \mathscr{T}_{\boldsymbol{k}} \backslash \{T_i\} \cup \{T_{i1}, T_{i2}\}$$

where $T_{i1}, T_{i2} \neq \phi$, $T_{i1} \cup T_{i2} = T_i$ *and* $T_{i1} \cap T_{i2} = \phi$.

Definition 2. *Suppose* $\mathscr{T}'_{\boldsymbol{k+1}} \succ_i \mathscr{T}_{\boldsymbol{k}}$ *with the split* $\{T_{i1}, T_i \backslash T_{i1}\}$ *of* T_i*. The error gain of the split* $\{T_{i1}, T_i \backslash T_{i1}\}$ *applied on* $\mathscr{T}_{\boldsymbol{k}}$ *is defined as:*

$$\Delta_{total}(\{T_{i1}, T_i \backslash T_{i1}\} \mid \mathscr{T}_{\boldsymbol{k}}) \triangleq \hat{\mathscr{E}}_{total}(q_{\mathscr{T}_{\boldsymbol{k}}}, m, t) - \hat{\mathscr{E}}_{total}(q_{\mathscr{T}'_{\boldsymbol{k+1}}}, m, t) \tag{4.14}$$

Furthermore, the optimal error gain of splitting T_i *in* $\mathscr{T}_{\boldsymbol{k}}$ *is defined to be:*

$$\Delta^*_{total}(T_i \mid \mathscr{T}_{\boldsymbol{k}}) \triangleq \Delta_{total}(\{T^*_{i1}, T_i \backslash T^*_{i1}\} \mid \mathscr{T}_{\boldsymbol{k}}) \tag{4.15}$$

where

$$T^*_{i1} = \arg\max_{T_{i1} \subset T_i} \Delta_{total}(\{T_{i1}, T_i \backslash T_{i1}\} \mid \mathscr{T}_{\boldsymbol{k}}) \tag{4.16}$$

For now, suppose we can efficiently compute $\Delta^*_{total}(T_i \mid \mathscr{T}_{\boldsymbol{k}})$ for all $T_i \in \mathscr{T}_{\boldsymbol{k}}$. Then given the current partition $\mathscr{T}_{\boldsymbol{k}}$, the greedy algorithm picks the subset $T_i \in \mathscr{T}_{\boldsymbol{k}}$ with the maximum gain $\Delta^*_{total}(T_i \mid \mathscr{T}_{\boldsymbol{k}})$ to be split into $\{T^*_{i1}, T_i \backslash T^*_{i1}\}$ and generates $\mathscr{T}'_{\boldsymbol{k+1}}$ for the next iteration. The algorithm stops when the gains for all subsets in the current partition are negative. Of course, this algorithm is based on the assumption that $\Delta^*_{total}(T_i \mid \mathscr{T}_{\boldsymbol{k}})$ is efficiently computable which is not the case because of the intractable set maximization problem in Eq. (4.16). To address this problem, first we define the gain for the approximation error *upper bound* obtained from splitting T_i into $\{T_{i1}, T_i \backslash T_{i1}\}$ as:

$$\begin{aligned}\Delta^U_{app}(\{T_{i1}, T_i \backslash T_{i1}\} \mid \mathscr{T}_{\boldsymbol{k}}) &\triangleq U_{app}(q_{\mathscr{T}_{\boldsymbol{k}}}, m, t) - U_{app}(q_{\mathscr{T}'_{\boldsymbol{k+1}}}, m, t) \\ &= -C \cdot MI(T_{i1}, T_i \backslash T_{i1}) \end{aligned} \tag{4.17}$$

where $MI(X,Y)$ denotes the *mutual information* between the random vectors X and Y and C is the constant defined in Theorem 2. The equality in Eq. (4.17) can be obtained from the result of Theorem 2 using some basic algebra. We propose to use Δ^U_{app} instead of Δ_{total} in Eq. (4.16) to find the split $\{\tilde{T}_{i1}, T_i \backslash \tilde{T}_{i1}\}$ as an approximation to $\{T^*_{i1}, T_i \backslash T^*_{i1}\}$; that is,

$$\begin{aligned}\tilde{T}_{i1} &= \arg\max_{T_{i1} \subset T_i} \Delta^U_{app}(\{T_{i1}, T_i \backslash T_{i1}\} \mid \mathscr{T}_{\boldsymbol{k}}) \\ &= \arg\min_{T_{i1} \subset T_i} MI(T_{i1}, T_i \backslash T_{i1}) \end{aligned} \tag{4.18}$$

Using this heuristic, finding the best splitting inside each subset reduces to finding the most independent bi-split of the subset. The benefit of using this heuristic is that the optimization function in Eq. (4.18) is a symmetric submodular function which can be minimized using the Queyranne algorithm in $O(|T_i|^3)$ [Narasimhan and Bilmes, 2004]. The disadvantage is, at the level of finding the best split inside each subset, we do not exactly maximize the estimated total error anymore. However at one level higher, when the algorithm decides which subset in the current partition should be split, it looks at the estimated total error, which is the original objective function we aim to minimize.

Once the split $\{\tilde{T}_{i1}, T_i \backslash \tilde{T}_{i1}\}$ is found, we can plug it in Eq. (4.15) to compute $\tilde{\Delta}_{total}(T_i \mid \mathscr{T}_{\boldsymbol{k}})$ as an approximation to $\Delta^*_{total}(T_i \mid \mathscr{T}_{\boldsymbol{k}})$ for all $T_i \in \mathscr{T}_{\boldsymbol{k}}$. Algorithm 9 above summarizes the greedy partitioning algorithm. Note that $Qu(\mathscr{D}, T_i)$ in Algorithm 9 denotes the Queyranne algorithm which finds the splitting of T_i into $\{\tilde{T}_{i1}, T_i \backslash \tilde{T}_{i1}\}$ that minimizes Δ^U_{app}.

There are a couple of points regarding the proposed algorithm in this section to be clarified.

(1) Although the estimation error $\mathscr{E}_{est}$ is not used in finding the best splitting of each T_i, it is implicitly included in Δ^*_{total} and therefore is used to decide which T_i should be split in the next iteration.

(2) $\Delta^U_{app}(T_i \mid \mathscr{T}_{\boldsymbol{k}})$ only depends on the subsets T_{i1} and $T_i \backslash T_{i1}$ inside T_i and does not change if we refine other T_j's. However, this isn't true for $\Delta^*_{total}(T_i \mid \mathscr{T}_{\boldsymbol{k}})$; that is, $\Delta^*_{total}(T_i \mid \mathscr{T}_{\boldsymbol{k}})$ depends on the whole partition $\mathscr{T}_{\boldsymbol{k}}$ and will change if any members of $\mathscr{T}_{\boldsymbol{k}}$ is split. Because of this, we cannot apply the splitting in all T_i's at the same time; in fact, any new split will change $\Delta^*_{total}(T_i \mid \mathscr{T}_{\boldsymbol{k}})$ for all T_i's.

(3) In practice we need a robust method to estimate the mutual information between different subsets of continuous random variables from the sample $\mathscr{D}$. One candidate is the Maximum Likelihood Density Ratio method [Suzuki et al., 2008] which roughly has the convergence rate of $O_p(N^{-\frac{1}{2}})$ [Suzuki et al., 2009].

(4) Depending on the size of problem and the method used for estimating mutual information, the optimization in Eq. (4.18) might be still too slow. To alleviate this problem in practice, one can substitute line 6 in Algorithm 9 with any method that finds nearly independent partitions of variables (e.g. partitioning of the covariance matrix).

(5) One needs to decide for which eigenfunction (i.e. which m) $\hat{\mathscr{E}}_{total}(q,m,t)$ should be minimized in the greedy partitioning algorithm. In our experiments, we have used a weighted average error over the first four principal eigenfunctions.

4.2.5 Experimental Results

4.2.5.1 Synthetic Data In the first experiment, our goal is to cluster the synthetic 3D dataset in Figure 19 (two balls surrounded by two rings) using spectral clustering. In particular, we applied K-means in the embedded spaces induced by the factorized diffusion map and the standard diffusion map. The dimension of the embedded spaces for both mappings is 3 using the first three non-trivial eigenvectors of the corresponding operators. Assuming the independence $X,Y \perp Z$ is known in advance, we passed the partition $\mathscr{T}_{\mathbf{2}} = \{\{X,Y\},\{Z\}\}$ to the factorized diffusion mapping algorithm (induced by Lemma 1). To assess the performance of mappings, we measure the divergence of the clustering result in each case from the true cluster labels. To do so, we have used the normalized *variation of information* which is a distance metric between two clusterings [Meila, 2003]. This metric measures the conditional entropy of cluster labels given the true labels and vice versa (the smaller this metric is, the closer two clusterings are). Figure 16(A) shows the variation of information for the two methods with the true cluster labels as the sample size changes. We also show the performance of standard K-means without any spectral embedding (the black curve). The curves are averages over 20 repetitions with the error bars showing the 95% confidence intervals.

As the results show, for small sample sizes there is no difference between the performance of the two spectral methods. However, as we increase the sample size, our method starts to outperform the standard diffusion map leading to significantly smaller variation of information with the true cluster labels. As we continue increasing the sample size, the difference between the two methods starts decreasing with both methods eventually reaching the perfect clustering given the sample size is sufficiently large (700 for our method). According to these observations, we conclude that the extra knowledge regarding the underlying distribution of data (i.e. the independence relation) is particularly useful for mid-range

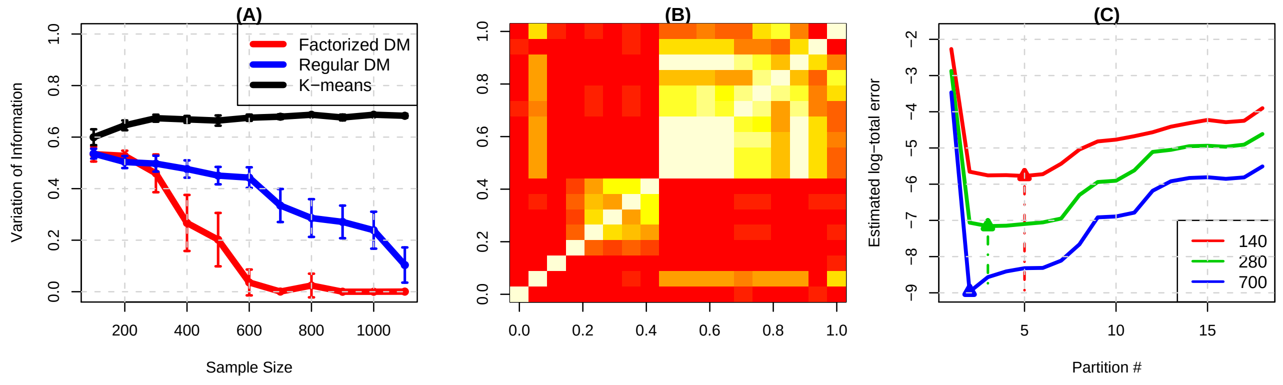

Figure 16: (A) Clustering results on synthetic data (B) Covariance matrix (C) The bootstrapping error for different partitions on the image data of features in the image data set.

sample sizes and can significantly improve the results of spectral clustering. However, for very small or very large sample sizes, this extra piece of information may not make a significant difference. Also, the standard K-means performed very poorly compared to the spectral methods.

4.2.5.2 Image Data In the second experiment, we have applied our framework on the image segmentation dataset [1]. This dataset consists of 2310 instances. Each instance was drawn randomly from a database of seven outdoor categories. The image, a 3×3 region, was

[1]http://archive.ics.uci.edu/ml/datasets/Image+Segmentation

n	**k***	**Baseline α**	**Factorized α**
140	5	0.727 ± 0.007	0.755 ± 0.007
280	3	0.707 ± 0.006	0.748 ± 0.007
700	2	0.704 ± 0.005	0.764 ± 0.006

Table 26: Results of Greedy Partitioning on image data set for different sample sizes.

hand-segmented to create a classification for each region. The seven classes are brickface, sky, foliage, cement, window, path, and grass. Each of the seven categories is represented by 330 instances. The extracted features are 19 continuous attributes that describe the position of the extracted image, line densities, edges, and color values. We have treated this classification problem as a clustering task with each class regarded as one cluster. The main reason for choosing this dataset is the features conceptually seem to be divided into nearly independent subsets (e.g. the position vs. the edge features). Figure 16(B) also shows the empirical covariance matrix of this dataset with nearly blocked structure which again indicates the existence of independent feature subsets. However, there might still exist some non-linear dependencies among features and therefore we cannot completely trust on the block structure suggested by the covariance matrix as the true partitioning. This observation motivates utilizing the proposed Greedy Partitioning algorithm to automatically find the best partition for factorized diffusion mapping of the data.

Figure 16(C) shows the optimization paths of the Greedy Partitioning algorithm for different sample sizes (the plots are averaged over 10 runs). The x-axis shows the number subsets in the partitioning on the variables while the y-axis is the total error estimated using bootstrapping in the log-scale (with the minimums marked on the plots). As the figure shows, all of the plots have the same general trend: namely as we start partitioning the features, there is a significant drop on the estimated total error until the total error reaches a minimum. We attribute this behavior to the decrease in the estimation error while introducing very small approximation error. However, if we continue refining the partitioning, the increase in the approximation error will dominate the decrease in the estimation error and therefore the total error starts increasing again. It is also apparent from the plots that as the sample size increases the estimated total error decreases for a fixed number of partitions. Finally, note that the position of minimum is shifted to the left (i.e. toward smaller partition numbers) as we increase the sample size. This observation, in fact, shows that for smaller sample sizes, the algorithm automatically regularizes more by imposing more independence assumptions (i.e. more refined partitioning) in order to get more accurate estimation of eigenfunctions.

Having found the optimal partitioning and used it for the factorized diffusion mapping, we can feed the resulted mapping to K-means to find the clusters. To evaluate the result of clustering given the true cluster labels, one option is to use the variation of information score as before. However, we observed that the results of K-means were more sensitive to initial centers in this real-world problem. To alleviate this issue, we develop a new evaluation metric called *separation* which assesses how separated the true cluster are in the embedded space, independent of the initial cluster positions for K-means. To compute this metric: given the new coordinates of data in the embedded space $\{z^{(1)}, \ldots, z^{(N)}\}$ and the true cluster labels, we compute the center μ_i for cluster C_i in the embedded space. Each C_i has N_i data points; we define w_i to be the number of data points among the N_i closest points to μ_i which actually belong to the cluster C_i (using the true labels). The separation is computed as $\alpha = \sum_i w_i / N$ which is a number in $[0, 1]$. For $\alpha = 1$, we have the perfect separation meaning that given a good set of initial points, K-means can completely separate the clusters based on the true labels. In fact, this metric is equivalent to *clustering purity metric* when K-means generates the ideal clustering by finding clusters centered at μ_i's. Table 4.2.5.1 summarizes the optimal number of partitions k^* found for each sample size as well as the separation α (and its 95% CI) for both standard and factorized diffusion maps. All the results are the average over 10 runs. As the results show, using factorized diffusion embedding, the separation of clusters in the embedded space is significantly improved.

4.2.5.3 SNP Data In the third experiment, we have applied the factorized diffusion mapping on a SNP dataset. A SNP represents a certain location on the human DNA which is an unordered pair of *alleles* from the alphabet $\{A, C, G, T\}$, for example AA, AC, GT, etc. The genetic code for an individual is represented by a set of SNPs locating at different positions on the human genome. Subsequently, in a SNP dataset, each record (or feature vector) represents the genetic code for an individual where the features are the SNPs. Across different individuals, each SNP is restricted to take values from only two distinct alleles; for example, SNP_1 may only take values from $\{A, C\}$ which means that the only legitimate values for this SNP are AA, AC and CC. One of these two alleles is the *major* allele meaning that it appears with much higher frequency across the population for that specific allele, whereas the *allele* is the result of genetic mutation and happens with much lower frequency. This means that if in our previous example for SNP_1, A is the major allele and C is the minor one, the pair AA happens with the highest frequency, the pair AC is the result of one genetic mutation and happens with lower frequency than AA across the population. The pair CC in this example is the result of two genetic mutations and is even rarer than AC. In our SNP dataset, the most frequent pair (AA in our example) is coded by 0 while the rarest pair (CC in our example) is coded by 2 and the one in between (AC in our example) is coded by 1. In other words, the features (i.e. the SNPs) in our dataset take values from the set $\{0, 1, 2\}$. Although the features are discrete, it is easy to see from the description above that the Euclidean distance between two indviduals over the same set of SNPs is meaningful; that is the order of feature values makes sense.

The dataset we worked on in this section consists of 1,411 individuals each of which are the SNP values over 88,709. That is, we have 1,411 datapoints in a space with the dimensionality of 88,709. Furthermore, each individual is labeled by a binary label showing whether he/she has the Alzheimer's disease. In this dataset, we have 860 healthy individuals vs. 551 individuals with Alzheimer.

Like many other biomedical datasets, this dataset suffers from high dimensionality compared to its sample size. Therefore, dimensionality reduction seems inevitable before any further Machine Learning task can be done. Due to our experimental goals, here we focus on the Diffusion Maps to carry out dimensionality reduction. Due to the shortage of examples compared to the dimensionality, the empirical eigenfunctions of the diffusion operator will be far from the true eigenfunctions as discussed before in this chapter. This means that our proposed factorized diffusion mapping framework can be a good solution in this case. To this end, first we need to partition the SNPs into (almost) independent subsets. One option is to use the algorithm proposed in Section 4.2.4.3. However, that algorithm is computationally expensive in terms of the number of dimensions such that having 88,709 dimensions makes it infeasible to apply this algorithm. Instead, we rely on the domain knowledge to partition the SNPs. In particular, we know each SNP belongs to a *gene* which covers a certain region on the DNA; that is, a gene is consecutive sequence of SNPs on the DNA. Therefore, genes provide a partitioning of SNPs into disjoint subsets. Now, the question is whether these subsets are independent of each other. From the domain knowledge, it is commonly believed that the SNPs belonging to the same gene exhibit much stronger dependence among each other than to the SNPs on other genes. As a result, we use this knowledge to partition the SNPs in our dataset into almost independent subsets based on the genes. In this datasets, we have 11,509 genes which generate 11,509 SNP subsets (or subspaces).

We have first computed the eigenvalues and the eigenfunctions of the empirical diffusion operator in each subspace independently and then found the Cartesian product of these values and functions to form the eigenvalues and the eigenfunctions of the entire space as described in Lemma 1. The resulted eigenfunctions are sorted according to the decreasing order of the resulted eigenvalues. We then used the first C eigenfunctions to embed the feature vectors into a C-dimensional space. As a baseline approach, we have also used the first C eigenfunctions of the regular diffusion map to embed the data.

In order to evaluate the regular and the factorized diffusion maps on the SNP data, we have run the K-means clustering algorithm in the embedded spaces generated by the two methods to cluster the datapoints into two groups. Then we computed the distance between the clustering result and the true labels in terms of the variation of information as described before in this section. Since, K-means is sensitive to initial cluster centers, we have repeated the clustering 5 times with different random initial centers. The reported results are the average variation of information. Figure 17 shows the average variation of information along with its 95% confidence interval for two methods as we change C (the number of dimensions in the embedded space) from 1 to 1,000 (Note that the X-axis is in the log-scale). As the plot shows, in all cases, using the factorized diffusion mapping according to the gene partition decreases the average variation of information; that is, by using the factorized eigenfunctions, we could decrease the distance between the true clustering (according to the true labels) and the result of K-means. Moreover, this difference is statistically significant in most cases. This results show that by using the factorized diffusion maps according to the gene partitioning of the SNPs, we could improve the result of clustering by decreasing the distance between the clustering results and the true labels.

An interesting question is how did the clustering result using the factorized diffusion maps get closer to the clustering induced by the true labels while we know that diffusion maps are inherently unsupervised techniques? To answer this question, we refer the reader to the analysis provided in the proof of Lemma 2. According to this analysis many top eigenfunctions of the factorized diffusion map correspond to one or only a few independent subspaces; that is, these eigenfunctions are the product of the second eigenfunction(s) with highest eigenvalue(s) from a (few) subspace(s) by the constant eigenfunctions corresponding to eigenvalue of 1 from other subspaces. Since we have a large number of (almost) independent subspaces in our problem (i.e. 11,509), we observed that many of the top eigenfunctions indeed correspond to only one subspace (or equivalently one gene). In other words, the factorized diffusion map reduces the dimensionality by effectively abstracting the feature vectors in the SNP space into lower-dimensional vectors in the gene space. This means that the feature values in the embedded space model the variation across the population at the gene level. On the other hand, some medical conditions and diseases like the Alzheimer's disease are known to have genetic risk factors which may not be well captured in the higher-dimensional SNP space with small set of examples. However, abstracing to the gene level can reduce the dimensionality while preserving the discriminant signal at the same time at the gene level and therefore improve the discrimination between the two classes of control and disease. It should be emphasized that the gene space is still high dimensional and far from perfect in terms of class discrimination. However, applying our proposed framework is still a significant improvement compared to the regular diffusion maps.

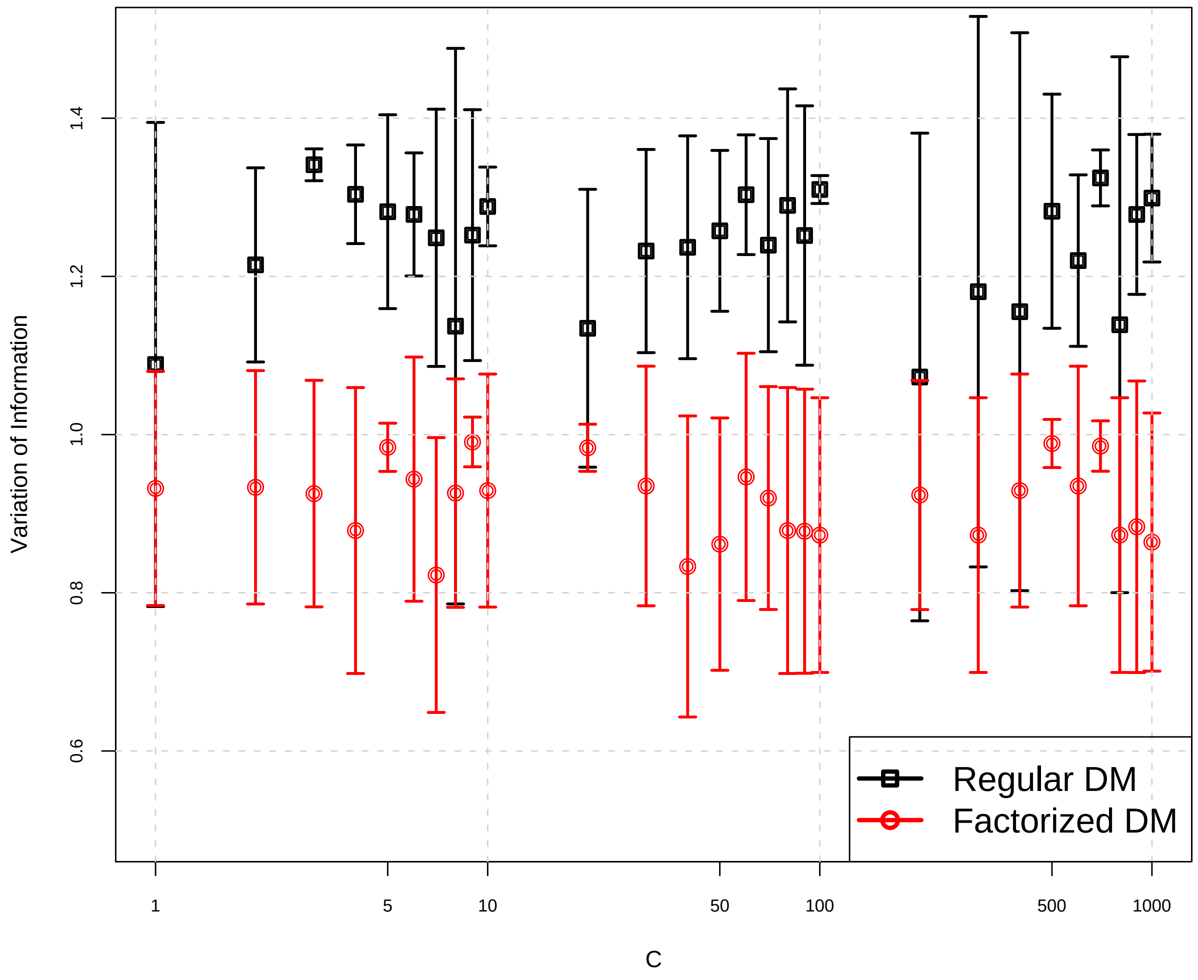

Figure 17: Speactral clustering of the SNP data: the average variation of information between the clustering resulted by the Regular & the Factorized Diffusion Mappings and the true labels vs. the dimensionality of the embedded space.

4.2.6 Discussion

In this section, we utilized the existence of independence structure in the underlying distribution of data to estimate diffusion maps. In particular, we studied the reduction on the estimation error of diffusion eigenfunctions resulting from a factorized distribution. We showed that if the underlying space is factorized into independent subspaces, the estimation error of the major eigenfunctions can be decomposed into errors in only a subset of these subspaces each of which has a much smaller dimensionality than the original space. Since in many real problems, the factorized distribution either does not exist or is not known in advance, we studied how much bias is introduced if we impose the factorized distribution assumption. To find the optimal trade-off between the approximation bias and the estimation error, we developed a greedy algorithm for finding a factorization that minimizes the estimated total error. The experimental results showed that the factorized approximation can significantly improve the results of spectral clustering on both the synthetic and real datasets.

The fundamental intuition underlying the framework proposed in this chapter is that the density estimation problem and the spectral analysis of data are closely related. Hence, the same structural assumptions that can help us to reduce the complexity of learning for density estimation purposes, can also help us for the empirical spectral analysis.

4.3 INFINITE DIMENSION: DEALING WITH FUNCTIONAL SPACES

4.3.1 Motivation

So far, we have considered the case of applying the graph-based methods for problems with high-dimensional input feature space. Although the feature vector for each datapoint can be high-dimensional, the assumption is it has the same finite length for all examples. This assumption holds in many real-world problems; however, there are some cases where the feature vectors for different examples have different lengths. In particular, we consider the case where the input dataset is composed of *functional objects* instead of feature vectors.

For example in [Buchman et al., 2009], the authors consider the problem of estimating the density of hurricane tracks where each datapoint is given as a sequence of geographical locations which represent the trajectory of a hurricane. The problem with this setting is two-fold: (a) different tracks might be represented by different number of geographical location depending on the distance that is traversed by the hurricane, and (b) even if two tracks have the same number of locations (features), it is not an easy task to match their features; as a result, many element-wise distance metrics (like the Euclidean distance) are not applicable. In fact, it is quite a difficult task to define a distance metric between the tracks in this problem. By taking a closer look at this problem, it becomes apparent that we are actually dealing with functions which are sampled at different number of points. Therefore, in theory, the input space is a functional space and infinite-dimensional. The main question is how we can represent these infinite-dimensional objects in a meaningful finite-dimensional feature space with a meaningful distance metric.

In this section, we propose a method for converting functional datapoints to fixed-length and meaningful feature vectors. In particular, we consider the problem of transforming the human brain fibers into a *low-dimensional* vector space. This problem is motivated by recent advances in Magnetic Resonance Imaging called *Diffusion Weighted Imaging* [Mori and van Zijl, 2002] that are capable of quickly and non-invasively imaging hundreds of thousands of over 150,000 km of myelinated nerve fibers present in the human brain. The *Connectome* (the set of all fibers in one brain) determines the information transmission between areas, and thus is important for obtaining a fundamental understanding of connectivity between regions of gray matter in the brain (Figure 18(A)). These fibers are not arbitrarily developed in the brain but mostly organized in a bundled structure such that in some cases the bundle is actually associated with a specific functionality of the brain (Figure 18(B)). Unfortunately, this structure is not fully known, and therefore, applications like fiber clustering, fiber density estimation and fiber visualization arise for the Connectome data. However, many of the standard methods for these applications do not handle functional data and/or large-scale datasets. This would motivate our methodology to transform functional input data into a low-dimensional vector representation such that the result becomes usable for standard Machine Learning algorithms. Automatic mining of the brain fibers also serves an important practical purpose of aiding in preserving critical brain function during neurosurgical tumor removal. In such case it is important to know how to avoid cutting the major cables connecting critical brain functions such as language motor control and visual input. Thus we would like to perform probabilistic data analysis such as density estimation and data visualization on this infinite-dimensional, large-scale dataset.

The transformation process presented in this section consists of two steps: in the first step, the functional objects are converted into high-dimensional but meaningful fixed-length feature vectors. In the second step, the high-dimensional feature vectors are transformed into a low-dimensional space that maintains the cluster (bundle) structure in the original fiber space. For the second step, we have used the Laplacian Eigenmaps, presented in Chapter 1, to reduce the dimensionality while preserving the cluster structure of the fiber space. However, since the number of datapoints (fibers) in our problem is 250,000, we are facing the Large N problem here. As a result, the application of the standard Laplacian Eigenmaps are not feasible. To address this issue, we have developed and applied a large-scale version of the Laplacian Eigenmaps, which is in fact a solution to the Large N problem.

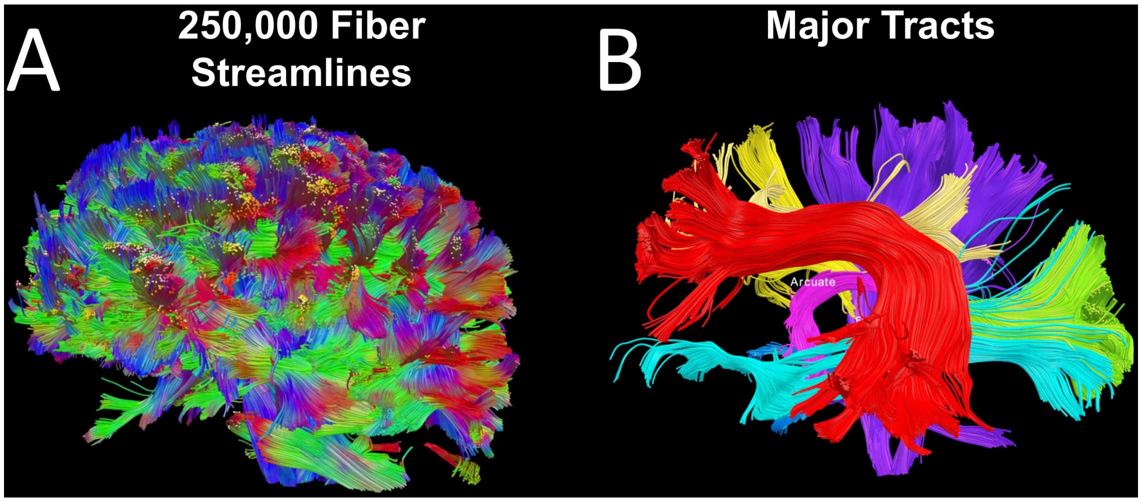

Figure 18: (A) The full Connectome (B) The bundles associated with brain functions.

4.3.2 Vector Representation

The Connectome dataset is a set of brain fibers each of which can be seen as a smooth 3D curve. Technically speaking, this would imply that the input space is infinite dimensional. However, in practice, each curve $\mathbf{C}^{(i)}$ is represented by a sequence of q_i 3D points $\{[x_k^{(i)} y_k^{(i)} z_k^{(i)}]^T \mid 1 \leq k \leq q_i\}$ in the dataset. In order to support comparisons of these objects and their analyses, we need to represent the curves compactly in a meaningful vector space with a well-defined distance measure.

A trivial approach of concatenating all 3D points that form the curve into a linear vector is not a promising solution since curves with different lengths will become vectors of different sizes, making hard to analyze. Instead, we build upon a technique originally introduced in *functional data analysis* (FDA) to process 1D functional datasets [Rossi et al., 2005]. In particular, we assume each coordinate of a curve is a parametric function of a (time) parameter t such that as t changes from 0 to 1, $[x^{(i)}(t), y^{(i)}(t), z^{(i)}(t)]^T$ generates $\mathbf{C}^{(i)}$. Then every point $[x_k^{(i)} y_k^{(i)} z_k^{(i)}]^T$ associated with one of the curves in the dataset can be thought of as a 3D point sampled from the original curve at time $t_k^{(i)} = (k-1)/(q_i - 1)$. Furthermore, we assume every curve can be represented in terms of basis m functions $\boldsymbol{\Psi} = \{\psi_1, \ldots, \psi_m\}$; that is, $x^{(i)}(t) = \sum_{j=1}^m \alpha_j^{(i,x)} \psi_j(t) = \left[\boldsymbol{\alpha}^{(i,x)}\right]^T \boldsymbol{\Psi}(t)$, where $\boldsymbol{\alpha}^{(i,x)} = \left[\alpha_1^{(i,x)}, \ldots, \alpha_m^{(i,x)}\right]^T$. [2] Using the sample 3D points associated with $\mathbf{C}^{(i)}$, one can estimate the coefficients $\boldsymbol{\alpha}^{(i)} = \{[\alpha_j^{(i,x)}, \alpha_j^{(i,y)}, \alpha_j^{(i,z)}]^T \mid 1 \leq j \leq m\}$ by minimizing the sum of squared errors (SSE):

$$\boldsymbol{\alpha}^{(i,x)} \leftarrow \arg\min_{\boldsymbol{\alpha}} \sum_{k=1}^{q_i} \left(x_k^{(i)} - \boldsymbol{\alpha}^T \boldsymbol{\Psi}(t_k^{(i)})\right)^2 \tag{4.19}$$

In FDA, the L2 distance between two functions f and g over Γ is defined as $\|f-g\|_2^2 = \int_\Gamma (f-g)^2 dx$. Similarly, we can define the distance between two curves $\mathbf{C}^{(i)}$ and $\mathbf{C}^{(j)}$ as the sum of L2 functional distances between their corresponding coordinates:

$$\|\mathbf{C}^{(i)} - \mathbf{C}^{(j)}\|_2^2 \triangleq \int_0^1 \left(x^{(i)}(t) - x^{(j)}(t)\right)^2 dt + \int_0^1 \left(y^{(i)}(t) - y^{(j)}(t)\right)^2 dt + \int_0^1 \left(z^{(i)}(t) - z^{(j)}(t)\right)^2 dt \tag{4.20}$$

Representing the curve coordinates in basis $\boldsymbol{\Psi}$, each term in (4.20) then can be written as:

$$\int_0^1 \left(x^{(i)}(t) - x^{(j)}(t)\right)^2 dt = \left[\boldsymbol{\alpha}^{(i,x)}\right]^T \boldsymbol{\Phi} \boldsymbol{\alpha}^{(i,x)} + \left[\boldsymbol{\alpha}^{(j,x)}\right]^T \boldsymbol{\Phi} \boldsymbol{\alpha}^{(j,x)} - 2\left[\boldsymbol{\alpha}^{(i,x)}\right]^T \boldsymbol{\Phi} \boldsymbol{\alpha}^{(j,x)} \tag{4.21}$$

[2]Notice that, for simplicity, we show the formulations only for the x coordinate; the same applies to y and z coordinates

where $\mathbf{\Phi} = \left[\int_0^1 \psi_i(t)\psi_j(t)dt\right]_{m\times m}$ is the *mass matrix* for the basis $\mathbf{\Psi}$ on $[0,1]$. If $\mathbf{\Psi}$ is chosen to be an orthonormal basis, then $\mathbf{\Phi}$ becomes the identity matrix. In this case, (4.20) reduces to:

$$\|\mathbf{C}^{(i)} - \mathbf{C}^{(j)}\|_2^2 = \|\boldsymbol{\alpha}^{(i,x)} - \boldsymbol{\alpha}^{(j,x)}\|_2^2 + \|\boldsymbol{\alpha}^{(i,y)} - \boldsymbol{\alpha}^{(j,y)}\|_2^2 + \|\boldsymbol{\alpha}^{(i,z)} - \boldsymbol{\alpha}^{(j,z)}\|_2^2 = \|\boldsymbol{\alpha}^{(i)} - \boldsymbol{\alpha}^{(j)}\|_2^2 \qquad (4.22)$$

That is, the L2 functional distance between $\mathbf{C}^{(i)}$ and $\mathbf{C}^{(j)}$ is equivalent to the Euclidean distance between their coefficient vectors $\boldsymbol{\alpha}^{(i)}$ and $\boldsymbol{\alpha}^{(j)}$. Therefore, the mapping $\mathbf{C}^{(i)} \mapsto \boldsymbol{\alpha}^{(i)}$ can be seen as a transformation from the functional space of curves to the vector space of their coefficient vectors which captures the L2 functional distance in the original space by the Euclidean distance in the new space. As a result, each curve $\mathbf{C}^{(i)}$ is represented by the vector $\boldsymbol{\alpha}^{(i)}$ of size $d = 3m$ regardless of its length (where m is the number of basis functions).

4.3.3 Large-Scale Laplacian Eigenmap

Our goal here is to embed the high dimensional functional space of fibers into a low dimensional vector space such that if two fibers belong to roughly the same bundle (cluster), their low dimensional representations are close in terms of Euclidean distance. We partly achieved this goal in the previous section by reducing the infinite dimensional function space into a high-dimensional vector space. For further dimensionality reduction (< 10) and preserving the clustering structure at the same time, we chose the *Laplacian Eigenmaps* [Belkin and Niyogi, 2002] framework described in Chapter 1. Just to review, by defining the similarity weight between two fibers $\mathbf{C}^{(i)}$ and $\mathbf{C}^{(j)}$ as $w_\sigma(i,j) = \exp(-\|\boldsymbol{\alpha}^{(i)} - \boldsymbol{\alpha}^{(j)}\|_2^2/2\sigma^2)$, one can compute Laplacian matrix, $\mathbf{L} = \mathbf{D} - \mathbf{W}$ where $\mathbf{W} = [w_\sigma(i,j)]_{N\times N}$ and $\mathbf{D}$ is a diagonal matrix with diagonal entries $d_{ii} = \sum_{j=1}^N w_\sigma(i,j)$. The eigenvalues of Laplacian matrix are all non-negative with the smallest one equal to 0 (i.e. $0 = \lambda_0 \le \lambda_1 \le \ldots \le \lambda_{N-1}$) [von Luxburg, 2007]. Defining $v_k(i)$ as the i-th element of the eigenvector of $\mathbf{L}$ associated with λ_k,

$$\phi_g(\mathbf{C}^{(i)}) \triangleq [\sqrt{g(\lambda_1)}v_1(i), \sqrt{g(\lambda_2)}v_2(i), \ldots, \sqrt{g(\lambda_K)}v_K(i)]^T \qquad (4.23)$$

is the K-dimensional Laplacian embedding of $\mathbf{C}^{(i)}$ with non-increasing spectral transformation function g [Zhu et al., 2006]. For $g(\cdot) = 1$, $\phi_g(\cdot)$ is called the *Standard Laplacian*

Eigenmap. The dimension of the output space, K, is usually determined using spectral gap analysis. The assumption is then the optimal K is much smaller than the original dimension, d. Given g, one can find the low-dimensional embedding of the input data by finding the eigen decomposition of $\mathbf{L}$.

The Large N arises when the number of datapoints (fibers in our case), N, on which $\mathbf{L}$ is defined, is so large that storing $\mathbf{L}$ in memory and computing its eigen decomposition become intractable. In our Connectome dataset, each brain image contains $N = 250,000$ fibers which is large enough to cause the problem. In the previous chapter, we saw many solutions to deal with the large-scale datasets for the graph-based methods. The most intuitive solution is to use subsampling to compute $\mathbf{L}$ only on a random subset of datapoints. The problem with this method is we may get very different results depending on the random sample [Kumar et al., 2009]; also, we only get the embedding for the datapoints inside the selected sample.

Algorithm 10: Approximate Large-scale Laplacian Eigen Decomposition

```
1: Partition data into $R$ partitions using K-means
2: Compute $\mathbf{L}_R$ over the cluster centers
3: Find the first $K$ eigenvectors of $\mathbf{L}_R$
4: **for all** $i = 1..N$ and $k = 1..K$ **do**
5:     find the set of $T$ nearest centers for $\mathbf{C}^{(i)}$
6:     approximate $v_k(i)$ using (4.24)
7: **end for**

8: Return the augmented eigenvectors
```

To address this problem, we have developed a robust method to approximate the eigenvectors of $\mathbf{L}$ over all datapoints. Algorithm 10 describes the proposed algorithm. The idea is to first partition the whole datapoints into $R \ll N$ clusters using K-means clustering algorithm which is relatively fast and does not need $O(N^2)$ memory space. To minimize the dependency of the output on the initial positions of the cluster centers, we repeat the process several times with different initial positions and pick the one with the smallest clustering risk. Then we compute the weight matrix, $\mathbf{W}_R$, and subsequently $\mathbf{L}_R$ over the cluster centers which are now of size R instead of N. Having computed the eigenvectors of $\mathbf{L}_R$ (which are R-dimensional), we extend the values of these eigenvectors from cluster centers to the whole datapoints by interpolation. More specifically, for a given datapoint $\mathbf{C}^{(i)}$, we find the set N_i of T nearest cluster centers, then we approximate the full eigenvector by the weighted average,

$$\forall k = 1..K,\ \forall i = 1..N,\ v_k(i) \leftarrow \frac{\sum_{u \in N_i} w_\sigma(i,u) v_k(u)}{\sum_{u \in N_i} w_\sigma(i,u)} \tag{4.24}$$

To summarize, we have developed a two-stage framework: in the first stage, we transform the infinite-dimensional space of 3D curves into a (possibly high-dimensional) vector space in which the coordinates as well as the Euclidean distance are meaningful (each resulted coordinate reveals the contribution of a specific basis function in formation of the curve). This might be sufficient for applications like clustering which only need a good distance measure. However, for other applications such as density estimation that are sensitive to the dimensionality of input space, it is essential to reduce the dimensionality without losing the cluster structure. One standard solution is to use Laplacian eigenmaps; however, we need a large-scale version of it since we have too many datapoints (fibers) in the dataset. That is what we have developed in the second stage of the framework.

4.3.4 Results and Discussion

Customizing our framework for the Connectome data, a good candidate for the basis $\mathbf{\Psi}$ will be *orthonormal B-splines* since fibers are known to be smooth non-periodic 3D curves, which are piece-wise polynomial functions with local support. The number of pieces between $[0,1]$ (a.k.a. resolution) on which the B-splines are defined determines the number of basis functions, m. Increasing resolution will increase m which in turn makes the curve approximation more accurate. By performing leave-one-out cross validation, we pick $m = 32$ resulting in 96-dimensional space for the vector representation of fibers (i.e. $d = 96$). For the second stage parameters, we have set $R = 1000$ and $T = 3$. Also, we have chosen the output dimension, $K = 10$.

Figure 19 (left) shows the first 3 dimensions of the resulted embedding of a specific brain. Each point represents a fiber in the original space. Fibers in the embedded space form a geometrical shape with highly concentrated vertices connected by edges (continuous paths) between them. We also observed that the fibers mapped to the vertices actually form visible bundles (clusters) in the original Connectome image. Figure 19 (right) shows the bundles corresponded to some of the vertices on the left side. Due to the preservation of similarity

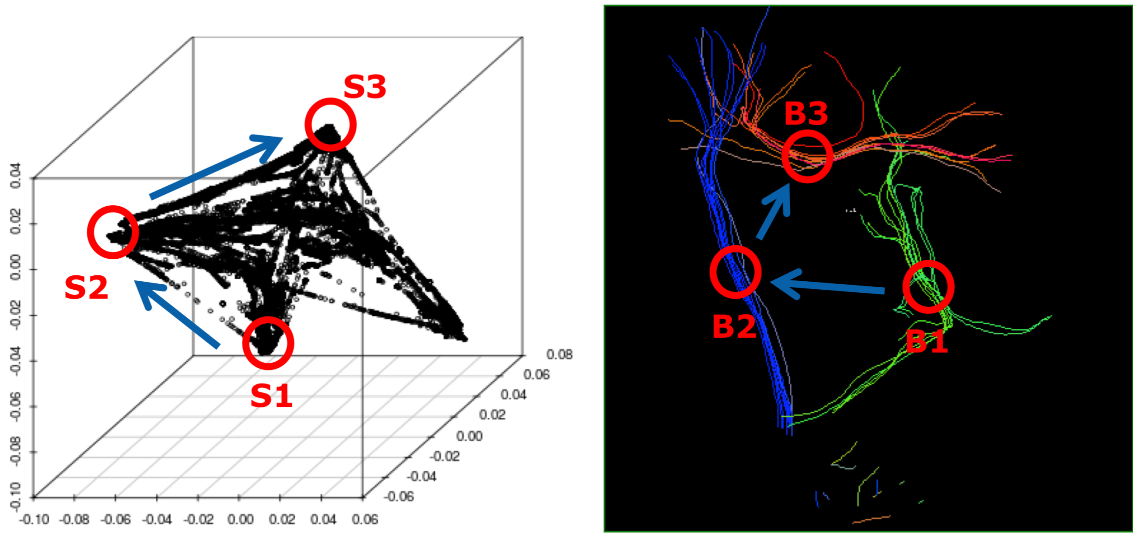

Figure 19: (Left) The embedded fibers (Right) The fibers corresponded to vertices.

in the embedded space, the fibers corresponded to nearby points along the edges are very similar in terms of shape and location and in fact show the *gradual shape transition* paths from one bundle (vertex) to another one (e.g. $S_1 \to S_2 \to S_3$ in Figure 19 (left)). Furthermore, these transition paths also reveal that the shape transition between two bundles in the brain cannot be arbitrary; that is, it can only happen through one of the *plausible* paths uncovered in the embedded space. In our dataset, we have cluster labels for a small portion of fibers. We used these labels to compare the separation of these fibers in vector space resulted from the first stage with the embedded space(s) resulted from the second stage. Separation in 96-dimensional space is 95.89% while the same quantity is 71.75%, 94.01%, 96.06%, using only the first 2, 3 and 10 dimensions of the embedded space, respectively. This shows that separation is preserved while dimensionality can be significantly reduced to 3, making the entire dataset approachable for applications such as density estimation and classification.

5.0 CONCLUSIONS AND FUTURE DIRECTIONS

5.1 CONTRIBUTIONS

In this thesis, we started with the detailed description of some of the famous graph-based methods in Machine Learning and their applications. We showed the applicability and the effectiveness of these methods on the real problem of computing data-driven similarity among (chunks) of terms in the text analysis domain. In particular, the proposed methodology features some important and practical characteristics:

(a) The induced similarity/distance is completely data-driven and therefore can adapt itself to different text or topic domains.

(b) The similarity/distance can be computed between *any* two terms even if they did not co-occur in the same sentence/paragraph/document.

(c) The similarity/distance can be computed not only between two terms but also between any two sets of terms with arbitrary sizes. We referred to this feature as the set-set similarity.

(d) The proposed framework is general and gives us the flexibility of choosing the similarity kernel.

(e) Using some techniques like the Nystrom approximation, the computation of set-set similarities is carried out in the linear time and therefore, it makes the proposed framework very efficient for practical purposes.

(f) By further usage of the Nystrom approximation in combination with the subsampling techniques, we showed the framework can scale up to large-scale term spaces without losing significantly in terms of the accuracy.

(g) The proposed framework showed its merits in practice by outperforming state-of-the-art baseline methods for two tasks of document retrieval and query expansion.

After making our case about the importance of graph-based methods in Machine Learning and their main advantages, we directed our attention to the problem of scalability for the graph-based methods when facing large-scale datasets. In particular, we focused on two main challenges: dealing with large number of datapoints (aka the Large N problem) and dealing with high dimensionality (aka the Large d problem).

As we stated in this thesis, large number of datapoints poses mainly a computational problem for the graph-based methods. The inherent quadratic computational and memory complexity of the graph-based methods makes them impractical choices for the real-world large-scale datasets. As we discussed before, this is a special case of the classical N-body problems. After reviewing the existing methods in the literature that address these problems for graph-based methods, we proposed our Variational Dual-tree (VDT) framework to address a specific problem in the graph-based methods: approximating the large-scale transition matrices of the random walk on similarity graphs. As we illustrated through this thesis, the random walk transition matrix plays the key role for many graph-based techniques such as Diffusion Maps, Laplacian Eigenmaps, label propagation on graphs, etc. The proposed VDT framework approaches the problem hierarchically reduce the transition matrix into blocks and estimating the block values using the variation approximation. The main features of this framework are as follows:

(a) The number of parameters in the transition matrix is reduced from $O(N^2)$ to $O(|\mathscr{B}|)$, where $|\mathscr{B}|$ is the number of blocks used to approximate the transition matrix and can be as small as $O(N)$. This is a huge gain in terms of the memory complexity.

(b) The construction time of the transition matrix is reduced from $O(N^2)$ to $O(N^{1.5}\log N + |\mathscr{B}|)$ if we use the Anchor Tree method for building the cluster hierarchy. If the cluster hierarchy is given, the construction time will be $O(|\mathscr{B}|)$ which again can be as small as $O(N)$ depending on the desired accuracy.

(c) The proposed framework is independent of the hierarchical clustering method used; therefore, one can use other metric trees than the anchor trees like kd-trees, ball trees, etc.

(d) The proposed framework provides a fast matrix-vector multiplication algorithm that can relieve the crucial matrix-vector bottleneck in many applications such as label propagation and eigen decomposition.

(e) The proposed framework provides an optimization-based technique to adjust the bandwidth parameter of the Gaussian kernel for the similarity graph construction.

(f) The refinement process included in the framework provides a systematic mechanism to adjust the trade-off between approximation accuracy and computational complexity of the model.

(g) As opposed to the sparsity-based methods such as k-nearest-neighbor (kNN), the resulted matrix (graph) from VDT is always connected. This is a crucial property for applications such as label propagation or diffusion on graphs where disconnectivity can hurt the end result. This property is in fact the direct result of the parameter sharing idea rather than sparsification behind our framework.

(h) By applying VDT on real datasets, we observed that while VDT pays a couple of percent in terms of accuracy, it gains orders of magnitude in terms of time and memory complexity compared to the exact method for computation of transition matrices.

(i) Due the connectivity of its resulted graph, VDT showed more robust behavior during the refinement process compared to its rival kNN method in the experiments.

(j) We showed that VDT can scale up to gigantic datasets with millions of datapoints.

The VDT framework is restricted to the Euclidean spaces where the local similarity between the datapoints is derived from the Euclidean distance. However, as explained in the thesis, this is a major restriction especially for those datasets where the Euclidean distance is simply not the best way to express similarity. On the other hand, replacing the Euclidean distance with an arbitrary distance can seriously endanger the efficiency of the VDT framework as discussed in more details before. To address this problem, we extend the VDT framework to support the big class of Bregman divergences of which the Euclidean distance is a special case. We called this general framework the Bregman Variational Dual-tree (BVDT) framework. The main features of BVDT can be summarized as:

(a) The class of Bregman divergences covers a large set of distances and divergences in Machine Learning such as Euclidean distance, KL-Divergence, Generalized I-Divergence (GID), Itakura-Saito distance, Logistic Loss, Mahalanobis Distance, etc. Therefore by extending VDT to BVDT, we effectively extend our proposed framework to all these different distances and divergences.

(b) Due to the special form of the general Bregman divergence, BVDT still keeps the same computational complexity as VDT and therefore stays a fast method for transition matrix computation.

(c) By utilizing the connection between the Bregman divergences and the exponential families, BVDT provides a neat probabilistic interpretation.

(d) The connection to the exponential families can be further used to derive the proper Bregman divergence for a given problem using the probabilistic modeling of the data generation process. We specifically used this procedure in our experiments to derive the GID for the frequency data.

(e) As a byproduct, BVDT extends the Euclidean anchor tree construction algorithm to support Bregman spaces.

(f) VDT is in fact a special case of BVDT where the Bregman divergence used is the Euclidean distance.

(g) The experiments on the simulated frequency data as well as the real text datasets show that BVDT can dramatically improve the accuracy of the VDT model while still enjoying the same computational benefits gained by the original VDT framework.

Finally, in the last part of this thesis, we focused our attention on the problem of high-dimensionality which has a statistical nature. As we explained, the graph structure resulted from a finite set of datapoints in a high-dimensional space can be quite different from the true manifold structure of the population in that space. In diffusion maps, this difference is reflected in terms the error between the eigenfunctions of the empirical and the true diffusion operators. This error exponentially depends on the dimensionality which signals the classical curse of dimensionality. To address this problem for diffusion maps, we proposed the factorized diffusion maps which uses the independence assumptions in the underlying distribution to conquer the curse of dimensionality. In particular, we showed that if the underlying distribution is factorizable into independent subspaces, then the eigenfunctions of the diffusion operator can be computed as the product of the eigenfunctions in the subspaces. The factorized diffusion framework provides the following features:

(a) The key idea of the proposed framework is to use the structure in the underlying distribution to compensate for the lack of enough examples.

(b) In case the underlying distribution is factorizable, the factorized diffusion framework exponentially decreases the estimation error of the eigenfunctions in the low-dimensional subspaces.

(c) In case the underlying distribution is almost factorizable, the factorized diffusion framework exponentially decreases the estimation error of the eigenfunctions in the low-dimensional subspaces while introducing a bounded approximation error (or bias).

(d) We derived theoretical upper bounds for the estimation and the approximation errors.

(e) In case the factorization is not known a priori, the framework provides an algorithm to automatically find the best factorization that minimizes the empirical bootstrap error to find an optimal trade-off between the estimation and approximation errors.

(f) In our experiments on synthetic and real datasets, we showed that the results of spectral clustering can be improved by using the factorized diffusion framework.

Furthermore, we proposed some practical techniques to deal with infinite dimensional spaces. In particular, we considered functional spaces where each datapoint is a function. Our proposed technique involves expressing each functional datapoint in a fixed size functional basis. To evaluate the applicability of our method, we apply it on the human brain connectome data. The human brain connectome consists of the brain fibers where each fiber is a 3D smooth curve. By considering each fiber as datapoint, one typical task on such a dataset is to cluster the fibers into fiber bundles. However, this is a non-trivial task due the functional nature of datapoints. By applying our technique combined with the Laplacian embedding, we could embed the human brain connectome into a low-dimensional space which can be further fed to a clustering algorithm.

5.2 OPEN QUESTIONS

The proposed frameworks in this thesis are complete and self-contained in the sense that they provide theoretical solutions for the problems they are designed for, and at the same time they are equipped with a toolset of useful techniques to deal with some practical issues in practice. However, these methods have their own limitations and open questions which are either dictated by the scope of the problems or left unanswered due to our time constraints. In this section, we list a set of more important open questions for the two main proposed frameworks for the Large N and the Large d problems.

Starting with the Large N problem, the main proposed frameworks in this thesis to address this problem for the graph-based methods were the VDT and its generalization the BVDT. Although, these methods performed well for the designated tasks, we still face some open questions:

(a) The VDT and the BVDT frameworks are essentially designed for the computation of large-scale transition matrices and they cannot be used to compute other types of matrices in the graph-based methods such as the similarity matrix.

(b) The fundamental assumption in these methods is the data resides in a metric space with a valid Euclidean distance or Bregman divergence. Therefore, it is not clear yet how to apply these methods to those problems with non-vector datapoints.

(c) The cluster tree construction phase can be quite sensitive to the structure of data in the input space such that in extreme cases the complexity of the proposed frameworks can be severely degraded due to the lack of structure in a high-dimensional input space.

(d) Although the proposed frameworks are independent of the cluster tree construction algorithm, we did not use other kinds of trees than the anchor trees in the current work. It can be worthwhile to use other types of trees like kd-trees or Ball trees and compare the results.

(e) More importantly, the tree construction phase is purely algorithmic and not directly encoded in the objective function of the variational approximation. Yet, we know that the quality of the cluster tree heavily affects the quality of the final approximation.

Therefore, one crucial question is how to directly incorporate the structure of the cluster tree in the optimization.

(f) With regard to the refinement process, we assumed that the maximum allowed number of blocks is an input parameter to the framework; that is, the user can specify the available budget in terms of the computational resources; then the proposed refinement algorithm tries to split the blocks in a way to maximize the accuracy until it reaches the maximum budget. However, we did not provide any error guarantee. More precisely, does the proposed technique for refinement give us an error upper bound for a given tree structure and a maximum budget? The other interesting question is can we design a refinement algorithm that accepts an error requirement as the input and find a partitioning with minimum number of blocks that meets the requirement.

(g) In our experiments for evaluating VDT and BVDT, we focused on label propagation for the semi-supervised learning problem. It would be interesting to evaluate the performance these methods in other Machine Learning applications such as eigen decomposition of the transition matrix for spectral clustering or low-dimensional embedding.

(h) In terms the baseline methods used in the experiments, we could not compare our frameworks with some of the methods in the N-body problems literature. The main reason for this is most of these methods are designed for fast kernel density estimation and *not* the transition matrix computation. Therefore, an interesting future investigation is to see whether these methods can inspire new fast algorithms for transition matrix approximation.

As for the Large d problem, the main proposed framework in this thesis was the factorized diffusion maps. In spite of its successful performance in our empirical evaluations, this framework also comes with some restrictions and unanswered questions. Here are some of the important ones:

(a) The current work is exclusively designed for diffusion maps. A natural extension is to extend the framework for other graph-based methods as well.

(b) The proposed framework essentially uses the unconditional independences in the underlying distribution for the factorization. One interesting question is whether one can somehow extend the framework to benefit from more general conditional independence assumptions as well.

(c) In those problems, where the factorization of the space is not given a priori, one needs to use the proposed partitioning algorithm; however, this algorithm is computationally expensive specially for high dimensional problems.

(d) Also, with limited number of examples, the empirical estimation of mutual informations used in the partitioning algorithm is highly inaccurate for high dimensional problems.

(e) Although the derived upper bounds on the estimation and the approximation errors give an insight on how fast the error terms grow, they are not tight and contain constant factors that are not computable in practical cases.

(f) The computational order of finding *all* the eigenvalues and the eigenfunctions of the factorized diffusion map with K subspaces is $O(N^K)$. This is clearly infeasible for problems with large K.

5.3 FUTURE DIRECTIONS

The presented methodologies and algorithms in this work open the door to new directions both in terms of the theoretical research and the applied side. Of course, the most immediate and natural extension of the current work is to resolve or improve the limitations presented in the previous section. However, in this section, we draw more general and fundamental future directions that can profoundly revolutionize the applicability and the performance of the proposed methods in this thesis. In particular, we have found the following directions to be the most promising:

(a) The key essence of the proposed methodologies to improve the computational complexity in this work is *approximation*. The other fundamental idea that can be incorporated for this purpose independent of approximation is *parallelization*. With cutting-edge technology in distributed computing and widespread usage of parallel processors, incorporating parallelization in our models opens a new avenue to speed up already-developed scalable solutions. For example. it is not hard to see that in the proposed factorized diffusion maps the computations in different subspaces can be done completely in parallel. Another interesting observation is that the underlying data structure for the proposed VDT and BVDT frameworks is binary tree which has a recursive structure by definition. Therefore, one can naturally think of parallelizing these frameworks by assigning different subtrees to different processors.

(b) The other fundamental assumption made through all the methods in this thesis was that we have the whole dataset in batch. However, this is not the case when one has to work with data streams or real-time systems. In most cases, the high computational cost of learning and modeling prohibits us from re-learning the model every time a new example arrives. This observation, in fact, signifies the importance of *online learning*. However, online learning can be quite challenging both algorithmically and statistically for the graph-based methods. Another closely-related observation is that the graph-based methods are inherently non-parametric and therefore *transductive*. Yet in many Machine Learning paradigms such as classification, a desired method needs to be *inductive* in order to be able to deal with future cases in timely manner.

(c) In this thesis, we studied the large N and the Large d problems in isolation. An interesting question is what if we have to deal with both at the same time? How can we integrate a system that applies both sets of solution simultaneously? Of course, the main dilemma here is how these two sets of solutions interact with each other. Unfortunately, they can go against each other sometimes. For example, for the Large d problem, the more datapoints we have in the dataset, the more statistically accurate analysis can be done to conquer the curse of dimensionality. However, more datapoints means a worsened Large N problem.

(d) The Large d problem or equivalently the curse of dimensionality is not specific to the graph-based methods. In fact, this problem has been studied deeply in other fields of Statistical Machine Learning such as density estimation. Therefore, another interesting future direction is to investigate the exact relationship between these two paradigms, namely the eigenfunctions of the similarity graph (or its variants) and the underlying density function. Moreover, the fact that both the density function and the similarity graph encode the structure of data in the input space, give us another sign that the graph-based methods might be inherently related to the density function. Note that we already used this connection to some extent to develop the VDT framework in Chapter 3.

(e) Throughout this thesis, whenever we referred to graphs, we meant data graphs whose node are the datapoints. However, these are not the only type of graphs in Machine Learning. A very famous counter-example is a *graphical model* whose nodes represent the variables (or features) in the problem. Although, graphical models are used for a completely different purpose, a very exciting question is whether we can somehow link these two paradigms. If such a connection exists, we might be able to move over many of the methods developed for one paradigm to the other one. For example, can we somehow use the scalable methods in this thesis for fast structure learning and inference in graphical models.

The items (a)-(c) have more practical implications, while (d)-(e) raise more interesting theoretical questions. Either way, these directions introduce new insights into the current work and help the interested researchers and practitioners in the field to further expand the reach of the methods proposed in this dissertation.

BIBLIOGRAPHY

[Amizadeh et al., 2012a] Amizadeh, S., Thiesson, B., and Hauskrecht, M. (2012a). Variational dual-tree framework for large-scale transition matrix approximation. In *the 28th Conference on Uncertainty in Artificial Intelligence (UAI-12)*, pages 64–73.

[Amizadeh et al., 2013] Amizadeh, S., Thiesson, B., and Hauskrecht, M. (2013). The bregman variational dual-tree framework. In *the 29th Conference on Uncertainty in Artificial Intelligence (UAI-13)*, pages 22–31.

[Amizadeh et al., 2012b] Amizadeh, S., Valizadegan, H., and Hauskrecht, M. (2012b). Factorized diffusion map approximation. *Journal of Machine Learning Research - Proceedings Track*, 22:37–46.

[Amizadeh et al., 2011] Amizadeh, S., Wang, S., and Hauskrecht, M. (2011). An efficient framework for constructing generalized locally-induced text metrics. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two*, pages 1159–1164. AAAI Press.

[Bach, 2008] Bach, F. (2008). Graph kernels between point clouds. In *ICML*.

[Bach and Jordan, 2004] Bach, F. R. and Jordan, M. I. (2004). Learning spectral clustering. In *Advances in Neural Information Processing Systems 16*.

[Bache and Lichman, 2013] Bache, K. and Lichman, M. (2013). UCI machine learning repository.

[Banerjee et al., 2005] Banerjee, A., Merugu, S., Dhillon, I. S., and Ghosh, J. (2005). Clustering with bregman divergences. *JMLR*, 6:1705–1749.

[Barnes and Hut, 1986] Barnes, J. and Hut, P. (1986). A hierarchical o (n log n) force-calculation algorithm.

[Belkin and Niyogi, 2002] Belkin, M. and Niyogi, P. (2002). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15:1373–1396.

[Belkin and Niyogi, 2005] Belkin, M. and Niyogi, P. (2005). Towards a theoretical foundation for laplacian-based manifold methods. In *Computational Learning Theory*, pages 486–500.

[Belkin et al., 2006] Belkin, M., Niyogi, P., and Sindhwani, V. (2006). Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:2399–2434.

[Beygelzimer et al., 2006] Beygelzimer, A., Kakade, S., and Langford, J. (2006). Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 97–104, New York, NY, USA. ACM.

[Bicego et al., 2006] Bicego, M., Lagorio, A., Grosso, E., and Tistarelli, M. (2006). On the use of sift features for face authentication. In *Computer Vision and Pattern Recognition Workshop, 2006. CVPRW'06. Conference on*, pages 35–35. IEEE.

[Bock et al., 2004] Bock, R., Chilingarian, A., Gaug, M., Hakl, F., Hengstebeck, T., Jiřina, M., Klaschka, J., Kotrč, E., Savickỳ, P., Towers, S., et al. (2004). Methods for multidimensional event classification: a case study using images from a cherenkov gamma-ray telescope. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 516(2):511–528.

[Bordino et al., 2010] Bordino, I., C. Castillo, D. D., and Gionis, A. (2010). Query similarity by projecting the query-flow graph. In *SIGIR '10*, pages 515–522. ACM.

[Brito et al., 1997] Brito, M., Chavez, E., Quiroz, A., and Yukich, J. (1997). Connectivity of the mutual k-nearest-neighbor graph in clustering and outlier detection. *Statistics & Probability Letters*, 35(1):33–42.

[Buchman et al., 2009] Buchman, S. M., Lee, A. B., and Schafer, C. M. (2009). High-Dimensional Density Estimation via SCA: An Example in the Modelling of Hurricane Tracks. *Statistical Methodology*, 8(1):18–30.

[Buckley and Voorhees, 2005] Buckley, C. and Voorhees, E. M. (2005). Retrieval system evaluation. *TREC: experiment and evaluation in information retrieval*.

[Cai et al., 2005] Cai, D., He, X., and Han, J. (2005). Document clustering using locality preserving indexing. *IEEE Trans. Knowl. Data Eng*, 17(12):1624–1637.

[Caillet et al., 2004] Caillet, M., Pessiot, J., Amini, M., and Gallinari, P. (2004). Unsupervised learning with term clustering for thematic segmentation of texts.

[Carreira-Perpinán and Zemel, 2005] Carreira-Perpinán, M. and Zemel, R. (2005). Proximity graphs for clustering and manifold learning. *Advances in neural information processing systems*, 17:225–232.

[Carrier et al., 1988] Carrier, J., Greengard, L., and Rokhlin, V. (1988). A fast adaptive multipole algorithm for particle simulations. *SIAM Journal on Scientific and Statistical Computing*, 9(4):669–686.

[Chapelle et al., 2006] Chapelle, O., Schölkopf, B., and Zien, A., editors (2006). *Semi-Supervised Learning*. MIT Press, Cambridge, MA.

[Charikar et al., 1997] Charikar, M., Chekuri, C., Feder, T., and Motwani, R. (1997). Incremental clustering and dynamic information retrieval. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 626–635. ACM.

[Cheng, 1995] Cheng, Y. (1995). Mean shift, mode seeking, and clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(8):790–799.

[Chennubhotla and Jepson, 2005] Chennubhotla, C. and Jepson, A. (2005). Hierarchical eigensolver for transition matrices in spectral methods. *Advances in Neural Information Processing Systems*, 17:273–280.

[Chung, 1997] Chung, F. R. K. (1997). *Spectral Graph Theory*. Amer Mathematical Society.

[Cohn and Chang, 2000] Cohn, D. and Chang, H. (2000). Learning to probabilistically identify authoritative documents. In *ICML '00*, pages 167–174.

[Coifman et al., 2008] Coifman, R. R., Kevrekidis, I. G., Lafon, S., Maggioni, M., and Nadler, B. (2008). Diffusion maps, reduction coordinates, and low dimensional representation of stochastic systems. *Multiscale Modeling & Simulation*, 7(2):842–864.

[Coifman et al., 2005a] Coifman, R. R., Lafon, S., Lee, A. B., Maggioni, M., Nadler, B., Warner, F., and Zucker, S. W. (2005a). Geometric diffusions as a tool for harmonic analysis and structure definition of data. part i: Diffusion maps. (102):7426–7431.

[Coifman et al., 2005b] Coifman, R. R., Lafon, S., Lee, A. B., Maggioni, M., Nadler, B., Warner, F., and Zucker, S. W. (2005b). Geometric diffusions as a tool for harmonic analysis and structure definition of data. part ii: Multiscale methods. (102):7432–7438.

[Collins-Thompson and Callan, 2005] Collins-Thompson, K. and Callan, J. (2005). Query expansion using random walk models. In *CIKM '05*, pages 704–711.

[Cover and Thomas, 2000] Cover, T. M. and Thomas, J. A. (2000). *Elements of Information Theory*. Wiley-Interscience, New York, USA.

[Cox and Cox, 1994] Cox, T. and Cox, M. (1994). *Multidimensional Scaling*. Chapman & Hall.

[Deng et al., 2011] Deng, H., Han, J., Zhao, B., Yu, Y., and Lin, C. X. (2011). Probabilistic topic models with biased propagation on heterogeneous information networks. In *KDD*, pages 1271–1279.

[Dhillon and Sra, 2005] Dhillon, I. and Sra, S. (2005). *Generalized nonnegative matrix approximations with Bregman divergences*.

[Dillon et al., 2007] Dillon, J., Mao, Y., Lebanon, G., and Zhang, J. (2007). Statistical translation, heat kernels, and expected distance. In *UAI*, pages 93–100.

[Fergus et al., 2009] Fergus, R., Weiss, Y., and Torralba, A. (2009). Semi-supervised learning in gigantic image collection. In *NIPS*.

[Friedman et al., 1977] Friedman, J., Bentley, J., and Finkel, R. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226.

[G. Cao and Robertson, 2008] G. Cao, J. Y. Nie, J. G. and Robertson, S. (2008). Selecting good expansion terms for pseudo-relevance feedback. In *SIGIR '08*, pages 243–250.

[Garcke and Griebel, 2005] Garcke, J. and Griebel, M. (2005). Semi-supervised learning with sparse grids. In *Proc. of the 22nd ICML Workshop on Learning with Partially Classified Training Data*.

[Giné and Guillou, 2002] Giné, E. and Guillou, A. (2002). Rates of strong uniform consistency for multivariate kernel density estimators. *Ann. Inst. Henri Poincaré (B)*, 38(6):907–921.

[Gine and Koltchinskii, 2006] Gine, E. and Koltchinskii, V. (2006). Empirical graph laplacian approximation of laplace–beltrami operators: Large sample results. *the IMS Lecture Notes Monograph Series by the Institute of Mathematical Statistics*, 51.

[Gonzalez, 1985] Gonzalez, T. F. (1985). Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38(2–3):293–306.

[Gray and Moore, 2003] Gray, A. and Moore, A. (2003). Nonparametric density estimation: Toward computational tractability. In *Proceedings of the third SIAM International Conference on Data Mining*, volume 112, page 203. Society for Industrial & Applied.

[Gray and Moore, 2000] Gray, A. G. and Moore, A. W. (2000). 'N-body' problems in statistical learning. In *NIPS*, pages 521–527. MIT Press.

[Greene and Cunningham, 2006] Greene, D. and Cunningham, P. (2006). Practical solutions to the problem of diagonal dominance in kernel document clustering. In *ICML*, pages 377–384.

[Greengard, 1994] Greengard, L. (1994). Fast algorithms for classical physics. *Science*, 265(5174):909–914.

[Greengard and Strain, 1991] Greengard, L. and Strain, J. (1991). The fast gauss transform. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94.

[Ham et al., 2004] Ham, J., Lee, D., Mika, S., and Schölkopf, B. (2004). A kernel view of the dimensionality reduction of manifolds. In *Proceedings of the twenty-first international conference on Machine learning*, page 47. ACM.

[He et al., 2005] He, X., Cai, D., Yan, S., and Zhang, H. (2005). Neighborhood preserving embedding. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1208–1213. IEEE.

[Jebara et al., 2009] Jebara, T., Wang, J., and Chang, S. (2009). Graph construction and b-matching for semi-supervised learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, volume 382, page 56. ACM.

[Jin et al., 2003] Jin, Q., Zhao, J., and Xu, B. (2003). Query expansion based on term similarity tree model. In *Natural Language Processing and Knowledge Engineering, 2003. Proceedings. 2003 International Conference on*, pages 400–406. IEEE.

[Jin et al., 2006] Jin, R., Ding, C., and Kang, F. (2006). A probabilistic approach for optimizing spectral clustering. In *Advances in Neural Information Processing Systems 18*.

[Karger and Ruhl, 2002] Karger, D. and Ruhl, M. (2002). Finding nearest neighbors in growth-restricted metrics. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 741–750. ACM.

[Kleinberg, 2006] Kleinberg, J. (2006). *Algorithm design*. Pearson Education India.

[Kondor and Jebara, 2003] Kondor, R. I. and Jebara, T. (2003). A kernel between sets of vectors. In *ICML*, pages 361–368. AAAI Press.

[Krauthgamer and Lee, 2004] Krauthgamer, R. and Lee, J. (2004). Navigating nets: simple algorithms for proximity search. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 798–807. Society for Industrial and Applied Mathematics.

[Kumar et al., 2009] Kumar, S., Mohri, M., and Talwalkar, A. (2009). Sampling techniques for the nystrom method. *Journal of Machine Learning Research - Proceedings Track*, 5:304–311.

[Lafon and Lee, 2006] Lafon, S. and Lee, A. B. (2006). Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning, and data set parameterization. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 28(9):1393–1403.

[Lang et al., 2005] Lang, D., Klaas, M., and de Freitas, N. (2005). Empirical testing of fast kernel density estimation algorithms. *UBC Technical repor*, 2.

[Lang, 1995] Lang, K. (1995). News weeder: Learning to filter netnews. In *Machine Learning International Workshop*, pages 331–339. Morgan Kufmann Publishers, Inc.

[Lebanon, 2006] Lebanon, G. (2006). Metric learning for text documents. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 28(4):497–508.

[Ledwich and Williams, 2004] Ledwich, L. and Williams, S. (2004). Reduced sift features for image retrieval and indoor localisation. In *Australian conference on robotics and automation*, volume 322. Citeseer.

[Lee and Wasserman, 2010] Lee, A. B. and Wasserman, L. (2010). Spectral connectivity analysis. *Journal of the American Statistical Association*.

[Lee and Gray, 2008] Lee, D. and Gray, A. (2008). Fast high-dimensional kernel summations using the monte carlo multipole method. *Advances in Neural Information Processing Systems*, 21.

[Lee et al., 2011] Lee, D., Gray, A., and Moore, A. (2011). Dual-tree fast gauss transforms. *arXiv preprint arXiv:1102.2878*.

[Liaw et al., 2010] Liaw, Y., Leou, M., and Wu, C. (2010). Fast exact k nearest neighbors search using an orthogonal search tree. *Pattern Recognition*, 43(6):2351–2358.

[Lin, 2003] Lin, J. (2003). Reduced rank approximations of transition matrices. In *Proceedings of the Sixth International Conference on Artificial Intelligence and Statistics*, pages 3–6.

[Ling and Okada, 2007] Ling, H. and Okada, K. (2007). An efficient earth mover's distance algorithm for robust histogram comparison. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(5):840–853.

[Liu et al., 2010] Liu, W., He, J., and Chang, S. (2010). Large graph construction for scalable semi-supervised learning. In *Proceedings of the 27th International Conference on Machine Learning*, pages 679–686.

[Logan and Salomon, 2001] Logan, B. and Salomon, A. (2001). A music similarity function based on signal analysis. In *ICME 2001*, pages 745–748.

[Losada and Barreiro, 2003] Losada, D. and Barreiro, A. (2003). Embedding term similarity and inverse document frequency into a logical model of information retrieval. *Journal of the American Society for Information Science and Technology*, 54(4):285–301.

[Maas et al., 2011] Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proc. of 49th An. Meeting of the Assoc. for Comp. Ling.*, pages 142–150.

[Macdonald et al., 2005] Macdonald, C., He, B., Plachouras, V., and Ounis, I. (2005). University of glasgow at trec 2005: Experiments in terabyte and enterprise tracks with terrier. In *TREC '05*.

[Maiker et al., 2009] Maiker, M., v. Luxburg, U., and Hein, M. (2009). Influence of graph construction on graph-based clustering measures.

[Meila, 2003] Meila, M. (2003). Comparing clusterings by the variation of information. In *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers*.

[Moldovan and Rus, 2001] Moldovan, D. and Rus, V. (2001). Explaining answers with extended wordnet. In *ACL '01*.

[Moore, 1991] Moore, A. W. (1991). An introductory tutorial on kd-trees. Technical Report No. 209, Computer Laboratory, University of Cambridge.

[Moore, 2000] Moore, A. W. (2000). The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 397–405.

[Mori and van Zijl, 2002] Mori, S. and van Zijl, P. C. M. (2002). Fiber tracking: principles and strategies - a technical review. *NMR in Biomedicine*.

[Nadler et al., 2006] Nadler, B., Lafon, S., Coifman, R. R., and Kevrekidis, I. G. (2006). Diffusion maps, spectral clustering and reaction coordinates of dynamical systems. In *Applied and Computational Harmonic Analysis: Special issue on Diffusion Maps and Wavelets*.

[Narasimhan and Bilmes, 2004] Narasimhan, M. and Bilmes, J. A. (2004). PAC-learning bounded tree-width graphical models. In *UAI-04*, pages 410–417. AUAI Press.

[Ng et al., 2001a] Ng, A., Jordan, M., and Weiss, Y. (2001a). On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*.

[Ng et al., 2001b] Ng, A. Y., Jordan, M. I., and Weiss, Y. (2001b). On spectral clustering: Analysis and an algorithm. In *NIPS '01*, pages 849–856.

[Ng et al., 2001c] Ng, A. Y., Zheng, A. X., and Jordan, M. I. (2001c). Link analysis, eigenvectors and stability. *IJCAI*, pages 903–910.

[Ng et al., 2001d] Ng, A. Y., Zheng, A. X., and Jordan, M. I. (2001d). Stable algorithms for link analysis. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on research and development in information retrieval*, pages 258–266. ACM Press.

[Niyogi, 2004] Niyogi, P. (2004). Locality preserving projections. *Advances in neural information processing systems*, 16:153–160.

[Penrose, 1999] Penrose, M. (1999). A strong law for the longest edge of the minimal spanning tree. *The Annals of Probability*, 27(1):246–260.

[Qian et al., 2011] Qian, J., Saligrama, V., and Zhao, M. (2011). Graph construction for learning with unbalanced data. *arXiv preprint arXiv:1112.2319*.

[Qiao et al., 2010] Qiao, L., Chen, S., and Tan, X. (2010). Sparsity preserving projections with applications to face recognition. *Pattern Recognition*, 43(1):331–341.

[Ram et al., 2009] Ram, P., Lee, D., March, W. B., and Gray, A. G. (2009). Linear-time algorithms for pairwise statistical problems. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009, Vancouver, British Columbia, Canada*, pages 1527–1535.

[Raykar et al., 2005] Raykar, V. C., Yang, C., Duraiswami, R., and Gumerov, N. (2005). Fast computation of sums of gaussians in high dimensions. Technical Report CS-TR-4767, Department of Computer Science, University of Maryland, CollegePark.

[Roe et al., 2005] Roe, B. P., Yang, H.-J., Zhu, J., Liu, Y., Stancu, I., and McGregor, G. (2005). Boosted decision trees as an alternative to artificial neural networks for particle identification. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 543(2):577–584.

[Rossi et al., 2005] Rossi, F., Delannay, N., Conan-Guez, B., and Verleysen, M. (2005). Representation of functional data in neural networks. *Neurocomputing*, 64:183–210.

[Roweis and Saul, 2000] Roweis, S. and Saul, L. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326.

[Rubner et al., 2000] Rubner, Y., Tomasi, C., and Guibas, L. (2000). The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121.

[Salton and McGill, 1983] Salton, G. and McGill, M. J. (1983). *Introduction to modern information retrieval*. McGraw-Hill.

[Shi and Malik, 2000] Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905.

[Shirdhonkar and Jacobs, 2008] Shirdhonkar, S. and Jacobs, D. (2008). Approximate earth moverÕs distance in linear time. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE.

[Silverman, 1982] Silverman, B. (1982). Algorithm as 176: Kernel density estimation using the fast fourier transform. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 31(1):93–99.

[Singer, 2006] Singer, A. (2006). From graph to manifold laplacian: The convergence rate. *Applied and Computational Harmonic Analysis*, 21:128–134.

[Subramanya and Bilmes, 2009] Subramanya, A. and Bilmes, J. (2009). Entropic graph regularization in non-parametric semi-supervised classification. In *NIPS*.

[Suzuki et al., 2008] Suzuki, T., Sugiyama, M., Sese, J., and Kanamori, T. (2008). Approximating mutual information by maximum likelihood density ratio estimation. *Journal of Machine Learning Research - Proceedings Track*, 4:5–20.

[Suzuki et al., 2009] Suzuki, T., Sugiyama, M., and Tanaka, T. (2009). Mutual information approximation via maximum likelihood estimation of density ratio. In *Proceedings of the 2009 IEEE international conference on Symposium on Information Theory - Volume 1*, pages 463–467. IEEE Press.

[Talwalkar et al., 2008] Talwalkar, A., Kumar, S., and Rowley, H. A. (2008). Large-scale manifold learning. In *CVPR*, pages 1–8.

[Telgarsky and Dasgupta, 2012] Telgarsky, M. and Dasgupta, S. (2012). Agglomerative bregman clustering. *arXiv:1206.6446*.

[Tenenbaum et al., 2000] Tenenbaum, J., De Silva, V., and Langford, J. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323.

[Thiesson and Kim, 2012] Thiesson, B. and Kim, J. (2012). Fast variational mode-seeking. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics 2012, JMLR 22: W&CP 22*. Journal of Machine Learning Research.

[Thiesson and Wang, 2010] Thiesson, B. and Wang, C. (2010). Fast large-scale mixture modeling with component-specific data partitions. In *Neural Inform. Process. Syst*, volume 22. Citeseer.

[Tsang and Kwok, 2006] Tsang, I. W. and Kwok, J. T. (2006). Large-scale sparsified manifold regularization. In *NIPS*, pages 1401–1408. MIT Press.

[Valizadegan et al., 2008] Valizadegan, H., Jin, R., and Jain, A. K. (2008). Semi-supervised boosting for multi-class classification. In *Principles of Data Mining and Knowledge Discovery*, pages 522–537.

[Valko et al., 2012] Valko, M., Kveton, B., Huang, L., and Ting, D. (2012). Online semi-supervised learning on quantized graphs. *arXiv preprint arXiv:1203.3522*.

[von Luxburg, 2007] von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416.

[von Luxburg et al., 2008] von Luxburg, U., Belkin, M., and Planck, M. (2008). Consistency of spectral clustering. *Annals of Statistics*, 36:555–586.

[Wan and Peng, 2005] Wan, X. and Peng, Y. (2005). The earth mover's distance as a semantic measure for document similarity. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 301–302. ACM.

[Wang and Dong, 2012] Wang, L. and Dong, M. (2012). Multi-level low-rank approximation-based spectral clustering for image segmentation. *Pattern Recognition Letters*.

[Wang and Hauskrecht, 2008] Wang, S. and Hauskrecht, M. (2008). Improving biomedical document retrieval using domain knowledge. In *SIGIR '08*, pages 785–786. ACM.

[Wang et al., 2010] Wang, S., Hauskrecht, M., and Visweswaran, S. (2010). Candidate gene prioritization using network based probabilistic models. In *AMIA TBI*.

[Williams and Seeger, 2001] Williams, C. and Seeger, M. (2001). Using the nystrom method to speed up kernel machines. *Advances in neural information processing systems*, pages 682–688.

[Xu and Croft, 1996] Xu, J. and Croft, B. W. (1996). Query expansion using local and global document analysis. In *SIGIR '96*, pages 4–11. ACM.

[Yan et al., 2009] Yan, D., Huang, L., and Jordan, M. (2009). Fast approximate spectral clustering. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 907–916. ACM.

[Yan et al., 2007] Yan, S., Xu, D., Zhang, B., Zhang, H., Yang, Q., and Lin, S. (2007). Graph embedding and extensions: A general framework for dimensionality reduction. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(1):40–51.

[Yang et al., 2005] Yang, C., Duraiswami, R., Davis, L., et al. (2005). Efficient kernel machines using the improved fast gauss transform. *Advances in neural information processing systems*, 17:1561–1568.

[Yang et al., 2003] Yang, C., Duraiswami, R., Gumerov, N., and Davis, L. (2003). Improved fast gauss transform and efficient kernel density estimation. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 664–671. IEEE.

[Yu et al., 2005] Yu, K., Yu, S., and Tresp, V. (2005). Blockwise supervised inference on large graphs. In *Proc. of the 22nd ICML Workshop on Learning*.

[Zelnik-Manor and Perona, 2005] Zelnik-Manor, L. and Perona, P. (2005). Self-tuning spectral clustering. In *Advances in Neural Information Processing Systems 17*, pages 1601–1608.

[Zhang et al., 2012] Zhang, L., Chen, S., and Qiao, L. (2012). Graph optimization for dimensionality reduction with sparsity constraints. *Pattern Recognition*, 45(3):1205–1210.

[Zhang et al., 2010] Zhang, L., Qiao, L., and Chen, S. (2010). Graph-optimized locality preserving projections. *Pattern Recognition*, 43(6):1993–2002.

[Zhou et al., 2003] Zhou, D., Bousquet, O., Lal, T. N., Weston, J., and Schölkopf, B. (2003). Learning with local and global consistency. In *NIPS*. MIT Press.

[Zhou et al., 2009] Zhou, H., Yuan, Y., and Shi, C. (2009). Object tracking using sift features and mean shift. *Computer Vision and Image Understanding*, 113(3):345–352.

[Zhu, 2005a] Zhu, X. (2005a). Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison.

[Zhu, 2005b] Zhu, X. (2005b). *Semi-supervised learning with graphs*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA.

[Zhu et al., 2003] Zhu, X., Ghahramani, Z., and Lafferty, J. (2003). Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*.

[Zhu et al., 2006] Zhu, X., Kandola, J., Lafferty, J., and Ghahramani, Z. (2006). Graph kernels by spectral transforms. *Semi-supervised learning*, pages 277–291.

[Zhu and Lafferty, 2005] Zhu, X. and Lafferty, J. D. (2005). Harmonic mixtures: combining mixture models and graph-based methods for inductive and scalable semi-supervised learning. In *ICML: Proceedings of the Twenty-Second International Conference on Machine Learning*, pages 1052–1059. ACM.

[Zwald and Blanchard, 2005] Zwald, L. and Blanchard, G. (2005). On the convergence of eigenspaces in kernel principal component analysis. In *NIPS*.