Large-Scale Machine Learning for Classification and Search

Wei Liu

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2012

ABSTRACT

Large-Scale Machine Learning for Classification and Search

Wei Liu

With the rapid development of the Internet, nowadays tremendous amounts of data including images and videos, up to millions or billions, can be collected for training machine learning models. Inspired by this trend, this thesis is dedicated to developing large-scale machine learning techniques for the purpose of making classification and nearest neighbor search practical on gigantic databases.

Our first approach is to explore data graphs to aid classification and nearest neighbor search. A graph offers an attractive way of representing data and discovering the essential information such as the neighborhood structure. However, both of the graph construction process and graph-based learning techniques become computationally prohibitive at a large scale. To this end, we present an efficient large graph construction approach and subsequently apply it to develop scalable semi-supervised learning and unsupervised hashing algorithms. Our unique contributions on the graph-related topics include:

1. Large Graph Construction: Conventional neighborhood graphs such as kNN graphs require a quadratic time complexity, which is inadequate for large-scale applications mentioned above. To overcome this bottleneck, we present a novel graph construction approach, called *Anchor Graphs*, which enjoys linear space and time complexities and can thus be constructed over gigantic databases efficiently. The central idea of the Anchor Graph is introducing a few anchor points and converting intensive data-to-data affinity computation to drastically reduced data-to-anchor affinity computation. A low-rank data-to-data affinity matrix is derived using the data-to-anchor affinity matrix. We also theoretically prove that the Anchor Graph lends itself to an intuitive probabilistic interpretation by showing

that each entry of the derived affinity matrix can be considered as a transition probability between two data points through Markov random walks.

2. Large-Scale Semi-Supervised Learning: We employ Anchor Graphs to develop a scalable solution for semi-supervised learning, which capitalizes on both labeled and unlabeled data to learn graph-based classification models. We propose several key methods to build scalable semi-supervised kernel machines such that real-world linearly inseparable data can be tackled. The proposed techniques take advantage of the Anchor Graph from a kernel point of view, generating a set of low-rank kernels which are made to encompass the neighborhood structure unveiled by the Anchor Graph. By linearizing these low-rank kernels, training nonlinear kernel machines in semi-supervised settings can be simplified to training linear SVMs in supervised settings, so the computational cost for classifier training is substantially reduced. We accomplish excellent classification performance by applying the proposed semi-supervised kernel machine - a linear SVM with a linearized Anchor Graph warped kernel.

3. Unsupervised Hashing: To achieve fast point-to-point search, compact hashing with short hash codes has been suggested, but how to learn codes such that good search performance is achieved remains a challenge. Moreover, in many cases real-world data sets are assumed to live on manifolds, which should be taken into account in order to capture meaningful nearest neighbors. To this end, we present a novel unsupervised hashing approach based on the Anchor Graph which captures the underlying manifold structure. The Anchor Graph Hashing approach allows constant time hashing of a new data point by extrapolating graph Laplacian eigenvectors to eigenfunctions. Furthermore, a hierarchical threshold learning procedure is devised to produce multiple hash bits for each eigenfunction, thus leading to higher search accuracy. As a result, Anchor Graph Hashing exhibits good search performance in finding semantically similar neighbors.

To address other practical application scenarios, we further develop advanced hashing techniques that incorporate supervised information or leverage unique formulations to cope with new forms of queries such as hyperplanes.

4. Supervised Hashing: Recent research has shown that the hashing quality could

be boosted by leveraging supervised information into hash function learning. However, the existing methods either lack adequate performance or often incur cumbersome model training. To this end, we present a novel kernel-based supervised hashing model which requires a limited amount of supervised information in the form of similar and dissimilar data pairs, and is able to achieve high hashing quality at a practically feasible training cost. The idea is to map the data to compact binary codes whose Hamming distances are simultaneously minimized on similar pairs and maximized on dissimilar pairs. Our approach is distinct from prior work in utilizing the equivalence between optimizing the code inner products and the Hamming distances. This enables us to sequentially and efficiently train the hash functions one bit at a time, yielding very short yet discriminative codes. The presented supervised hashing approach is general, allowing search of both semantically similar neighbors and metric distance neighbors.

5. Hyperplane Hashing: Hyperplane hashing aims at rapidly searching the database points near a given hyperplane query, and has shown practical impact on large-scale active learning with SVMs. The existing hyperplane hashing methods are randomized and need long hash codes to achieve reasonable search accuracy, thus resulting in reduced search speed and large memory overhead. We present a novel hyperplane hashing technique which yields high search accuracy with compact hash codes. The key idea is a novel bilinear form used in designing the hash functions, leading to a higher collision probability than all of the existing hyperplane hash functions when using random projections. To further increase the performance, we develop a learning based framework in which the bilinear functions are directly learned from the input data. This results in compact yet discriminative codes, as demonstrated by the superior search performance over all random projection based solutions.

We divide the thesis into two parts: scalable classification with graphs (topics 1 and 2 mentioned above), and nearest neighbor search with hashing (topics 3, 4 and 5 mentioned above). The two parts are connected in the sense that the idea of Anchor Graphs in Part I enables not only scalable classification but also unsupervised hashing, and hyperplane hashing in Part II can directly benefit classification under an active learning framework. All

of the machine learning techniques developed in this thesis emphasize and pursue excellent performance in both speed and accuracy, which are verified through extensive experiments carried out on various large-scale tasks of classification and search. The addressed problems, classification and nearest neighbor search, are fundamental for many real-world applications. Therefore, we expect that the proposed solutions based on graphs and hashing will have a tremendous impact on a great number of realistic large-scale applications.

Table of Contents

1 Introduction **1**

1.1 Motivations 1

1.2 Our Techniques 4

1.3 Thesis Overview 7

I Scalable Classification with Graphs **9**

2 Large Graph Construction – Anchor Graphs **10**

2.1 Problem Background 11

2.2 Related Work 15

2.3 Notations 18

2.4 Prerequisites 18

2.4.1 Anchor-Based Label Prediction 18

2.4.2 Adjacency Matrix Design 21

2.4.3 Design Principles 22

2.5 Anchor Graphs 23

2.5.1 Design of Z 23

2.5.2 Design of W 24

2.5.3 Connection to Low-Rank Matrix Approximation 30

2.6 Anchor Graph Regularization 31

2.6.1 Analysis 32

2.7 Experiments 34

2.7.1 Toy Datasets . . . 35
2.7.2 Real-World Datasets . . . 35
2.8 Summary and Discussion . . . 40

3 Large-Scale Semi-Supervised Learning **49**
3.1 Problem Background . . . 50
3.2 Related Work . . . 52
3.3 Notations . . . 55
3.4 Kernel Linearization . . . 57
3.5 Generating Low-Rank Kernels . . . 58
3.5.1 Anchor Graph Kernels . . . 59
3.5.2 Anchor Graph Laplacian Kernels . . . 60
3.5.3 Anchor Graph Warped Kernels . . . 62
3.6 Experiments . . . 65
3.6.1 CIFAR-10 . . . 66
3.6.2 MNIST . . . 70
3.6.3 Extended MNIST . . . 71
3.7 Summary and Discussion . . . 71

II Nearest Neighbor Search with Hashing **74**

4 Unsupervised Hashing **75**
4.1 Problem Background . . . 76
4.2 Related Work . . . 80
4.3 Notations . . . 82
4.4 Anchor Graph Hashing (AGH) . . . 84
4.4.1 Preview of AGH . . . 84
4.4.2 Formulation . . . 86
4.4.3 Anchor Graphs . . . 87
4.4.4 Eigenfunction Generalization . . . 89
4.4.5 Hierarchical Hashing . . . 93

4.4.6 Complexity Analysis . . . 96
4.5 Experiments . . . 96
4.5.1 Methods and Evaluation Protocols . . . 96
4.5.2 Datasets . . . 97
4.5.3 Results . . . 98
4.6 Summary and Discussion . . . 103

5 Supervised Hashing **105**
5.1 Problem Background . . . 106
5.2 Related Work . . . 108
5.3 Notations . . . 111
5.4 Kernel-Based Supervised Hashing (KSH) . . . 111
5.4.1 Hash Functions with Kernels . . . 111
5.4.2 Manipulating Code Inner Products . . . 115
5.4.3 Greedy Optimization . . . 118
5.4.4 Analysis . . . 121
5.5 Experiments . . . 124
5.5.1 CIFAR-10 . . . 125
5.5.2 Tiny-1M . . . 131
5.6 Summary and Discussion . . . 139

6 Hyperplane Hashing **141**
6.1 Problem Background . . . 142
6.2 Related Work . . . 144
6.3 Notations . . . 146
6.4 Point-to-Hyperplane Search . . . 146
6.5 Randomized Hyperplane Hashing . . . 150
6.5.1 Background – Linear Hash Functions . . . 150
6.5.2 Bilinear Hash Functions . . . 151
6.5.3 Theoretic Analysis . . . 155
6.6 Compact Hyperplane Hashing . . . 161

6.6.1 Motivations 161
6.6.2 Learning Bilinear Hash Functions 162
6.7 Experiments 167
6.7.1 Datasets 167
6.7.2 Evaluations and Results 168
6.8 Summary and Discussion 175

III Conclusions **177**

7 Conclusions **178**
7.1 Summary of Contributions 178
7.2 Future Work 180

IV Bibliography **183**

Bibliography **184**

List of Figures

1.1 Data graphs discover manifolds. (a) Swiss roll toy data; (b) a 10NN graph. 2

1.2 Linear vs. nonlinear hash function. (a) A standard linear hash function cuts off the two manifolds. (b) A nonlinear hash function adaptively yields hash bits along the two manifolds. 3

1.3 A desirable hash function subject to supervised information. 4

1.4 Two distinct nearest neighbor search problems. (a) Point-to-point search; (b) point-to-hyperplane search. 5

1.5 Design the data-to-data affinity matrix W using the data-to-anchor affinity matrix Z. 6

2.1 An example of a graph $\mathcal{G}$ with only five nodes. W is the associated weighted adjacency matrix, $D = \text{diag}(W\mathbf{1})$ is the node-degree matrix, and Y is the label matrix. 14

2.2 A graph-based semi-supervised learning task. 15

2.3 Design the data-to-data affinity matrix W using the data-to-anchor affinity matrix Z. The circles represent data points $\boldsymbol{x}_i$'s, and the solid circles represent anchor points $\boldsymbol{u}_k$'s. 27

2.4 Random walks over a data-anchor bipartite graph. (a) The data-anchor bipartite graph $\mathcal{B}(\mathcal{X}, \mathcal{U}, \mathcal{E}, Z)$; (b) $path\ 1 = p^{(1)}(\boldsymbol{u}_1|\boldsymbol{x}_2)p^{(1)}(\boldsymbol{x}_3|\boldsymbol{u}_1)$ and $path\ 2 = p^{(1)}(\boldsymbol{u}_2|\boldsymbol{x}_2)p^{(1)}(\boldsymbol{x}_3|\boldsymbol{u}_2)$; (c) $W_{23} = p^{(2)}(\boldsymbol{x}_3|\boldsymbol{x}_2) = path\ 1 + path\ 2$. 28

2.5 The two-moon toy data consisting of 1200 2D points. (a) 100 anchor points from K-means clustering centers; (b) a 10NN graph built on original points; (c) the proposed Anchor Graph with $m = 100, s = 2$ built on original points. 30

2.6 Results on the two-ring toy dataset. (a) The original data points; (b) a 10NN graph; (c) 100 anchor points; (d) an Anchor Graph with $m = 100, s = 2$; (e) 200 anchor points; (f) an Anchor Graph with $m = 200, s = 2$. 36

2.7 Results on the two-sun toy dataset. (a) The original data points; (b) a 10NN graph; (c) 100 anchor points; (d) an Anchor Graph with $m = 100, s = 2$; (e) 200 anchor points; (f) an Anchor Graph with $m = 200, s = 2$. 37

2.8 Example images from the **COIL-20** dataset. The original figure is from `http://www1.cs.columbia.edu/CAVE/software/softlib/coil-20.php`. 40

2.9 Example images from the **SCENE-15** dataset. 41

2.10 Example images from the **USPS** dataset. The original figure is from `http://www.cad.zju.edu.cn/home/dengcai/Data/USPS/images.html`. 41

2.11 Results on **COIL-20**. (a)(b)(c)(d) Classification error rates with increasing s under different l, m settings; (e)(f) error rates with increasing m under $s = 10$. 43

2.12 Results on **SCENE-15**. (a)(b)(c)(d) Classification accuracy with increasing s under different l, m settings; (e)(f) accuracy with increasing m under $s = 3$. 45

2.13 Results on **USPS**. (a)(b)(c)(d) Classification error rates with increasing s under different l, m settings; (e)(f) error rates with increasing m under $s = 3$. 47

3.1 Example images from the **CIFAR-10** dataset. 70

3.2 Example images from the **MNIST** dataset. The original figure is from `http://www.cad.zju.edu.cn/home/dengcai/Data/MNIST/images.html`. 71

4.1 A nearest neighbor search example in image retrieval. The query image is a butterfly image, and the most visually relevant images, i.e., images containing at least a butterfly, are expected to be retrieved from the given image database. 76

4.2 Schematic diagrams for a tree and a hash table. $\boldsymbol{q}$ represents a query point. (a) A binary space partitioning tree (e.g., kd-tree) where each node contains a database point. (b) A linear hash table in which the binary codes of database points are treated as addresses (or indices) into the hash table and every address points to a hash bucket that stores corresponding database points. 78

4.3 Linear vs. nonlinear hash function. (a) A standard linear hash function $h(\boldsymbol{x}) = \text{sgn}(\boldsymbol{w}^\top \boldsymbol{x} + b)$ used by many hashing algorithms, which cuts off the two manifolds. (b) The nonlinear hash function $h(\boldsymbol{x}) = \text{sgn}(\phi(\boldsymbol{x}))$ used by our proposed AGH, which adaptively yields hash bits along the two manifolds. 85

4.4 Two-bit Hamming embeddings of two-moon toy data consisting of 12,000 2D points. The trivial eigenvector **1** has been excluded in each embedding. The blue color denotes '1' bits (positive eigenvector entries) and the red color denotes '-1' bits (negative eigenvector entries). (a) The Hamming embedding of Idealized Graph Hashing, (b) the Hamming embedding of Anchor Graph Hashing, and (c) the Hamming embedding of Spectral Hashing. 90

4.5 Hierarchical hashing on a data graph. $x_1, \cdots, x_8$ are data points and $\boldsymbol{y}$ is a graph Laplacian eigenvector. The data points of filled circles take '1' hash bit and the others take '-1' hash bit. The entries with dark color in $\boldsymbol{y}$ are positive and the others are negative. (a) The first-layer hash function h^1 uses threshold 0 ; (b) the second-layer hash functions h^2 use thresholds b^+ and b^-. 93

4.6 Example images from the **NUS-WIDE** dataset, every two of which respectively come from 12 most frequent tags: 'animal', 'buildings', 'clouds', 'grass', 'person', 'plants', 'sky', 'water', 'window', 'lake', 'ocean' and 'road'. 98

4.7 Hash lookup results on **MNIST**. (a) Mean precision of Hamming radius 2 hash lookup using the varying number of hash bits (r); (b) hash lookup success rate using the varying number of hash bits (r). 101

4.8 Hamming ranking results on **MNIST**. (a) Mean precision of top-5000 ranked neighbors using the varying number of anchors (m); (b) mean precision curves of Hamming ranking; (c) mean recall curves of Hamming ranking. 101

4.9 Hash lookup results on **NUS-WIDE**. (a) Mean precision of Hamming radius 2 hash lookup using the varying number of hash bits (r); (b) hash lookup success rate using the varying number of hash bits (r). 102

4.10 Hamming ranking results on **NUS-WIDE**. (a) Mean precision of top-5000 ranked neighbors using the varying number of anchors (m); (b) mean precision curves of Hamming ranking; (c) mean recall curves of Hamming ranking. 102

5.1 Two types of supervised information. A blue solid line denotes a similar relation, and a red dashed line denotes a dissimilar relation. (a) Semantic supervision: 'similar' represents that two images belong to the same object category, and 'dissimilar' represents that two images come from different categories. (b) Metric supervision: 'similar' represents that two samples hold a sufficiently short metric distance, and 'dissimilar' represents that two samples incur a sufficiently long metric distance. 107

5.2 A desirable hash function subject to supervised information. 108

5.3 The core idea of our proposed supervised hashing. (a) Three data points with supervised pairwise labels: "1" (similar) and "-1" (dissimilar); (b) optimization on Hamming distances; (c) optimization on code inner products (r is the bit number). The latter two are equivalent due to the one-to-one correspondence between a Hamming distance and a code inner product. . . 116

5.4 The sign function $\mathrm{sgn}(x)$ and the sigmoid-shaped function $\varphi(x) = 2/(1 + \exp(-x)) - 1$. 120

5.5 The hash lookup results on **CIFAR-10**. Six (semi-)supervised hashing methods use 1K labeled examples. (a)(b) Mean precision of Hamming radius 2 hash lookup; (c)(d) hash lookup success rate. 128

5.6 Mean precision-recall curves of Hamming ranking with 48 bits on **CIFAR-10**. 129

5.7 The results on **CIFAR-10**. Six (semi-)supervised hashing methods use 2K labeled examples. (a) Mean precision of Hamming radius 2 hash lookup; (b) hash lookup success rate; (c) mean precision-recall curves of Hamming ranking with 48 bits. 130
5.8 100 query images from the **Tiny-1M** dataset. 131
5.9 The hash lookup results on **Tiny-1M**. Six (semi-)supervised hashing methods use 5,000 pseudo-labeled examples. (a)(b) Mean precision of Hamming radius 2 hash lookup; (c)(d) hash lookup success rate. 134
5.10 The Hamming ranking results on **Tiny-1M**. Six (semi-)supervised hashing methods use 5,000 pseudo-labeled examples. (a)(b) Mean precision curves of Hamming ranking with 48 bits; (c)(d) mean recall curves of Hamming ranking with 48 bits. 135
5.11 The results on **Tiny-1M**. Six (semi-)supervised hashing methods use 10,000 pseudo-labeled examples. (a) Mean precision of Hamming radius 2 hash lookup; (b) hash lookup success rate; (c) mean precision curves of Hamming ranking with 48 bits; (d) mean recall curves of Hamming ranking with 48 bits. 136
5.12 The average quadratic residues $R = \mathcal{Q}/l^2 = \|B_l B_l^\top/r - S\|/l^2$ of reconstructing the pairwise label matrix S with the r-bit codes B_l learned by KSH^0 and KSH, respectively. (a) R of reconstructing 5,000×5,000 S with 5,000 binary codes; (b) R of reconstructing 10,000×10,000 S with 10,000 binary codes. . 137
5.13 Top six neighbors of four query images, returned by SSH, BRE, MLH and KSH using 5,000 pseudo-labeled training images and 48 binary bits on the **Tiny-1M** dataset. 138

6.1 Two distinct nearest neighbor search problems. (a) Point-to-point search, the blue solid circle represents a point query and the red circle represents the found nearest neighbor point. (b) Point-to-hyperplane search, the blue plane denotes a hyperplane query $\mathcal{P}_{\boldsymbol{w}}$ with $\boldsymbol{w}$ being its normal vector, and the red circle denotes the found nearest neighbor point. 143

6.2 The point-to-hyperplane search problem encountered in SVM active learning. $\mathcal{P}_{\boldsymbol{w}}$ is the SVM's hyperplane decision boundary, $\boldsymbol{w}$ is the normal vector to $\mathcal{P}_{\boldsymbol{w}}$, and $\boldsymbol{x}$ is a data vector. (a) Point-to-hyperplane distance $D(\boldsymbol{x}, \mathcal{P}_{\boldsymbol{w}})$ and point-to-hyperplane angle $\alpha_{\boldsymbol{x},\boldsymbol{w}}$. (b) Informative $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ and uninformative $(\boldsymbol{x}_3, \boldsymbol{x}_4)$ samples for the margin-based active sample selection criterion. The ideal neighbors of $\mathcal{P}_{\boldsymbol{w}}$ are the points $\perp \boldsymbol{w}$. 149

6.3 The single-bit outputs of the proposed bilinear hash function $h^{\mathcal{B}}$. Data vector $\boldsymbol{x}_1$ is nearly parallel to the normal vector $\boldsymbol{w}$, data vector $\boldsymbol{x}_2$ is closely $\perp \boldsymbol{w}$, and data vector $\boldsymbol{x}^* \perp \boldsymbol{w}$ is the ideal neighbor for the goal of angle-based hyperplane hashing. The blue "// bin" is useless but the red "$\perp$" bin is imperative. 152

6.4 Enforcing multiple hash bits to refine the near neighbor region for point-to-hyperplane search. $(\boldsymbol{u}_1, \boldsymbol{v}_1)$ and $(\boldsymbol{u}_2, \boldsymbol{v}_2)$ are projection pairs used by two bilinear hash functions $h_1^{\mathcal{B}}$ and $h_2^{\mathcal{B}}$, respectively. 153

6.5 Theoretical comparison of three randomized hash schemes. (a) p_1 (probability of collision) vs. r (squared point-to-hyperplane angle); (b) ρ (query time exponent) vs. r for $\epsilon = 3$. 160

6.6 Learning the bilinear projections $(\boldsymbol{u}, \boldsymbol{v})$ such that the resulting hash function $h(\boldsymbol{x}) = \text{sgn}(\boldsymbol{u}^\top \boldsymbol{x} \boldsymbol{x}^\top \boldsymbol{v})$ yields the same bit for nearly // inputs whereas different bits for nearly $\perp$ inputs. 163

6.7 Results on **20 Newsgroups** using 16 hash bits. (a) Learning curves of MAP averaged over 20 classes and 5 runs; (b) minimum-margin curves of active sample selection averaged over 20 classes and 5 runs; (c) the number of queries (≤ 300) receiving nonempty hash lookups across 20 classes averaged over 5 runs; and (d) the number of database points visited in the found hash buckets per query averaged over 5 runs. 169

6.8 Results on **Tiny-1M** using 20 hash bits. (a) Learning curves of MAP averaged over 10 classes and 5 runs; (b) minimum-margin curves of active sample selection averaged over 10 classes and 5 runs; (c) the number of queries (≤ 300) receiving nonempty hash lookups across 10 classes averaged over 5 runs; and (d) the number of database points visited in the found hash buckets per query averaged over 5 runs. 170

List of Tables

2.1 Summary of the properties of efficient large graphs. d is the data dimension and n is the data size. Denote by $L \succeq 0$ positive semidefinite L. 18

2.2 Table of notations. 19

2.3 Table of notations (continued). 20

2.4 Time complexity analysis of the developed scalable GSSL algorithm Anchor Graph Regularization (AGR). n is the data size, m is the number of anchor points, d is the data dimension, s is the number of nearest anchors in LAE, T is the number of iterations in K-means clustering, and T' is the number of iterations in LAE ($n \gg m \gg s$). 33

2.5 Comparison of three classical GSSL algorithms with AGR. LGC, GFHF and GR use the same kNN graph, while AGR uses the Anchor Graph (m anchors). $F \in \mathbb{R}^{n \times c}$ expresses the target soft label matrix associated with all n data points and c classes, L and $\bar{L}$ respectively denote the graph Laplacian and normalized graph Laplacian matrices of the kNN graph, and $\tilde{L}$ represents the reduced Laplacian matrix of the Anchor Graph. 34

2.6 Classification performance on **COIL-20** (1,440 samples). l denotes the number of labeled examples for training GSSL algorithms. All running time is recorded in second. The K-means clustering time for 100 and 200 anchors is 0.049 and 0.083 seconds, respectively. AGR-related methods set $s = 10$. At a fixed l, two lowest error rate values are displayed in boldface type. 42

2.7 Classification performance on **SCENE-15** (4,485 samples). l denotes the number of labeled examples for training GSSL algorithms. All running time is recorded in second. The K-means clustering time for 500 and 1000 anchors is 0.465 and 0.815 seconds, respectively. AGR-related methods set $s = 3$. At a fixed l, two highest accuracy values are displayed in boldface type. . . . 44

2.8 Classification performance on **USPS** (9,298 samples). l denotes the number of labeled examples for training GSSL algorithms. All running time is recorded in second. The K-means clustering time for 500 and 1000 anchors is 1.49 and 2.76 seconds, respectively. AGR-related methods set $s = 3$. At a fixed l, two lowest error rate values are displayed in boldface type. 46

3.1 Summary of anchor-based GSSL models which share the same form of classification functions $f(\boldsymbol{x}) = \sum_{k=1}^{m} Z_{xk} a_k$. Z_{xk} denotes the affinity between data point $\boldsymbol{x}$ and anchor $\boldsymbol{u}_k$, $S(\cdot)$ is some similarity function, $\kappa(\cdot)$ is a kernel function, and $p(\mathcal{C}_k|\boldsymbol{x})$ is the posterior probability of $\boldsymbol{x}$ assigned to cluster $\mathcal{C}_k$. 55

3.2 Table of notations. 56

3.3 Table of notations (continued). 57

3.4 Summary of three low-rank kernels generated from and with Anchor Graphs. 65

3.5 Classification accuracy (%) on **CIFAR-10** (60,000 samples). 67

3.6 Classification error rates (%) on **MNIST** (70,000 samples). 68

3.7 Classification error rates (%) on **Extended MNIST** (1,030,000 samples). 69

4.1 Compact hashing mode: hash table lookup times within Hamming radius 2. 80

4.2 Table of notations. 83

4.3 Table of notations (continued). 84

4.4 Hamming ranking performance on **MNIST**. Mean Average Precision (MAP), training time, and test time are evaluated for each hashing method. r denotes the number of hash bits used in hashing algorithms, and also the number of eigenfunctions used in SE ℓ_2 linear scan. The K-means execution time is 20.1 seconds for training AGH on **MNIST**. All training and test time is recorded in second. At a fixed bit number, two highest MAP values achieved by hashing are displayed in boldface type. 99

4.5 Hamming ranking performance on **NUS-WIDE**. Mean Precision (MP) of top-5000 ranked neighbors, training time, and test time are evaluated for each hashing method. r denotes the number of hash bits used in hashing algorithms, and also the number of eigenfunctions used in SE ℓ_2 linear scan. The K-means execution time is 105.5 seconds for training AGH on **NUS-WIDE**. All training and test time is recorded in second. At a fixed bit number, two highest MP values achieved by hashing are displayed in boldface type. 100

5.1 The basic properties of various hashing methods. $\boldsymbol{x}$ denotes a data vector, $\boldsymbol{k}(\boldsymbol{x})$ denotes a kernel mapping, $\boldsymbol{z}(\boldsymbol{x})$ denotes the data-to-anchor mapping defined in Chapter 4, $\boldsymbol{w}$ denotes a projection vector, k denotes an integer, and b, t denote shift scalars. 112

5.2 Table of notations. 113

5.3 Table of notations (continued). 114

5.4 The formulations of supervised hashing methods. $Loss_\rho(x, s) = \max(s(x - \rho) + 1, 0)$ (ρ is a hyperparameter) is a hinge loss function used in MLH. . . 119

5.5 Hamming ranking performance on **CIFAR-10** (60K). l denotes the number of labeled examples for training (semi-)supervised hashing methods. Six unsupervised methods LSH, PCAH, SH, KLSH, 1-AGH and 2-AGH do not use any labels. All training and test time is recorded in second. At a fixed bit number, two highest MAP values achieved by hashing are displayed in boldface type. 126

5.6 Hamming ranking performance on **CIFAR-10** (60K). l denotes the number of labeled examples for training (semi-)supervised hashing methods. All training and test time is recorded in second. At a fixed bit number, two highest MAP values achieved by hashing are displayed in boldface type. . 127

5.7 Comparison with SVM hashing on **CIFAR-10**. MAP of Hamming ranking is reported for SVM hashing, KSH^0 and KSH using 10 (the number of classes) hash bits. 127

5.8 Hamming ranking performance on **Tiny-1M**. l denotes the number of pseudo-labeled examples for training (semi-)supervised hashing methods. Six unsupervised methods LSH, PCAH, SH, KLSH, 1-AGH and 2-AGH do not use any labels. All training and test time is recorded in second. At a fixed bit number, two highest MP values achieved by hashing are displayed in boldface type. 132

5.9 Hamming ranking performance on **Tiny-1M**. l denotes the number of pseudo-labeled examples for training (semi-)supervised hashing methods. All training and test time is recorded in second. At a fixed bit number, two highest MP values achieved by hashing are displayed in boldface type. 133

6.1 Table of notations. 147

6.2 Table of notations (continued). 148

6.3 Several key properties of three randomized hyperplane hashing methods AH-Hash, EH-Hash and BH-Hash. r denotes the squared point-to-hyperplane angle $\alpha^2_{\boldsymbol{x},\boldsymbol{w}}$, k denotes the number of hash bits used in a single hash table, and d denotes the data dimension. "Hashing Time" refers to the time of evaluating k hash functions to hash an input (database point or query) into a single hash table. 161

6.4 Results on **20 Newsgroups** using 8 and 12 hash bits. All preprocessing and search time is recorded in second. At a fixed bit number, two highest MAP values achieved by hashing are displayed in boldface type. 172

6.5 Results on **20 Newsgroups** using 16 hash bits. All preprocessing and search time is recorded in second. Two highest MAP values achieved by hashing are displayed in boldface type. 173
6.6 Results on **Tiny-1M** using 8 and 12 hash bits. All preprocessing and search time is recorded in second. At a fixed bit number, two highest MAP values achieved by hashing are displayed in boldface type. 174
6.7 Results on **Tiny-1M** using 16 and 20 hash bits. All preprocessing and search time is recorded in second. At a fixed bit number, two highest MAP values achieved by hashing are displayed in boldface type. 175

Acknowledgments

Above all, I would express my supreme gratitude to my supervisor, Professor Shih-Fu Chang, for his great support in the past five years. It is Prof. Chang who directs me into the challenging fields of machine learning, computer vision, and pattern recognition. He always tends first to my needs in research. His insightful guidance and suggestions motivate me towards a professional researcher.

I would also like to thank Prof. Jiebo Luo and Prof. Gang Hua. They were my mentors when I conducted internship in 2010 summer and 2011 summer, respectively. Both of them spent a lot of time in research discussions with me and offered me a great deal of assistance in patent application and paper revision. I am also grateful to Prof. Lexing Xie and Dr. Shahram Ebadollahi for their help with my TA job of the Spring 2008 course, Digital Image Processing.

Most notably, Prof. Yu-Gang Jiang is a very good roommate for me. His advice on research routes and his collaboration in our joint papers impress the value of friendship on my memory during the past five years since I came to the United States. My thanks also go to my colleagues in Columbia University, including Dr. Lyndon Kennedy, Dr. Eric Zavesky, Dr. Wei Jiang, Dr. Jun Wang, Dr. Shiqian Ma, Dr. Qi Wu, Dr. Zhenguo Li, Dr. Rongrong Ji, Dr. Dong Liu, Dr. Yadong Mu, Peng Liu, Weiwei Guo, Hao Dang, Ju Sun, Guangnan Ye, Yan Wang, Felix Xinnan Yu, Xiao-Ming Wu, Hongzhi Li, and Brendan Jou. I learned a lot in many aspects from them during these years so that I would never forget the beautiful and joyful days spent with them. Besides, great appreciations to Prof. Zhouyu Fu, Dr. Fuxin Li, Dr. Liefeng Bo, and Deli Zhao for their advice on research methodologies and proofreading on my papers.

My special appreciation is dedicated to my parents with whom I share my success and frustration and never feel alone. Hardly can I attain my goals without their encouragement.

They provide me power each time I am confronted with difficulties. Most of my inspiration in computer vision and machine learning stems from them. Finally, I would like to thank all committee members (Prof. Shih-Fu Chang, Prof. Dan Ellis, Prof. John Wright, Dr. Sanjiv Kumar, and Dr. Rong Yan) for attending my PhD thesis defense, all staffs in the Department of Electrical Engineering at Columbia University and Multimedia Group at IBM Thomas J. Watson Research Center, and all other people who discussed the thesis with me.

This work is dedicated to my parents
who love, encourage, and urge me all along.

Chapter 1

Introduction

This thesis is dedicated to developing large-scale machine learning techniques for the purpose of making classification and nearest neighbor search practical on gigantic databases.

1.1 Motivations

In the current era of data divulgence, there is emerging attention in leveraging massive amounts of data available in open sources such as the Web to help solve long standing computer vision, data mining, and information retrieval problems. Then, how to effectively incorporate and efficiently exploit large-scale data corpora is an open problem. In this thesis, we try to deliver effective machine learning models for disposing of and searching into large-scale data collections.

Above all, we explore graph representation to load massive data samples. A data graph offers a very attractive way of representing data and discovering the underlying information, e.g., the neighborhood structure, essential for the data. It is very intuitive that neighborhood graphs [33] can reveal the manifolds that underlie the input data. Figure 1.1 showcases a kNN graph which discovers the underlying data manifold existing in the Swiss roll toy data [144]. In the literature, neighborhood graphs have demonstrated wide usage in a great number of problems including classification, clustering, regression, dimensionality reduction, relevance ranking, and so on. However, scalability of data graphs to the explosive growth of data sets remains challenging. For example, the time complexity for constructing

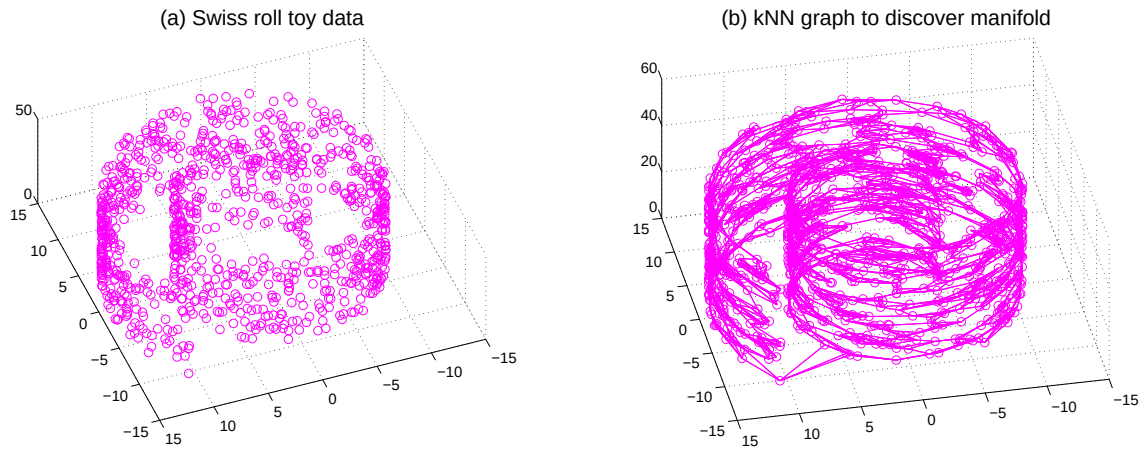


Figure 1.1: Data graphs discover manifolds. (a) Swiss roll toy data; (b) a 10NN graph.

a neighborhood graph is quadratic in the database size, which is computationally infeasible for practical large-scale data sets consisting of millions up to billions of samples.

Semi-supervised learning (SSL) [23][211][213] has significant impact on pervasive applications of machine learning and pattern recognition. In these practical applications, one frequently encounters the situations where only a few labeled data are available and large amounts of data remain unlabeled. The labeled data often suffer from difficult and expensive acquisition, whereas the unlabeled data can be cheaply and automatically gathered. Among the vast number of SSL methods, graph-based semi-supervised learning (GSSL) is substantially appealing because it is easy to implement and generally renders closed-form solutions. With rapid development of the Internet, now we can gather massive unlabeled data, and then the need for large scale SSL arises. Despite appealing advantages of GSSL in practical applications, most GSSL methods scale poorly with the data size because of the computationally expensive steps stemming from large graph construction and classifier training over graphs.

Hashing is becoming increasingly popular for time-efficient nearest neighbor search in massive databases, because it removes the curse of dimensionality that traditional tree-based indexing methods suffer from. However, learning compact hash codes that yield good search performance is still a challenge. Moreover, in many cases real-world data nearly live on low-dimensional manifolds, which should be taken into account to capture meaningful

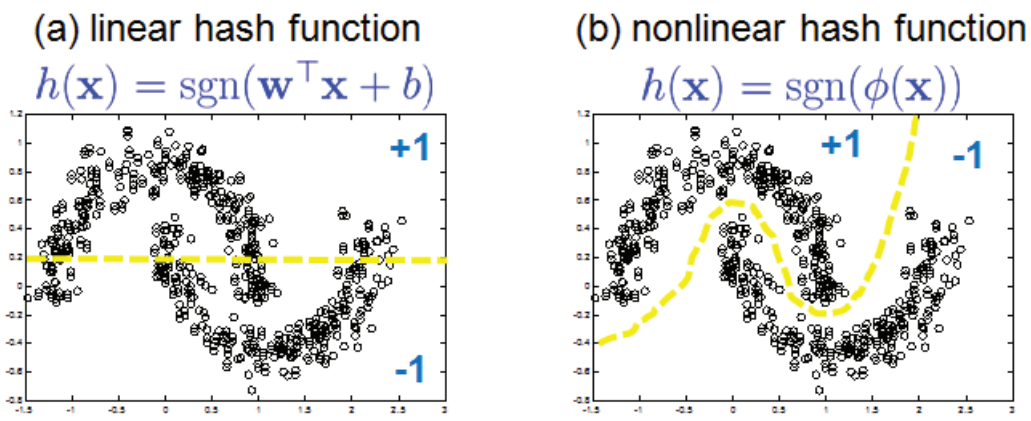


Figure 1.2: Linear vs. nonlinear hash function. (a) A standard linear hash function cuts off the two manifolds. (b) A nonlinear hash function adaptively yields hash bits along the two manifolds.

nearest neighbors in the code learning process. As shown in Figure 1.2, the desired manifold-motivated hash function should be in a nonlinear form ($\phi(\boldsymbol{x})$ is a nonlinear function) so that hash bits can be continuously yielded along data manifolds.

Recent years have witnessed the growing popularity of unsupervised hashing in large-scale machine learning, computer vision, and information retrieval problems. Nevertheless, unsupervised hashing usually achieves limited search accuracy. To this end, there has been some recent research [93][137] which shows that the hashing quality could be boosted by leveraging supervised information into hash function learning. Such supervised information is customarily expressed as similar and dissimilar data pairs. Figure 1.3 discloses that a desirable hash function should be consistent with the supervised information. While supervised hashing makes sense, the existing supervised hashing methods either lack adequate performance or often incur cumbersome model training.

In margin-based active learning, the data point nearest to the current decision boundary is chosen to request its label, which inspires a "point-to-hyperplane" search problem distinct from the conventional point-to-point search problem. Figure 1.4 illustrates the two problems in contrast. To speed up the point-to-hyperplane search process, a new hashing scenario, hyperplane hashing, arises. Such a new scenario requires us to rationally hash

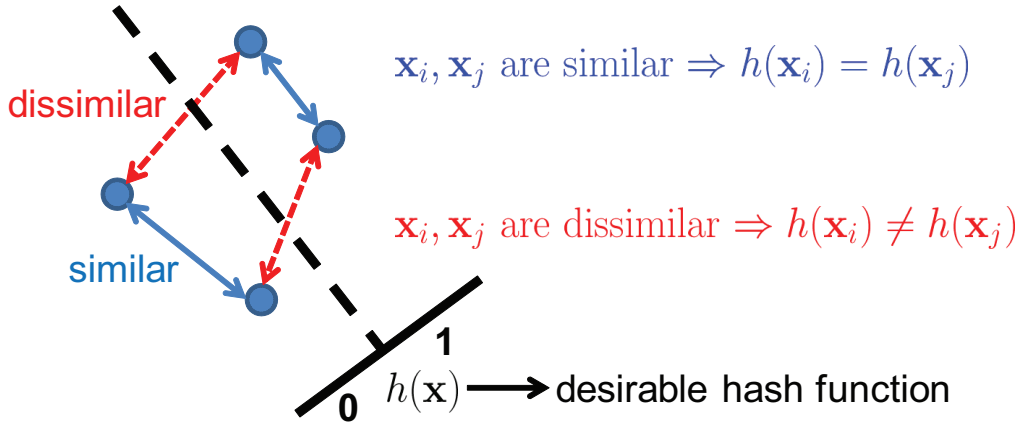


Figure 1.3: A desirable hash function subject to supervised information.

a hyperplane query to near database points, which is not easy to fulfill because point-to-hyperplane distances are quite different from routine point-to-point distances in terms of the computation mechanism. Despite the bulk of nearest neighbor search and hashing literature, this special hashing paradigm is rarely touched.

1.2 Our Techniques

First, we address the scalability issue of large graph construction in Chapter 2. Typically, provided with a data set $\mathcal{X} = \{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^n$, the widely used kNN graph needs the $O(dn^2)$ time complexity that is computationally infeasible for practical million-scale data. To overcome this computational bottleneck, we present a novel graph model *Anchor Graph* which has linear space and time complexities $O(n)$ and can thus be constructed over gigantic data collections efficiently. As shown in Figure 1.5, the central idea of the Anchor Graph is introducing a few anchor points and converting intensive data-to-data affinity computation to drastically reduced data-to-anchor affinity computation. Concretely, m anchor points $\{\boldsymbol{u}_k \in \mathbb{R}^d\}_{k=1}^m$ are used to form a highly sparse data-to-anchor affinity matrix $Z \in \mathbb{R}^{n \times m}$ in which $Z_{ij} > 0$ if and only if anchor $\boldsymbol{u}_k$ is among s ($\ll m$) nearest anchors of data point $\boldsymbol{x}_i$. Then, the data-to-data affinity matrix of the Anchor Graph is $W = Z\Lambda^{-1}Z^\top$ ($Z\mathbf{1} = \mathbf{1}$ and $\Lambda = \text{diag}(\mathbf{1}^\top Z)$), which is derived from Markov random walks between data points and anchors. The nonnegative, sparse, and low-rank characteristics of the affinity (or

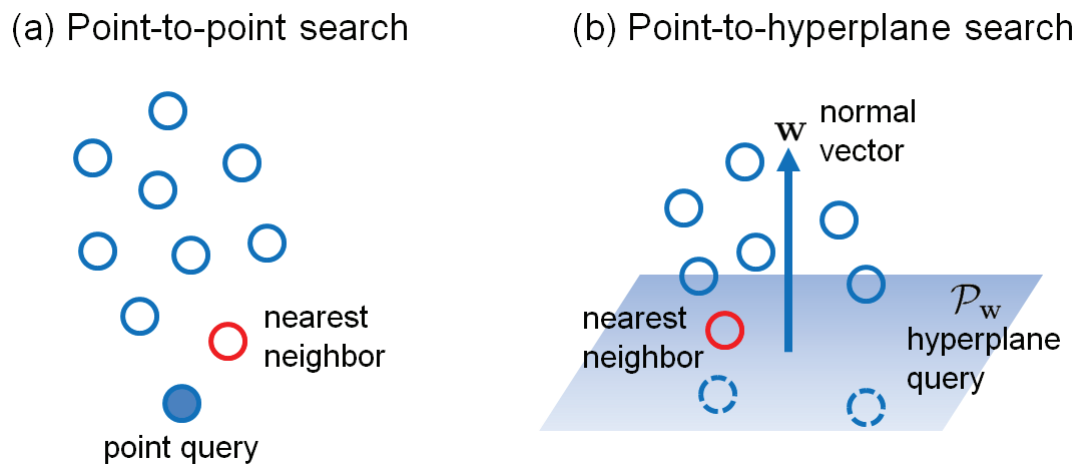


Figure 1.4: Two distinct nearest neighbor search problems. (a) Point-to-point search; (b) point-to-hyperplane search.

adjacency) matrices W yielded by Anchor Graphs are substantially critical, which ensure positive semidefinite graph Laplacians and allow efficient eigenfunction extensions of graph Laplacians. The memory cost of an Anchor Graph is $O(sn)$ for storing Z, and the time cost is $O(dmn)$. Since $m \ll n$, the time cost for constructing an Anchor Graph is linear in the data size n. Our experiments demonstrate that Anchor Graphs exhibit high fidelity to conventional kNN graphs yet with much shorter construction time.

Second, we address the scalability issue plaguing graph-based semi-supervised learning in Chapter 3. Our purpose is to develop nonlinear and discriminative semi-supervised classifiers from a kernel view. We realize that the classical GSSL algorithms [212][208] perform label propagation over neighborhood graphs without optimizing the margins in between different classes. To this end, we aim at learning SVM-like classifiers to maximize the margins. The central idea is to generate a low-rank kernel by leveraging an Anchor Graph into a kernel machine framework. In doing so, the large-scale semi-supervised learning task on all data samples is reduced to a supervised linear classification task carried out on much fewer labeled samples. Therefore, we eventually apply a linear SVM over a new feature space which is explicitly derived from decomposing the low-rank kernel. The generated low-rank kernel and its direct linearization succeed in addressing the scalability issue of GSSL. Extensive semi-supervised classification experiments performed on several large datasets of up

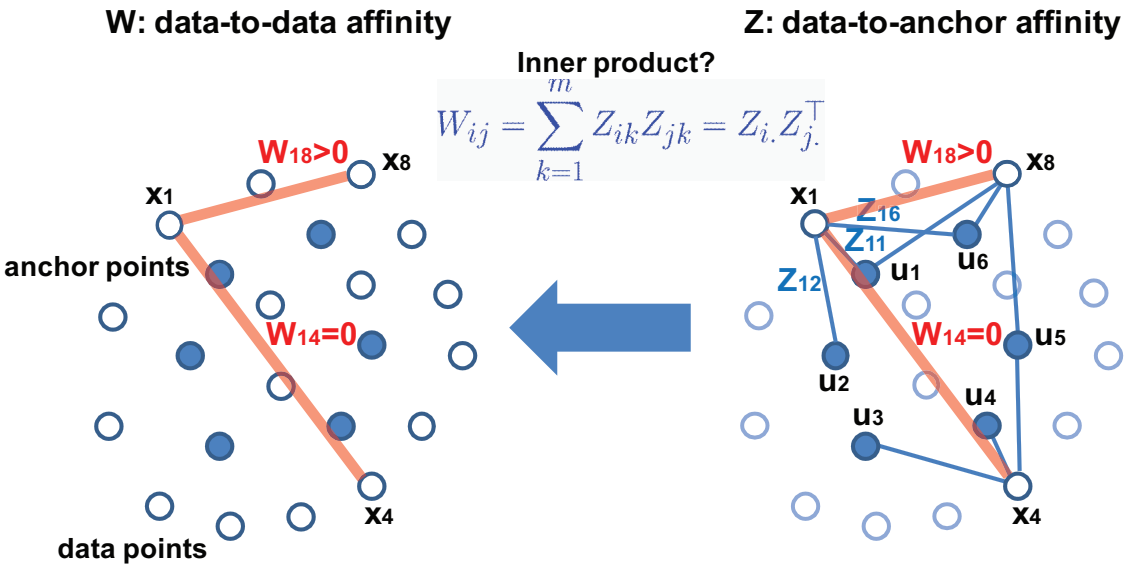


Figure 1.5: Design the data-to-data affinity matrix W using the data-to-anchor affinity matrix Z.

to one million samples demonstrate the efficacy of the Anchor Graph-based low-rank kernel generation technique. A linear SVM with the proposed low-rank kernel achieves remarkable performance gains over state-of-the-art large-scale GSSL algorithms.

Third, we present a novel graph-based hashing technique that we name *Anchor Graph Hashing* (AGH) in Chapter 4. AGH is fully unsupervised but can automatically discover the neighborhood structure inherent in the data to learn appropriate compact codes. To make such an approach computationally feasible on large databases, we utilize Anchor Graphs having been presented in Chapter 2 to obtain tractable low-rank adjacency matrices and derive the nonlinear hash functions from the eigenspectra of such low-rank matrices. The formulation of our Anchor Graph-driven hash functions allows constant time hashing of a new data point by extrapolating graph Laplacian eigenvectors to eigenfunctions. Finally, we describe a hierarchical threshold learning procedure in which each eigenfunction yields multiple bits, leading to higher search accuracy. Experimental comparison with the other state-of-the-art hashing methods on several large datasets demonstrates the efficacy of the presented AGH approach in terms of searching semantically similar neighbors.

Fourth, we present a novel *Kernel-Based Supervised Hashing* (KSH) technique in Chap-

ter 5. KSH requires a limited amount of supervised information, i.e., similar and dissimilar data pairs, and a feasible training cost in achieving high quality hashing. The idea of KSH is to map the data to compact binary codes whose Hamming distances are simultaneously minimized on similar pairs and maximized on dissimilar pairs. Our approach is distinct from prior work in utilizing the equivalence between optimizing the code inner products and the Hamming distances. This enables us to sequentially and efficiently train the hash functions one bit at a time, yielding very short yet discriminative codes whose code inner products are optimized explicitly and at the same time whose Hamming distances are optimized in an implicit manner. Extensive experiments on several image benchmarks of up to one million samples demonstrate that our approach KSH significantly outperforms state-of-the-arts in searching both semantically similar neighbors and metric distance neighbors.

Lastly, we present a novel hyperplane hashing technique in Chapter 6. The existing hyperplane hashing methods are randomized in nature and need long hash codes to achieve reasonable search accuracy, thus suffering from reduced search speed and large memory overhead. To this end, we make our hyperplane hashing technique yield compact hash codes through designing principled hash functions. The key idea is the bilinear form of the designed hash functions, which leads to a higher collision probability than the existing hyperplane hash functions when using random projections. To further increase the performance, we develop a learning based framework in which the bilinear functions are directly learned from the input data. This results in short yet discriminative codes, and also boosts the search performance over the random projection based solutions. Large-scale active learning experiments carried out on several large datasets of up to one million samples demonstrate the overall superiority of the presented hyperplane hashing technique.

1.3 Thesis Overview

Finally, we provide the overview of this thesis in brief. We divide the whole thesis into two main parts:

Part I: Scalable Classification with Graphs, containing Chapter 2 and Chapter 3.

Part II: Nearest Neighbor Search with Hashing, containing Chapter 4, Chapter

5 and Chapter 6.

The two parts are correlated as the idea of Anchor Graphs presented in Chapter 2 is not only applied to develop scalable classification models in Part I but also exploited to derive unsupervised hashing with compact codes in Part II. Moreover, Chapter 6 actually marries up classification explored in Part I with hashing investigated in Part II, since hyperplane hashing can directly benefit classification under an active learning framework.

As an ending, Part III (containing only Chapter 7) draws our conclusions of the thesis, where we summarize the presented large-scale machine learning techniques, highlight their contributions in both theory and practice, and provide some future research directions.

Part I

Scalable Classification with Graphs

Chapter 2

Large Graph Construction – Anchor Graphs

A data graph offers a very attractive way of representing high-dimensional data and discovering the underlying information, e.g., the neighborhood structure, essential for the data. In the literature, neighborhood graphs have demonstrated wide usage in a great number of machine learning, data mining, and computer vision problems including classification, clustering, regression, dimensionality reduction, relevance ranking, and so on. However, scalability of data graphs to the explosive growth of data sets remains challenging. For example, the time complexity for constructing a neighborhood graph is quadratic in the database size, which is computationally infeasible for practical large-scale data sets of millions up to billions of samples.

In this chapter, we address the scalability issue plaguing neighborhood graph construction via a small number of anchor points which adequately cover the entire point cloud. Critically, these anchor points enable nonparametric regression that predicts the label for each data point as a locally weighted average of the labels on anchor points. Because conventional graph construction is inefficient in a large scale, we present a computationally and storage efficient large graph by coupling anchor-based label prediction and adjacency matrix design. Contrary to the Nyström approximation of graph adjacency matrices which results in indefinite graph Laplacians and in turn leads to potential non-convex optimiza-

tion over graphs, the presented graph construction approach based on a unique idea called *Anchor Graph* provides nonnegative adjacency matrices to guarantee positive semidefinite graph Laplacians. Our approach scales linearly with the data size and in practice usually produces a large and sparse graph. Experiments on three toy datasets and three real-world datasets validate the scalable trait and good quality of Anchor Graphs.

In this chapter, we state the problem background of neighborhood graphs and typical graph-based learning in Section 2.1, review the related work in Section 2.2, introduce the notations that we will use in Section 2.3, give the prerequisites for large graph construction in Section 2.4, present our large graph Anchor Graph in Section 2.5, develop a scalable graph-based learning method Anchor Graph Regularization in Section 2.6, show the experimental results in Section 2.7, and finally give our summary and discussion in Section 2.8.

2.1 Problem Background

Above all, we elicit the graph representation of data which will be used in this chapter. Given a data set $\mathcal{X} = \{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^n$ with cardinality $|\mathcal{X}| = n$, an undirected graph converted from the input data $\mathcal{X}$ is represented by $\mathcal{G} = (\mathcal{X}, E, W)$. $\mathcal{X}$ refers to a set of nodes (or vertices) each of which corresponds to a data point. $E = \{e_{ij}\}$ is a set of graph edges and each edge e_{ij} has a nonnegative weight W_{ij}. The edge weights are collected to form a weighted graph adjacency matrix $W = (W_{ij})_{i,j} \in \mathbb{R}^{n \times n}$. A proper edge connecting strategy is crucial to the topological structure of the data graph $\mathcal{G}$. Throughout this thesis, we consider the widely exploited neighborhood graphs which adopt such a connecting strategy: put an edge e_{ij} between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ if they are neighbors under some distance function $\mathcal{D}(\cdot)$. There are two main types of neighborhood graphs, kNN graph and ε-neighborhood graph [33][153][10]. The former sets up edges over k nearest neighbors among $\mathcal{X}$, and the latter identifies an edge e_{ij} if and only if $\mathcal{D}(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq \varepsilon$. Although there are other strategies for establishing edges over data points, it turns out that a kNN graph has advantages over others (e.g., a ε-neighborhood graph) as shown in [67]. One of main advantages is that a kNN graph provides a better adaptive connectivity.

Let us define a diagonal node-degree matrix $D \in \mathbb{R}^{n \times n}$ whose diagonal entries are

$D_{ii} = \sum_{j=1}^{n} W_{ij}$. Importantly, the well-known (weighted) *graph Laplacian* [33] is calculated by

$$L = D - W. \tag{2.1}$$

The graph Laplacian determines a *point cloud Laplace operator* $\Delta_{\mathcal{X}}$ applied to a function $f : \mathbb{R}^d \mapsto \mathbb{R}$, which is defined on data points:

$$(\Delta_{\mathcal{X}} f)(\boldsymbol{x}_i) = D_{ii} f(\boldsymbol{x}_i) - \sum_{j=1}^{n} W_{ij} f(\boldsymbol{x}_j). \tag{2.2}$$

Such a Laplace operator $\Delta_{\mathcal{X}}$ leads to a graph-based semi-norm

$$\begin{aligned}
\|f\|_{\mathcal{G}}^2 &= \langle f, \Delta_{\mathcal{X}} f \rangle_{\mathcal{G}} \\
&= \sum_{i=1}^{n} f(\boldsymbol{x}_i)(\Delta_{\mathcal{X}} f)(\boldsymbol{x}_i) \\
&= \sum_{i=1}^{n} f(\boldsymbol{x}_i) \left(D_{ii} f(\boldsymbol{x}_i) - \sum_{j=1}^{n} W_{ij} f(\boldsymbol{x}_j) \right) \\
&= \sum_{i=1}^{n} D_{ii} f^2(\boldsymbol{x}_i) - \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} f(\boldsymbol{x}_i) f(\boldsymbol{x}_j) \\
&= \boldsymbol{f}^\top D \boldsymbol{f} - \boldsymbol{f}^\top W \boldsymbol{f} \\
&= \boldsymbol{f}^\top L \boldsymbol{f},
\end{aligned} \tag{2.3}$$

where the vector $\boldsymbol{f} = \begin{bmatrix} f(\boldsymbol{x}_1) \\ \cdots \\ f(\boldsymbol{x}_n) \end{bmatrix}$ contains the outputs of f on the point cloud $\mathcal{X}$. It turns out that $\|f\|_{\mathcal{G}}^2 = \boldsymbol{f}^\top L \boldsymbol{f} \geq 0$ for any function f, because

$$\begin{aligned}
\|f\|_{\mathcal{G}}^2 &= \boldsymbol{f}^\top L \boldsymbol{f} \\
&= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} \left(f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j) \right)^2 \\
&\geq 0.
\end{aligned} \tag{2.4}$$

Immediately, the graph Laplacian matrix L is guaranteed to be positive semidefinite.

Delving into the norm $\|f\|_{\mathcal{G}}$, it sums weighted variations of the function f on all edges of the graph $\mathcal{G}$. Therefore, minimizing the norm $\|f\|_{\mathcal{G}}$ encourages that f varies smoothly

along the edges of $\mathcal{G}$ such that adjacent nodes have similar function values. Essentially, the graph Laplacian L, that decides the smoothness norm $\|f\|_{\mathcal{G}}$, plays a critical role in differential geometry. [11][172] have proven that under certain conditions, the point cloud Laplace operator $\Delta_{\mathcal{X}}$ determined by the graph Laplacian L asymptotically converges to the *Laplace-Beltrami operator* $\Delta_{\mathcal{M}}$ on the underlying Riemannian manifold $\mathcal{M}$ from which the data points $\mathcal{X}$ were sampled. Simply speaking, $\lim_{|\mathcal{X}|\to\infty} \Delta_{\mathcal{X}} \to \Delta_{\mathcal{M}}$. Hence, the smoothness norm $\|f\|_{\mathcal{G}}$ described above is geometrically meaningful and hereby applied to many manifold-motivated fields for measuring smoothness of target functions.

Let us introduce a classical learning paradigm with graphs which is referred to as *graph-based semi-supervised learning* (GSSL) in literature. Assume that we are given a set of labeled samples as $\mathcal{X}_l = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_l\}$ with cardinality $|\mathcal{X}_l| = l$ and a set of unlabeled samples $\mathcal{X}_u = \{\boldsymbol{x}_{l+1}, \cdots, \boldsymbol{x}_n\}$ with cardinality $|\mathcal{X}_u| = u = n - l$, where typically $l \ll n$. The labeled data set $\mathcal{X}_l$ is associated with labels $\mathcal{Y}_l = \{y_1, \cdots, y_l\}$ in which $y_i \in \{1, \cdots, c\}$ ($i = 1, \cdots, l$ and c is the total number of classes). The goal of GSSL is to infer the missing labels $\mathcal{Y}_u = \{y_{l+1}, \cdots, y_n\}$ of the unlabeled data $\mathcal{X}_u$. A crucial component of GSSL is the construction of a neighborhood graph $\mathcal{G}(\mathcal{X}, E, W)$ from the entire input data $\mathcal{X} = \mathcal{X}_l \cup \mathcal{X}_u$. After that, GSSL algorithms use $\mathcal{G}$ and the initial seed labels $\mathcal{Y}_l$ to infer $\hat{\mathcal{Y}}_u = \{\hat{y}_{l+1}, \cdots, \hat{y}_n\}$ which are expected to match the true labels $\mathcal{Y}_u$ as well as possible.

For convenient expression, we encode the initial label information $\mathcal{Y}_l$ into a label matrix $Y \in \{1, 0\}^{n\times c}$, where $Y_{ij} = 1$ if and only if sample $\boldsymbol{x}_i$ has a label $j \in \{1, \cdots, c\}$, i.e., $y_i = j$, and $Y_{ij} = 0$ otherwise. If the label of sample $\boldsymbol{x}_i$ is unknown or unavailable, the whole row $Y_{i\cdot}$ is zero. For illustration, Figure 2.1 (originally from [115]) gives an example of a graph as well as the corresponding graph quantities. Several classical GSSL algorithms [212][208][184] solve the semi-supervised classification problem by propagating or spreading the initial *discrete* labels $\mathcal{Y}_l$ from the labeled data $\mathcal{X}_l$ to the unlabeled data $\mathcal{X}_u$ over the built neighborhood graph $\mathcal{G}$. Meanwhile, a *continuous* soft label matrix $F \in \mathbb{R}^{n\times c}$ is obtained by means of minimizing a proper cost or penalty over the graph $\mathcal{G}$. Then the missing labels of the unlabeled data $\mathcal{X}_u$ are estimated as $\hat{y}_i = \arg\max_{j\in\{1,\cdots,c\}} F_{ij}$ $(i = l + 1, \cdots, n)$.

Semi-supervised learning (SSL) [23][211][213], that is the central subject we will investigate in this chapter and the next chapter, has significant impact in pervasive applications

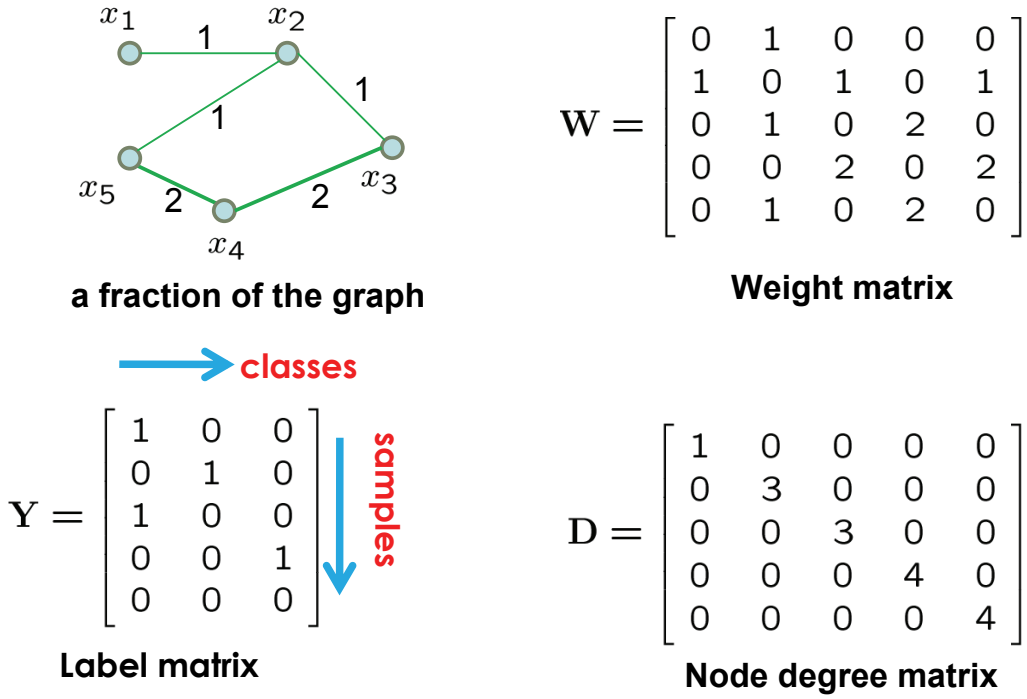


Figure 2.1: An example of a graph $\mathcal{G}$ with only five nodes. W is the associated weighted adjacency matrix, $D = \text{diag}(W\mathbf{1})$ is the node-degree matrix, and Y is the label matrix.

of machine learning and pattern recognition. In these practical applications, one frequently encounters situations where only a few labeled data are available and large amounts of data remain unlabeled. The labeled data often suffer from difficult and expensive acquisition whereas the unlabeled data can be cheaply and automatically gathered. SSL has been intensively explored to cope with the very situations of limited labeled data and abundant unlabeled data.

With rapid development of the Internet, now we can collect massive (up to hundreds of millions) unlabeled data such as images and videos, and then the need for large scale SSL arises. As shown in Figure 2.2, once some seed labels are available (these labels may be acquired from users' annotation or samples' meta-data, e.g., tags, captions, etc.), one can establish a data graph over the collected data set and then apply GSSL with the graph to infer labels for a large amount of unlabeled data samples. Despite the appealing scenario of GSSL in large-scale learning tasks, most GSSL methods scale poorly with the data size n because of the computationally challenging step of large graph construction. Typically,

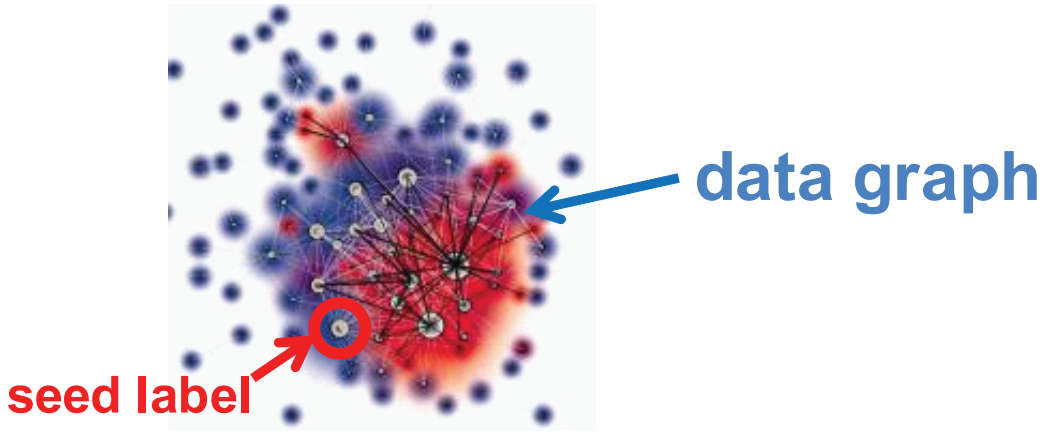


Figure 2.2: A graph-based semi-supervised learning task.

provided with a data set $\mathcal{X} = \{x_i \in \mathbb{R}^d\}_{i=1}^n$, the widely used kNN graph needs a time complexity of $O(dn^2)$ which is certainly expensive and intractable for practical large-scale data collections in the level of millions or even billions.

In this chapter, to overcome the computational bottleneck of traditional graph construction approaches, we present a novel large graph construction approach named *Anchor Graph* [113] which is efficient in both computation and storage, achieving linear time and space complexities. Our approach usually yields a large sparse graph and is capable of scaling linearly to gigantic databases.

2.2 Related Work

In the past decade, researchers found that data graphs, acting as an informative platform, can be employed to deal with tremendous data-driven applications in a large variety of problems arising from, but not limited to, machine learning, pattern recognition, computer vision, data mining, information retrieval, and bioinformatics. Why have graphs attracted extensive attention in both theory and practice? One intuitive advantage that graph-based methods bear is that data graphs are capable of reflecting and revealing the intrinsically *low-dimensional* manifolds which are embedded in *high-dimensional* data collections, such as images and videos, and tend to capture the hidden semantics of data. Theoretically, it turns out that the graph Laplacian L (or its normalized versions $D^{-1}L$ and $D^{-1/2}LD^{-1/2}$) of a

point cloud of data samples converges to the Laplace-Beltrami operator defined on the underlying manifold where the data samples reside [65][66][11][172], and that the eigenvectors of the graph Laplacian converge to the eigenfunctions of the Laplace-Beltrami operator on the manifold [13][35][131][34]. In-depth algebraic theory about graphs and graph spectrums can be found in the *Spectral Graph Theory* [33].

Due to the intuitive yet provable characteristics of graphs, graphs have led to enormous implications in manifold learning [169][144][10], dimensionality reduction [63], feature extraction [64], feature selection [62], image segmentation [153][107], spectral embedding and clustering [135][203][182], semi-supervised learning [212][208][184], active semi-supervised learning [215][57][73], online learning [70][69][68], online semi-supervised learning [177], online active semi-supervised learning [54], relevance ranking [209][210], and so on. Besides the progress in theory, graphs have also motivated numerous exciting applications such as image and video retrieval [59][72][199], web image search and reranking [82][185][114], web image tag ranking [109] and refinement [110], web page categorization [206], and protein classification [192].

Among the vast number of semi-supervised learning methods, graph-based semi-supervised learning (GSSL) is substantially appealing because it is easy to implement and generally renders closed-form solutions. The representative techniques include graph partitioning [15][16][84][99], graph-based random walks [167][7], manifold regularization [12][157][124], graph regularization [212][208][184][183][166], and graph Laplacian eigenfunction fitting [159][51][49]. Comprehensive surveys of these methods can be found in [23][211][213].

Although GSSL has been studied extensively, it is likely to lack sufficient robustness on real-world data sets which usually involve noise, because of the sensitivity of graphs [112]. The quality of graphs is very sensitive to the edge connecting strategy, the choice of edge weighting functions, and the related parameters (e.g., k and ε). These factors will considerably affect the performance of GSSL. Therefore, beyond traditional kNN graph and ε-neighborhood graph [121], more sophisticated methods for fine graph construction are needed. For example, [76] tried to learn to prune the redundant edges by optimizing b-matching, resulting in a regular graph in which each node is incident to an equal number of edges; [112] attempted to learn the entire adjacency matrix W of a graph in a nonparametric

mode, leading to a unit-degree graph; [37] integrated graph construction and label diffusion into a joint learning framework, producing a discriminative graph. Although these methods show up influences of different graph topological structures on the outcome performance of GSSL, they all need to build a kNN graph as a heuristic solution to start their graph learning procedures. Consequently, all of them still suffer from the quadratic computational cost $O(dn^2)$, and cannot yet be scalable to gigantic databases.

From above discussion, we are aware that designing efficient large graphs constitutes a major bottleneck of large-scale GSSL. In the recent literature, constructing large graphs over gigantic databases begins to draw attention. [28] used divide and conquer algorithms to compute an approximate kNN graph in nearly linear time $O(dn^\rho)$ $(1 < \rho < 2)$, but the space complexity could still be large because a kd-tree indexing structure must be saved in memory. Another approximate kNN graph construction approach [41] performed local search instead of shared global indexing like kd-tree according to arbitrarily given similarity measures, achieving an empirical time complexity of $O(dn^{1.14})$.

By utilizing a small set of anchor points $\mathcal{U} = \{\boldsymbol{u}_k\}_{k=1}^m$ $(m \ll n)$, [204] applied the Nyström matrix approximation [193] to yield a low-rank adjacency matrix

$$W = K_{nm} K_{mm}^{-1} K_{nm}^\top, \tag{2.5}$$

where the two matrices K_{nm} and K_{mm} denote two kernel matrices between the raw data $\mathcal{X}$ and the anchor points $\mathcal{U}$, and $\mathcal{U}$ and $\mathcal{U}$, respectively. Although the Nyström graph enjoys a strictly linear time complexity $O(dmn)$, it may yield improper dense graphs that could limit the performance of GSSL. The Anchor Graph that we will describe in Section 2.5 capitalizes on a Markov random walk model across data points and anchor points to derive a low-rank graph adjacency matrix:

$$W = Z \Lambda^{-1} Z^\top, \tag{2.6}$$

where $Z \in \mathbb{R}^{n \times m}$ is the data-to-anchor affinity matrix, and $\Lambda \in \mathbb{R}^{m \times m}$ is a diagonal matrix with definition $\Lambda = \mathrm{diag}(Z^\top \mathbf{1})$. Because Z is made highly sparse, such an adjacency matrix W is also sparse empirically.

A very important property stemming from the Anchor Graph's design strategy is $W \geq 0$. This nonnegative property is sufficient to guarantee the resulting graph Laplacian $L =$

Table 2.1: Summary of the properties of efficient large graphs. d is the data dimension and n is the data size. Denote by $L \succeq 0$ positive semidefinite L.

Large Graph Construction	**Time Complexity**	W is sparse?	$W \geq 0$?	$L \succeq 0$?
Approximate kNN Graph [28][41]	$O(dn^{\rho})$ $(1 < \rho < 2)$	Yes	Yes	Yes
Nyström Graph [204]	$O(dmn)$ $(m \ll n)$	No	No	No
Anchor Graph (this chapter)	$O(dmn)$ $(m \ll n)$	Yes	Yes	Yes

$D - W$ to be positive semidefinite [33], and thus ensures global optimum of GSSL. Table 2.1 summarizes the key properties of aforementioned efficient large graphs.

2.3 Notations

In this section, we first define the notations and symbols that we will use to delineate our approach in the rest of this chapter. All notations as well as their definitions are listed in Tables 2.2 and 2.3.

2.4 Prerequisites

We try to address the scalability issue pertaining to large graph construction from two perspectives: anchor-based label prediction and adjacency matrix design.

2.4.1 Anchor-Based Label Prediction

Our key observation is that the computational intensiveness of graph-based semi-supervised learning (GSSL) stems from the full-size label prediction models. Since the number of unlabeled samples is huge in large scale applications, learning full-size prediction models is inefficient.

Suppose a soft label prediction function $f : \mathbb{R}^d \mapsto \mathbb{R}$ defined on the input data points $\mathcal{X} = \{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^n$. To work under a large scale, [40][216] made the label prediction function be a weighted average of the labels on a small set of anchor (or landmark) points. As such,

Table 2.2: Table of notations.

Notation	Definition
n	The number of data points
l	The number of the labeled data
m	The number of anchor points
d	The dimension of data or anchor points
i, j	The indices of data points
k	The index of anchor points
$\boldsymbol{x}_i \in \mathbb{R}^d$	The ith data point
$\boldsymbol{u}_k \in \mathbb{R}^d$	The kth anchor point
$\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^n$	The data set
$\mathcal{X}_l = \{\boldsymbol{x}_i\}_{i=1}^l$	The labeled data set
$\mathcal{U} = \{\boldsymbol{u}_k\}_{k=1}^m$	The anchor set
$U = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m] \in \mathbb{R}^{d \times m}$	The anchor data matrix
c	The number of classes
$y_i \in [1:c]$	The class label of $\boldsymbol{x}_i$, $i \in [1:l]$
$\hat{y}_i \in [1:c]$	The estimated class label of $\boldsymbol{x}_i$, $i \in [l+1:n]$
$Y \in \mathbb{R}^{n \times c}$	The initial label matrix
$\mathcal{G}(\mathcal{X}, E, W)$	A data graph
$E \subseteq \mathcal{X} \times \mathcal{X}$	The set of edges in $\mathcal{G}$
$W = (W_{ij})_{i,j} \in \mathbb{R}^{n \times n}$	The weighted adjacency matrix of $\mathcal{G}$
$D = \operatorname{diag}(W\mathbf{1}) \in \mathbb{R}^{n \times n}$	The diagonal node-degree matrix of $\mathcal{G}$
$L = D - W \in \mathbb{R}^{n \times n}$	The graph Laplacian matrix of $\mathcal{G}$
$\|\cdot\|_{\mathcal{G}}$	The graph Laplacian regularization norm
$f : \mathbb{R}^d \mapsto \mathbb{R}$	A soft label prediction function
$\boldsymbol{f} \in \mathbb{R}^n$	The soft label vector of all data points $\mathcal{X}$
$\boldsymbol{a} \in \mathbb{R}^m$	The soft label vector of anchor points $\mathcal{U}$
$Z = (Z_{ij})_{i,j} \in \mathbb{R}^{n \times m}$	The data-to-anchor affinity matrix between $\mathcal{X}$ and $\mathcal{U}$
$\Lambda = \operatorname{diag}(Z^\top \mathbf{1}) \in \mathbb{R}^{m \times m}$	The diagonal anchor-degree matrix of an Anchor Graph
s	The number of nearest anchors in $\mathcal{U}$ for each data point
$\langle i \rangle \subset [1:m]$	The set of indices of s nearest anchors in $\mathcal{U}$ for $\boldsymbol{x}_i$
$\mathcal{K}_h : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$	A kernel function with bandwidth h

Table 2.3: Table of notations (continued).

Notation	Definition
$\boldsymbol{z}_i \in \mathbb{R}^s$	The coefficient vector of combining s anchors for $\boldsymbol{x}_i$
$g(\boldsymbol{z}_i) \in \mathbb{R}$	The objective function for optimizing $\boldsymbol{z}_i$
$\mathbb{S} \subset \mathbb{R}^s$	The multinomial simplex in $\mathbb{R}^s$
$\Pi_{\mathbb{S}}(\boldsymbol{z})$	The projection of $\boldsymbol{z}$ onto simplex $\mathbb{S}$
$\mathcal{B}(\mathcal{X},\mathcal{U},\mathcal{E},Z)$	The data-anchor bipartite graph
$\mathcal{E} \subseteq \mathcal{X} \times \mathcal{U}$	The set of edges in $\mathcal{B}$
$B \in \mathbb{R}^{(n+m)\times(n+m)}$	The adjacency matrix of $\mathcal{B}$
$D^B = \mathrm{diag}(B\mathbf{1}) \in \mathbb{R}^{(n+m)\times(n+m)}$	The diagonal node-degree matrix of $\mathcal{B}$
$p^{(1)}$	One-step transition probability of random walks on $\mathcal{B}$
$p^{(2)}$	Two-step transition probability of random walks on $\mathcal{B}$
$\boldsymbol{a}_j \in \mathbb{R}^m$	The soft label vector of $\mathcal{U}$ for the jth class
$A = [\boldsymbol{a}_1, \cdots, \boldsymbol{a}_c] \in \mathbb{R}^{m\times c}$	The soft label matrix of $\mathcal{U}$
$Y_l \in \mathbb{R}^{l\times c}$	The sub-matrix of Y corresponding to $\mathcal{X}_l$
$Z_l \in \mathbb{R}^{l\times m}$	The sub-matrix of Z corresponding to $\mathcal{X}_l$
$\gamma > 0$	The graph regularization parameter
$\tau_j > 0$	The normalization factor for the jth class
$\boldsymbol{z}(\boldsymbol{x}) \in \mathbb{R}^m$	The data-to-anchor mapping

if one can infer the labels associated with the much smaller anchor set, the labels of the raw data points will be easily obtained by a simple linear combination.

The idea is to introduce an anchor set $\mathcal{U} = \{\boldsymbol{u}_k \in \mathbb{R}^d\}_{k=1}^m$ in which each $\boldsymbol{u}_k$ acts as an *anchor* point since we represent the label prediction function f in terms of these points, that is,

$$f(\boldsymbol{x}_i) = \sum_{k=1}^{m} Z_{ik} f(\boldsymbol{u}_k), \tag{2.7}$$

where Z_{ik}'s are data-adaptive weights. Such a label prediction essentially falls into non-parametric regression [58]. Let us define two soft label vectors $\boldsymbol{f} = [f(\boldsymbol{x}_1), \cdots, f(\boldsymbol{x}_n)]^\top$ and $\boldsymbol{a} = [f(\boldsymbol{u}_1), \cdots, f(\boldsymbol{u}_m)]^\top$, and rewrite eq. (2.7) as

$$\boldsymbol{f} = Z\boldsymbol{a},\ Z \in \mathbb{R}^{n\times m},\ m \ll n. \tag{2.8}$$

This formula serves as a main disposal of scalable SSL because it reduces the solution space of unknown labels from large $\boldsymbol{f}$ to much smaller $\boldsymbol{a}$. This economical label prediction model eq. (2.8) surely mitigates the computational burden of the original full-size prediction models.

Importantly, we take these anchor points $\{\boldsymbol{u}_k\}$ as K-means clustering centers instead of randomly sampled exemplars because of the found fact that K-means clustering centers have a stronger representation power to adequately cover the vast point cloud $\mathcal{X}$ [205][204]. While the K-means clustering step incurs additional computation, its time complexity $O(dmnT)$ (T is the iteration number) is relatively manageable compared with the quadratic complexity $O(dn^2)$ needed by traditional graph construction schemes. If we set m to a number much smaller than n, the K-means running time is much faster than the time for constructing a kNN graph. Additionally, some fast implementations of approximate K-means clustering such as [86] may be exploited to further mitigate the computational overhead.

2.4.2 Adjacency Matrix Design

Recall that in the literature an undirected weighted graph $\mathcal{G}(\mathcal{X}, E, W)$ is built on n data points. $\mathcal{X}$ also refers to a set of nodes (or vertices). $E \subseteq \mathcal{X} \times \mathcal{X}$ is a set of edges connecting adjacent nodes, and $W \in \mathbb{R}^{n \times n}$ is a weighted adjacency matrix which measures the strength of edges. The time cost of the broadly used kNN graph construction is $O(dn^2)$, so even this conventional graph construction approach is infeasible at a large scale. Although we may utilize approximate kNN graph construction as proposed by [28][41] to save the time cost, large scale matrix manipulations such as inversion and linear system solving upon the huge matrix W remain a big hurdle. The associated time complexities are $O((kn)^{1.31})$ at least (see Subsection 2.6.1).

On the other hand, it is unrealistic to save in memory a matrix W as large as $n \times n$. Hence, designing a memory and computationally tractable W constitutes a major bottleneck of large scale GSSL. We should find an approach to parsimoniously represent W for large graphs.

2.4.3 Design Principles

Now we investigate some principles for designing Z and W tailored to large scale problems.

Principle (1) We impose the nonnegative normalization constraints $\sum_{k=1}^{m} Z_{ik} = 1$ and $Z_{ik} \geq 0$ to maintain the unified range of values for all soft labels predicted via regression. The *manifold assumption* [12] implies that contiguous data points should have similar labels while distant data points are very unlikely to take similar labels. This motivates us to also impose $Z_{ik} = 0$ when anchor $\boldsymbol{u}_k$ is far away from $\boldsymbol{x}_i$ so that the regression on $\boldsymbol{x}_i$ is a locally weighted average in spirit. As a result, $Z \in \mathbb{R}^{n \times m}$ is nonnegative as well as sparse.

Principle (2) We require $W \geq 0$. The nonnegative adjacency matrix is sufficient to make the resulting graph Laplacian $L = D - W$ ($D = \text{diag}(W\mathbf{1}) \in \mathbb{R}^{n \times n}$ is the diagonal node-degree matrix) positive semidefinite [33]. This nonnegative property is important to guarantee global optimum of many GSSL algorithms.

Principle (3) We prefer a sparse W because sparse graphs have much less spurious connections among dissimilar points and tend to exhibit high quality. [211] has pointed out that fully-connected dense graphs perform worse than sparse graphs in practice.

Intuitively, we would use the nonnegative sparse matrix Z to design the nonnegative sparse matrix W. Actually, in the next section, we are able to design Z and W jointly and generate empirically sparse large graphs. On the contrary, the recently proposed Prototype Vector Machine (PVM) [204] designed Z and W separately, producing improper dense graphs. In addition, when using the Nyström method [193] to approximate a predefined W like a kernel matrix, PVM fails to preserve the nonnegative property of graph adjacency matrices. Therefore, PVM cannot guarantee that the graph Laplacian regularization term appearing in its cost functions is convex, so PVM suffers heavily from local minima. Crucially, we are not trying to approximate any predefined W; instead, we design it directly to cater for the nonnegative and sparse properties.

2.5 Anchor Graphs

2.5.1 Design of Z

We aim at designing a regression matrix Z that measures the potential relationship between raw samples $\mathcal{X}$ and anchors $\mathcal{U}$ (note that $\mathcal{U}$ is outside $\mathcal{X}$). Following *Principle* (1) in the last section, we desire to keep nonzero Z_{ik} for s ($< m$) closest anchors to $\boldsymbol{x}_i$. Exactly, the Nadaraya-Watson kernel regression [58] defines such Z_{ik} based on a kernel function $\mathcal{K}_h()$ with a bandwidth h:

$$Z_{ik} = \frac{\mathcal{K}_h(\boldsymbol{x}_i, \boldsymbol{u}_k)}{\sum_{k' \in \langle i \rangle} \mathcal{K}_h(\boldsymbol{x}_i, \boldsymbol{u}_{k'})}, \ \forall k \in \langle i \rangle, \tag{2.9}$$

where the notation $\langle i \rangle \subset [1:m]$ is the set saving the indexes of s nearest anchors of $\boldsymbol{x}_i$. Typically, we may adopt the Gaussian kernel $\mathcal{K}_h(\boldsymbol{x}_i, \boldsymbol{u}_k) = \exp(-\|\boldsymbol{x}_i - \boldsymbol{u}_k\|^2/2h^2)$ for the kernel regression.

Under the consideration that the kernel-defined weights may be sensitive to the hyperparameter h and lack a meaningful interpretation, we obtain them from another perspective: a geometric reconstruction similar to LLE [144]. Concretely, we reconstruct any data point $\boldsymbol{x}_i$ as a convex combination of its closest anchors, and the combination coefficients are preserved for the weights used in the nonparametric regression. Let us define a matrix $U = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m]$ and denote by $U_{\langle i \rangle} \in \mathbb{R}^{d \times s}$ a sub-matrix composed of s nearest anchors of $\boldsymbol{x}_i$. Then, we develop *Local Anchor Embedding* (LAE) to optimize the convex combination coefficients:

$$\begin{aligned} \min_{\boldsymbol{z}_i \in \mathbb{R}^s} \quad & g(\boldsymbol{z}_i) = \frac{1}{2}\|\boldsymbol{x}_i - U_{\langle i \rangle}\boldsymbol{z}_i\|^2 \\ \text{s.t.} \quad & \mathbf{1}^\top \boldsymbol{z}_i = 1, \quad \boldsymbol{z}_i \geq 0 \end{aligned} \tag{2.10}$$

where s entries of the vector $\boldsymbol{z}_i$ correspond to s combination coefficients contributed by s closest anchors. Beyond LLE, LAE imposes the nonnegative constraint and then the convex solution set to eq. (2.10) constitutes a *multinomial simplex*

$$\mathbb{S} = \left\{ \boldsymbol{z} \in \mathbb{R}^s : \mathbf{1}^\top \boldsymbol{z} = 1, \boldsymbol{z} \geq 0 \right\}. \tag{2.11}$$

In contrast to the regression weights as predefined in eq. (2.9), LAE is more advantageous

because it provides optimized regression weights that are also sparser than the predefined ones.

Standard quadratic programming (QP) [18] solvers can be applied to solve eq. (2.10), but most of them need to compute some approximation of the Hessian and are thus relatively expensive. We apply the projected gradient method, a first-order optimization procedure, to solve eq. (2.10) instead. The updating rule in the projected gradient method is expressed as the following iterative formula

$$\boldsymbol{z}_i^{(t+1)} = \Pi_{\mathbb{S}}\left(\boldsymbol{z}_i^{(t)} - \eta_t \nabla g(\boldsymbol{z}_i^{(t)})\right), \tag{2.12}$$

where t denotes the time stamp, $\eta_t > 0$ denotes the appropriate step size, $\nabla g(\boldsymbol{z})$ denotes the gradient of g at $\boldsymbol{z}$, and $\Pi_{\mathbb{S}}(\boldsymbol{z})$ denotes the simplex projection operator on any $\boldsymbol{z} \in \mathbb{R}^s$. Mathematically, the projection operator is formulated as

$$\Pi_{\mathbb{S}}(\boldsymbol{z}) = \arg\min_{\boldsymbol{z}' \in \mathbb{S}} \|\boldsymbol{z}' - \boldsymbol{z}\|. \tag{2.13}$$

Such a projection operator has been implemented efficiently in $O(s \log s)$ time [44], which is described in Algorithm 1.

To achieve faster optimization, we employ *Nesterov's method* [134] to accelerate the gradient decent in eq. (2.12). As a brilliant achievement in the optimization field, Nesterov's method has a much faster convergence rate than the traditional methods such as gradient descent and subgradient descent. We present LAE accelerated by Nesterov's method in Algorithm 2. After solving the optimal weight vector $\boldsymbol{z}_i$, we set

$$Z_{i,\langle i \rangle} = \boldsymbol{z}_i^\top,\ |\langle i \rangle| = s,\ \boldsymbol{z}_i \in \mathbb{R}^s \tag{2.14}$$

and $Z_{i,\overline{\langle i \rangle}} = 0$ for the rest entries of Z. To summarize, we optimize the weights used for anchor-based nonparametric regression by means of data reconstruction with contiguous anchors. For each data point, the presented LAE algorithm converges within a few iterations T' in practice. Ultimately, LAE outputs a highly sparse matrix Z (a memory space of $O(sn)$) at a total time complexity $O(dmn + s^2 nT')$. Also keep $Z\mathbf{1} = \mathbf{1}$ in mind.

2.5.2 Design of W

So far, we have set up m anchors (cluster centers) to cover a point cloud of n data samples, and also designed a nonnegative matrix Z that supports the economical label prediction

Algorithm 1 Simplex Projection

Input: A vector $\boldsymbol{z} \in \mathbb{R}^s$.

sort $\boldsymbol{z}$ into $\boldsymbol{v}$ such that $v_1 \geq v_2 \geq \cdots \geq v_s$

find $\rho = \max\{j \in [1:s] : v_j - \frac{1}{j}(\sum_{r=1}^{j} v_r - 1) > 0\}$

compute $\theta = \frac{1}{\rho}(\sum_{j=1}^{\rho} v_j - 1)$

Output: A vector $\boldsymbol{z}' = [z'_1, \cdots, z'_s]^\top$ such that $z'_j = \max\{z_j - \theta, 0\}$.

model formulated in eq. (2.8). Now we want to design the large graph adjacency matrix W directly using the obtained regression matrix Z. As displayed in Figure 2.3, Z actually captures data-to-anchor affinities whose computational cost is drastically lower than the cost of directly computing data-to-data affinities, which has been conducted in constructing conventional neighborhood graphs. The central idea of our large graph construction approach is to infer the data-to-data affinity W_{ij} in the form of the inner product between two data-to-anchor affinity vectors $Z_{i.} \in \mathbb{R}^{1\times m}$ and $Z_{j.} \in \mathbb{R}^{1\times m}$, that is, $W_{ij} \cong Z_{i.}Z_{j.}^\top$. The intuitive interpretation is that since $\{Z_{ij}\}_j$ shapes local supports of anchors for each data point $\boldsymbol{x}_i$, W_{ij} can be regarded as the overlap between the local supports of two data points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. For illustration, Figure 2.3 shows that data pair $\boldsymbol{x}_1$ and $\boldsymbol{x}_8$ become connected in W because their nearest anchors overlap, while data pair $\boldsymbol{x}_1$ and $\boldsymbol{x}_4$ remain disconnected because their nearest anchors do not overlap.

Specifically, our approach designs the graph adjacency matrix W as follows

$$W = Z\Lambda^{-1}Z^\top, \tag{2.15}$$

in which the diagonal matrix $\Lambda = \mathrm{diag}(Z^\top \mathbf{1}) \in \mathbb{R}^{m\times m}$ can be considered as an anchor-degree matrix. The diagonal element $\Lambda_{kk} = \sum_{i=1}^{n} Z_{ik}$ stores the total affinity between anchor point $\boldsymbol{u}_k$ and its adjacent data points $\boldsymbol{x}_i$'s. Immediately, such a defined adjacency matrix W satisfies *Principle* (2) since Z is nonnegative. Further, we find out that a nonnegative sparse Z leads to an empirically sparse W when the anchor points are set to cluster centers such that most data point pairs across different clusters do not share the same set of closest cluster centers. Accordingly, W satisfies *Principle* (3) in most cases[1].

[1]In an extreme case, if a *hub* anchor point exists such that a large number of data points are connected to it, then W may be dense.

Algorithm 2 Local Anchor Embedding (LAE)

Input: data points $\{\boldsymbol{x}_i\}_{i=1}^n \subset \mathbb{R}^d$, anchor point matrix $U \in \mathbb{R}^{d\times m}$, integer s.

for $i = 1$ **to** n **do**

for $\boldsymbol{x}_i$ find s nearest neighbors in U, saving the index set $\langle i\rangle$;

define functions $g(\boldsymbol{z}) = \|\boldsymbol{x}_i - U_{\langle i\rangle}\boldsymbol{z}\|^2/2$, $\nabla g(\boldsymbol{z}) = U_{\langle i\rangle}^\top U_{\langle i\rangle}\boldsymbol{z} - U_{\langle i\rangle}^\top \boldsymbol{x}_i$, and $\tilde{g}_{\beta,\boldsymbol{v}}(\boldsymbol{z}) = g(\boldsymbol{v}) + \nabla g(\boldsymbol{v})^\top(\boldsymbol{z}-\boldsymbol{v}) + \beta\|\boldsymbol{z}-\boldsymbol{v}\|^2/2$;

initialize $\boldsymbol{z}^{(1)} = \boldsymbol{z}^{(0)} = \mathbf{1}/s$, $\delta_{-1} = 0$, $\delta_0 = 1$, $\beta_0 = 1$, $t = 0$;

repeat

$t = t+1$, $\alpha_t = \frac{\delta_{t-2}-1}{\delta_{t-1}}$

set $\boldsymbol{v}^{(t)} = \boldsymbol{z}^{(t)} + \alpha_t(\boldsymbol{z}^{(t)} - \boldsymbol{z}^{(t-1)})$

for $j = 0, 1, \cdots$ **do**

$\beta = 2^j\beta_{t-1}$, $\boldsymbol{z} = \Pi_{\mathbb{S}}(\boldsymbol{v}^{(t)} - \frac{1}{\beta}\nabla g(\boldsymbol{v}^{(t)}))$

if $g(\boldsymbol{z}) \le \tilde{g}_{\beta,\boldsymbol{v}^{(t)}}(\boldsymbol{z})$ **then**

update $\beta_t = \beta$ and $\boldsymbol{z}^{(t+1)} = \boldsymbol{z}$

break

end if

end for

update $\delta_t = \frac{1+\sqrt{1+4\delta_{t-1}^2}}{2}$

until $\boldsymbol{z}^{(t)}$ converges;

$\boldsymbol{z}_i = \boldsymbol{z}^{(t)}$.

end for

Output: LAE vectors $\{\boldsymbol{z}_i\}_{i=1}^n$.

We term the large graph $\mathcal{G}$ delineated by the adjacency matrix W in eq. (2.15) an *Anchor Graph*. Eq. (2.15) is the core finding of this chapter, which constructs a nonnegative and empirically sparse graph adjacency matrix W via crafty matrix factorization. Furthermore, it couples anchor-based label prediction and adjacency matrix design via the common matrix Z. Hence, we only need to save Z, linear in the data size n, in memory as it not only contributes to the final label prediction but also skillfully constructs the Anchor Graph. The resultant graph Laplacian of the Anchor Graph is derived by $L = I - Z\Lambda^{-1}Z^\top$ where

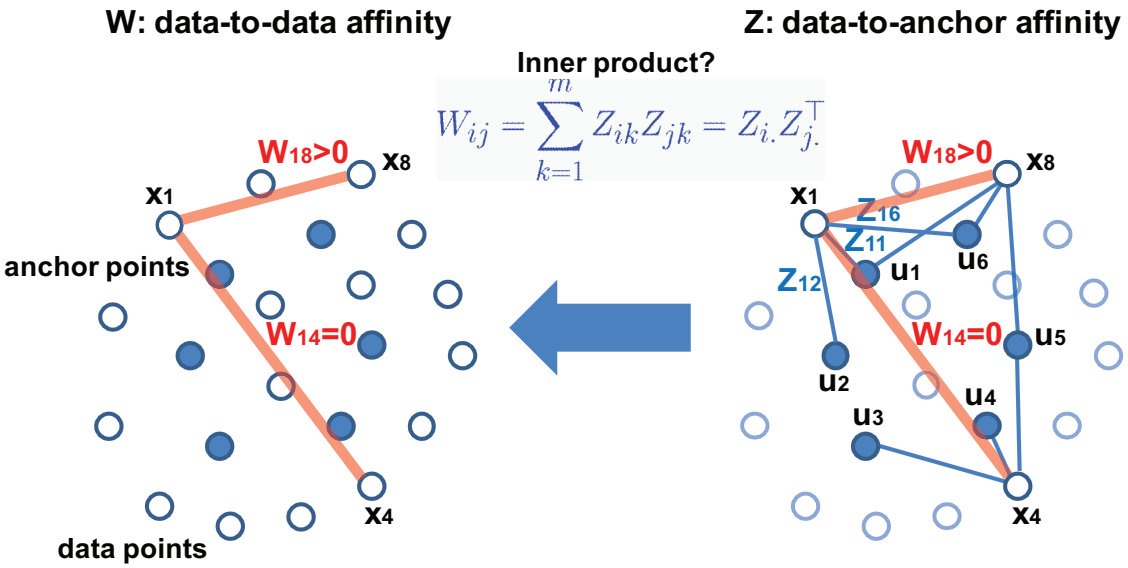


Figure 2.3: Design the data-to-data affinity matrix W using the data-to-anchor affinity matrix Z. The circles represent data points $\boldsymbol{x}_i$'s, and the solid circles represent anchor points $\boldsymbol{u}_k$'s.

the diagonal node-degree matrix is

$$\begin{aligned} D &= \operatorname{diag}(W\mathbf{1}) = \operatorname{diag}(Z\Lambda^{-1}Z^{\top}\mathbf{1}) = \operatorname{diag}(Z\Lambda^{-1}\Lambda\mathbf{1}) \\ &= \operatorname{diag}(Z\mathbf{1}) = \operatorname{diag}(\mathbf{1}) = I. \end{aligned} \tag{2.16}$$

Theoretically, we can derive eq. (2.15) in a probabilistic means. As the presented LAE algorithm derives Z from a geometrical reconstruction view, this matrix Z actually unveils a tight affinity measure between data points and anchor points. That is sound in the sense that the more an anchor $\boldsymbol{u}_k$ contributes to the reconstruction of a data point $\boldsymbol{x}_i$, the larger the affinity between them. To explicitly capture the data-to-anchor relationship, we introduce a *bipartite graph* [33] $\mathcal{B}(\mathcal{X}, \mathcal{U}, \mathcal{E}, Z)$. The new node set $\mathcal{U}$ is composed of the anchor points $\{\boldsymbol{u}_k\}_{k=1}^{m}$. The edge set $\mathcal{E}$ contains the cross edges between data points $\mathcal{X}$ and anchor points $\mathcal{U}$. Concretely, we connect an undirected edge between $\boldsymbol{x}_i$ and $\boldsymbol{u}_k$ if and only if $Z_{ik} > 0$ and designate the edge weight as Z_{ik}. Then the cross adjacency matrix between $\mathcal{X}$ and $\mathcal{U}$ is Z, and the full adjacency matrix of the bipartite graph $\mathcal{B}$ is thus $B = \begin{bmatrix} 0 & Z \\ Z^{\top} & 0 \end{bmatrix} \in \mathbb{R}^{(n+m)\times(n+m)}$ in which $Z\mathbf{1} = \mathbf{1}$. A toy example for $\mathcal{B}$ is visualized in

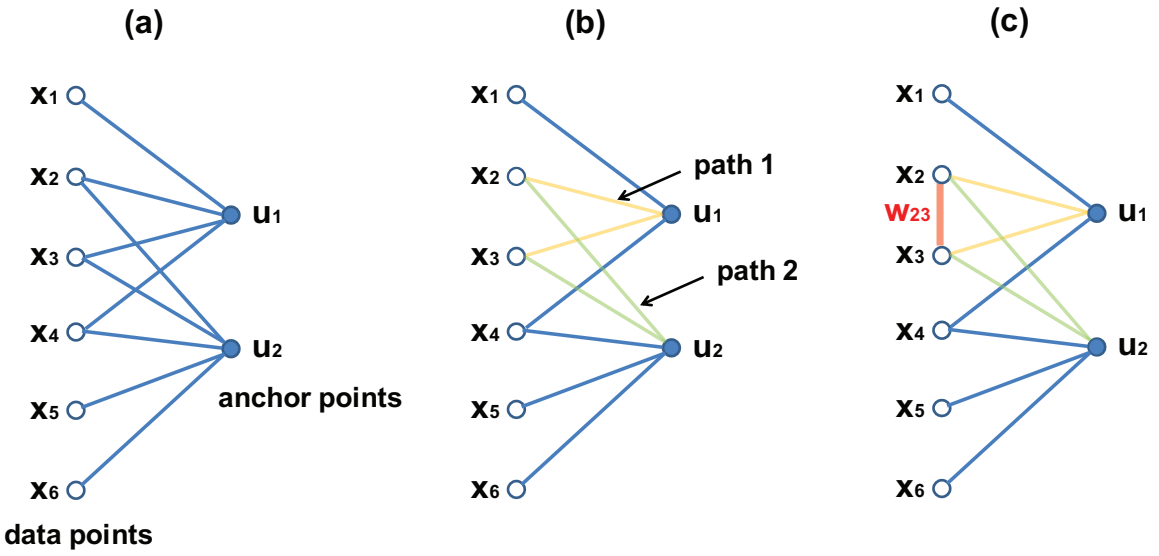


Figure 2.4: Random walks over a data-anchor bipartite graph. (a) The data-anchor bipartite graph $\mathcal{B}(\mathcal{X},\mathcal{U},\mathcal{E},Z)$; (b) $path\ 1 = p^{(1)}(\boldsymbol{u}_1|\boldsymbol{x}_2)p^{(1)}(\boldsymbol{x}_3|\boldsymbol{u}_1)$ and $path\ 2 = p^{(1)}(\boldsymbol{u}_2|\boldsymbol{x}_2)p^{(1)}(\boldsymbol{x}_3|\boldsymbol{u}_2)$; (c) $W_{23} = p^{(2)}(\boldsymbol{x}_3|\boldsymbol{x}_2) = path\ 1 + path\ 2$.

Figure 2.4(a).

Over the introduced bipartite graph $\mathcal{B}$, we establish stationary *Markov random walks* through defining the one-step transition probability matrix $P = (D^B)^{-1}B$ in which $D^B = \text{diag}(B\mathbf{1}) \in \mathbb{R}^{(n+m)\times(n+m)}$ is the diagonal node-degree matrix of the bipartite graph $\mathcal{B}$. By doing so, we obtain the transition probabilities in one time step as follows:

$$p^{(1)}(\boldsymbol{u}_k|\boldsymbol{x}_i) = \frac{Z_{ik}}{\sum_{k'=1}^{m} Z_{ik'}} = Z_{ik},\ p^{(1)}(\boldsymbol{x}_i|\boldsymbol{u}_k) = \frac{Z_{ik}}{\sum_{j=1}^{n} Z_{jk}}$$
$$\forall i \in [1:n],\ \forall k \in [1:m]. \tag{2.17}$$

Obviously, $p^{(1)}(\boldsymbol{x}_j|\boldsymbol{x}_i) = 0$ and $p^{(1)}(\boldsymbol{u}_r|\boldsymbol{u}_k) = 0$ since there are no direct edges connecting them in $\mathcal{B}$. Let us think about the two-step transition probabilities $p^{(2)}(\boldsymbol{x}_j|\boldsymbol{x}_i)$. We have the following theorem to provide an elegant mathematical expression for them.

Theorem 1. *Given one-step transition probabilities defined in eq. (2.17), the transition*

probabilities in two time steps are

$$p^{(2)}(\boldsymbol{x}_j|\boldsymbol{x}_i) = p^{(2)}(\boldsymbol{x}_i|\boldsymbol{x}_j) = \sum_{k=1}^{m} \frac{Z_{ik}Z_{jk}}{\Lambda_{kk}}. \tag{2.18}$$

Proof. Let us exploit the chain rule of Markov random walks and notice $\Lambda_{kk} = \sum_{i=1}^{n} Z_{ik}$, thus deducing as follows

$$\begin{aligned} p^{(2)}(\boldsymbol{x}_j|\boldsymbol{x}_i) &= \sum_{k=1}^{m} p^{(1)}(\boldsymbol{x}_j|\boldsymbol{u}_k)p^{(1)}(\boldsymbol{u}_k|\boldsymbol{x}_i) \\ &= \sum_{k=1}^{m} \frac{Z_{jk}}{\sum_{j'=1}^{n} Z_{j'k}} Z_{ik} = \sum_{k=1}^{m} \frac{Z_{ik}Z_{jk}}{\Lambda_{kk}}, \end{aligned}$$

which does not depend on the order of i and j, so we complete the proof. □

Theorem 1 indicates

$$W_{ij} = \sum_{k=1}^{m} \frac{Z_{ik}Z_{jk}}{\Lambda_{kk}} = p^{(2)}(\boldsymbol{x}_j|\boldsymbol{x}_i) = p^{(2)}(\boldsymbol{x}_i|\boldsymbol{x}_j), \tag{2.19}$$

which interprets the designed adjacency matrix in a probabilistic measure and thereby testifies the correctness of our design for W. For visual explanation, Figures 2.4(b) and (c) display that W_{ij} is the sum of probabilities of all two-step walking paths starting from $\boldsymbol{x}_i$ and arriving on $\boldsymbol{x}_j$.

It is noticeable that we may also define a graph adjacency matrix using the higher-order transition probabilities such as $W'_{ij} = p^{(4)}(\boldsymbol{x}_j|\boldsymbol{x}_i)$, but this leads to a denser adjacency matrix $W' = Z\Lambda^{-1}Z^\top Z\Lambda^{-1}Z^\top$ and increases the computational cost as well.

From a qualitative perspective, the proposed Anchor Graph resembles the classical kNN graph in terms of edge connection structure. On the two-moon toy data, the Anchor Graph with $m = 100$ and $s = 2$, which is really sparse and shown in Figure 2.5(c), is close to the 10NN graph shown in Figure 2.5(b). However, to clarify, the essence of the Anchor Graph is an approximate neighborhood graph since we derive W in an indirect mode for the sake of saving the time complexity. In Anchor Graphs, $W_{ij} > 0$ does not necessarily mean that data pair $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are nearest neighbor to each other.

In summary, the graph adjacency matrix W given by an Anchor Graph is nonnegative, sparse, and low-rank (its rank is at most m). Hence, Anchor Graphs do not need to compute W explicitly, but instead keep the low-rank form. The space cost of an Anchor Graph is

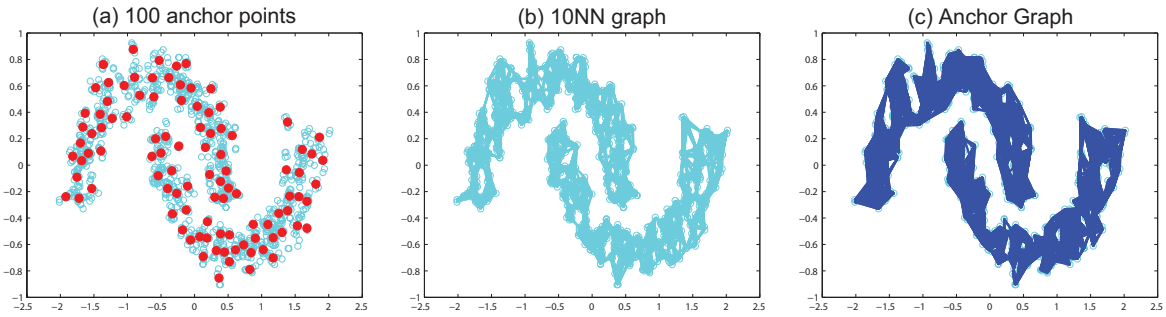


Figure 2.5: The two-moon toy data consisting of 1200 2D points. (a) 100 anchor points from K-means clustering centers; (b) a 10NN graph built on original points; (c) the proposed Anchor Graph with $m = 100, s = 2$ built on original points.

$O(sn)$ for storing Z, and the time cost is $O(dmnT + dmn)$ in which $O(dmnT)$ originates from K-means clustering. Since $m \ll n$, the time complexity for constructing an Anchor Graph is linear in the data size n.

2.5.3 Connection to Low-Rank Matrix Approximation

It is attractive that our proposed Anchor Graph yields a low-rank adjacency matrix $W = Z\Lambda^{-1}Z^\top$, which is in a very similar mathematical formulation to the low-rank approximation $\tilde{K} = K_{nm}K_{mm}^{-1}K_{nm}^\top$ of a positive semidefinite matrix K, typically a kernel matrix, proposed by the well-know Nyström method [193]. While both methods are using the low-rank trick, the motivations are quite different. As clarified in Subsection 2.4.3, our purpose is to design a large graph adjacency matrix W such that the nonnegative and sparse characteristics are explicitly satisfied to make the constructed graph sparse and the resulting graph Laplacian positive semidefinite. In contrast, the Nyström method generates a low-rank matrix $\tilde{K}$ such that the approximation error between this matrix and the original kernel matrix K is kept as small as possible, where only the positive semidefiniteness is imposed on the approximated kernel matrix $\tilde{K}$ whereas nonnegativeness and sparsity are not insured.

The latest progress of the Nyström method either investigated advanced sampling strategies [205][98] to achieve smaller approximation errors, or developed faster variants [97][105] to accelerate the kernel computation under a larger rank ($\geq$10,000). While our Anchor Graph is not trying to approximate any predefined W, we could evaluate its quality according to the error between the Anchor Graph's adjacency matrix and kNN graph's adjacency matrix, since their edge connecting structures are very close as shown in Figures 2.5(b)(c). Consequently, we may explore the error bound in the future. Such a bound should also be dependent on the number of anchors m (i.e., the rank of the Anchor Graph's adjacency matrix) like the error bounds having been proven by the Nyström-related approaches [193][205][97][105].

2.6 Anchor Graph Regularization

Though Anchor Graphs offer approximate neighborhoods, they are found to enjoy similar topological structures to exact neighborhood graphs such as kNN graphs. To testify Anchor Graphs deeply and thoroughly, we must make quantitative evaluations. Accordingly, we hold Anchor Graph-based semi-supervised learning as the testbed and take classification accuracy as a quantitative criterion of Anchor Graph's quality.

Now we develop a novel GSSL algorithm named *Anchor Graph Regularization* (AGR) which establishes a regularized framework upon an Anchor Graph. Let us concentrate on a standard multi-class SSL setting where each labeled sample $\boldsymbol{x}_i$ ($i = 1 \cdots, l$) carries a discrete label $y_i \in \{1, \cdots, c\}$ from c distinct classes. We denote by $Y_l = [\boldsymbol{y}_l^1, \cdots, \boldsymbol{y}_l^c] \in \mathbb{R}^{l \times c}$ a class indicator matrix on the labeled samples $\mathcal{X}_l$ with $(Y_l)_{ij} = 1$ if $y_i = j$ and $(Y_l)_{ij} = 0$ otherwise. Amenable to the aforementioned anchor-based label prediction model, we only need to solve the soft labels associated with m anchors which are put in the label matrix $A = [\boldsymbol{a}_1, \cdots, \boldsymbol{a}_c] \in \mathbb{R}^{m \times c}$ of which each column vector accounts for a class.

We employ the smoothness norm $\|f\|_{\mathcal{G}}^2 = \boldsymbol{f}^\top L \boldsymbol{f}$ introduced in Section 2.1 to impose Anchor Graph regularization on c label prediction functions $\boldsymbol{f}_j = Z\boldsymbol{a}_j$ ($j = 1, \cdots, c$) each

of which is tailored to a single class. Then we formulate a GSSL framework as follows:

$$\begin{aligned}\min_{A=[\boldsymbol{a}_1,\cdots,\boldsymbol{a}_c]} \mathcal{Q}(A) &= \sum_{j=1}^{c} \|Z_l\boldsymbol{a}_j - \boldsymbol{y}_l^j\|^2 + \gamma \sum_{j=1}^{c} \|Z\boldsymbol{a}_j\|_{\mathcal{G}}^2 \\ &= \|Z_lA - Y_l\|_{\mathrm{F}}^2 + \gamma \sum_{j=1}^{c} \boldsymbol{a}_j^\top Z^\top LZ\boldsymbol{a}_j \\ &= \|Z_lA - Y_l\|_{\mathrm{F}}^2 + \gamma \mathrm{tr}(A^\top Z^\top LZA),\end{aligned}$$

where $Z_l \in \mathbb{R}^{l\times m}$ is the sub-matrix according to the labeled partition, $\|.\|_{\mathrm{F}}$ stands for the Frobenius norm, and $\gamma > 0$ is the regularization parameter.

Meanwhile, we compute a "reduced" Laplacian matrix:

$$\begin{aligned}\tilde{L} = Z^\top LZ &= Z^\top(I - Z\Lambda^{-1}Z^\top)Z \\ &= Z^\top Z - (Z^\top Z)\Lambda^{-1}(Z^\top Z),\end{aligned}$$

which is both memory-wise and computationally tractable, taking $O(m^2)$ space and $O(m^3 + m^2n)$ time. Subsequently, we can simplify the cost function $\mathcal{Q}(A)$ to

$$\mathcal{Q}(A) = \|Z_lA - Y_l\|_{\mathrm{F}}^2 + \gamma \mathrm{tr}(A^\top \tilde{L}A). \tag{2.20}$$

With simple algebra, we can easily obtain the globally optimal solution to eq. (2.20):

$$A^* = (Z_l^\top Z_l + \gamma\tilde{L})^{-1} Z_l^\top Y_l. \tag{2.21}$$

As such, we yield a closed-form solution for addressing large scale GSSL. In the sequel, we employ the solved soft labels associated with the anchors to predict the hard label for any unlabeled sample as

$$\hat{y}_i = \arg\max_{j\in\{1,\cdots,c\}} \frac{Z_{i\cdot}\boldsymbol{a}_j}{\tau_j}, \quad i = l+1,\cdots,n, \tag{2.22}$$

where $Z_{i\cdot} \in \mathbb{R}^{1\times m}$ denotes the ith row of Z, and the normalization factor $\tau_j = \mathbf{1}^\top Z\boldsymbol{a}_j$, suggested as a useful *class mass normalization* in the classical GSSL paper [212], balances the possibly skewed class distribution in the labeled data set $\mathcal{X}_l$.

2.6.1 Analysis

The developed AGR algorithm consists of three stages: 1) find anchors by running K-means clustering, 2) design Z, and 3) perform graph regularization. In each stage the space

Table 2.4: Time complexity analysis of the developed scalable GSSL algorithm Anchor Graph Regularization (AGR). n is the data size, m is the number of anchor points, d is the data dimension, s is the number of nearest anchors in LAE, T is the number of iterations in K-means clustering, and T' is the number of iterations in LAE ($n \gg m \gg s$).

Algorithm	**Find Anchors**	**Design Z**	**Graph Regularization**	**Total Time Complexity**
AGR	$O(dmnT)$	$O(dmn)$ or $O(dmn + s^2nT')$	$O(m^3 + m^2n)$	$O(dmnT + m^2n)$

complexity is bounded by $O(dm + dn)$. In the second stage, we may use either a predefined Z in eq. (2.9) or an optimized Z that the LAE algorithm yields. The time complexity of each stage is listed in Table 2.4. We have used a fixed number m ($s \ll m \ll n$) of anchor points for constructing the Anchor Graph and specifying the label prediction model, which is independent of the data size n. Table 2.4 indicates that the total time complexity of AGR is $O(dmnT + m^2n)$, so AGR scales linearly with the data size n.

In order to highlight the differences between our approach AGR and classical GSSL approaches including *Local and Global Consistency* (LGC) [208], *Gaussian Fields and Harmonic Functions* (GFHF) [212], and *Graph Regularization* (GR) [213], we plot Table 2.5 to list several key features of these approaches. It is noted that GFHF is the extreme case of GR as the graph regularization parameter γ is prone to be 0, and that AGR may be thought of as an anchor-approximated version of GR. As far as the time complexity is concerned, LGC, GFHF and GR need to invert sparse and symmetric matrices as large as $n \times n$ in addition to the expensive cost $O(dn^2)$ incurred by kNN graph construction. Since there exists a nearly linear solver for solving sparse and symmetric linear systems [160], one can avoid the expensive matrix inversion operation and apply the sparse linear system solver instead. Thus, we give the lowest time complexity $O\left(c(kn)^{1.31}\right)$ for LGC, GFHF and GR in Table 2.5.

Lastly, we must state that our approach AGR is able to predict labels for any novel data beyond the n data points available in training. This is simply fulfilled by either defining a

Table 2.5: Comparison of three classical GSSL algorithms with AGR. LGC, GFHF and GR use the same kNN graph, while AGR uses the Anchor Graph (m anchors). $F \in \mathbb{R}^{n\times c}$ expresses the target soft label matrix associated with all n data points and c classes, L and $\bar{L}$ respectively denote the graph Laplacian and normalized graph Laplacian matrices of the kNN graph, and $\tilde{L}$ represents the reduced Laplacian matrix of the Anchor Graph.

Algorithm	**Objective Function**	**Parameter** γ	**Time Complexity**	**Handle Novel Data**
LGC [208]	$\min_F \|F-Y\|_{\mathrm{F}}^2 + \gamma \mathrm{tr}(F^\top \bar{L} F)$	$\gamma > 0$	$O(dn^2) + O(c(kn)^{1.31})$	No
GFHF [212]	$\min_F \|F_l - Y_l\|_{\mathrm{F}}^2 + \gamma \mathrm{tr}(F^\top L F)$	$\gamma \to 0$	$O(dn^2) + O(c(kn)^{1.31})$	No
GR [213]	$\min_F \|F_l - Y_l\|_{\mathrm{F}}^2 + \gamma \mathrm{tr}(F^\top L F)$	$\gamma > 0$	$O(dn^2) + O(c(kn)^{1.31})$	No
AGR	$\min_A \|Z_l A - Y_l\|_{\mathrm{F}}^2 + \gamma \mathrm{tr}(A^\top \tilde{L} A)$	$\gamma > 0$	$O(dmnT + m^2 n)$	Yes

nonlinear data-to-anchor mapping (or embedding) $\boldsymbol{z}: \mathbb{R}^d \mapsto \mathbb{R}^m$

$$\boldsymbol{z}(\boldsymbol{x}) = \frac{[\delta_1 \mathcal{K}_h(\boldsymbol{x}, \boldsymbol{u}_1), \cdots, \delta_m \mathcal{K}_h(\boldsymbol{x}, \boldsymbol{u}_m)]^\top}{\sum_{k=1}^m \delta_k \mathcal{K}_h(\boldsymbol{x}, \boldsymbol{u}_k)}, \tag{2.23}$$

where $\delta_k \in \{1, 0\}$ and $\delta_k = 1$ if and only if anchor $\boldsymbol{u}_k$ is one of s nearest anchors in $\mathcal{U}$ of the novel sample $\boldsymbol{x}$; or performing Local Anchor Embedding (formulated in eq. (2.10)) for $\boldsymbol{x}$ to obtain a kernel-free $\boldsymbol{z}(\boldsymbol{x})$. Because of $\boldsymbol{z}(\boldsymbol{x}_i) \equiv [Z_{i1}, \cdots, Z_{im}]^\top$, we can generalize the aforementioned label prediction model in eq. (2.7) to the following universal expression:

$$f(\boldsymbol{x}) = \boldsymbol{z}^\top(\boldsymbol{x})\boldsymbol{a}, \tag{2.24}$$

in which $\boldsymbol{x}$ can be any sample. In the literature, the ability of coping with unseen samples for a SSL algorithm is called *inductive* [216][211]. It has manifested that all of LGC, GFHF and GR are only transductive but not inductive. In contrast, our approach AGR developed in this chapter preferably exhibits the inductive capability besides the high computational efficiency.

2.7 Experiments

In this section, we evaluate the proposed large graph construction approach Anchor Graph on two synthetic toy datasets and three real-word datasets which vary in size from 1,000 to

9,000 samples.

2.7.1 Toy Datasets

In addition to the two-moon toy dataset that we have used to qualitatively test Anchor Graphs in Section 2.5.2, here we try two more toy datasets: the two-ring toy dataset [76] consisting of 1,000 2D points and the two-sun toy dataset of 1,500 2D points. On both datasets, we assign K-means clustering centers to the anchor points with which Anchor Graphs are built. The number m of anchors is chosen to 100 or 200. To construct Anchor Graphs, we use predefined matrices Z and fix the parameter s to 2 on both datasets.

The visual results are shown in Figures 2.6 and 2.7. These results again verify that Anchor Graphs are close to kNN graphs in topology, and that the number of adopted anchors controls the sparsity, possibly quality, of Anchor Graphs. Figures 2.6(d)(f) and 2.7(d)(f) indicate that more anchors used, sparser Anchor Graphs gained.

2.7.2 Real-World Datasets

In this subsection, we provide quantitative experiments to testify the quality of Anchor Graphs in contrast with kNN graphs. Concretely, we use the classification accuracy of graph-based semi-supervised learning (GSSL) carried out on three real-world datasets to evaluate Anchor Graphs as well as Anchor Graph Regularization (AGR).

The compared GSSL algorithms include Local and Global Consistency (LGC) [208], Gaussian Fields and Harmonic Functions (GFHF) [212], and Graph Regularization (GR) [213], all of which exploit kNN graphs. We run K-means clustering in five iterations (i.e, $T = 5$), and the generated clustering centers are taken as anchor points that are fed to construct Anchor Graphs. We run two versions of AGR: 1) AGR with predefined Z, denoted by AGR^0, and 2) AGR with LAE-optimized Z, denoted by AGR. To speedup the construction time of Anchor Graphs, we limit the iterations of LAE to $T' = 10$. The classification experiments are conducted in a standard *transductive* setting as [208][212][184][76][112]: use both labeled and unlabeled samples for learning classifiers and afterwards calculate the classification accuracy over the unlabeled samples. All these experiments are run on a workstation with a 2.53 GHz Intel Xeon CPU and 48GB RAM.

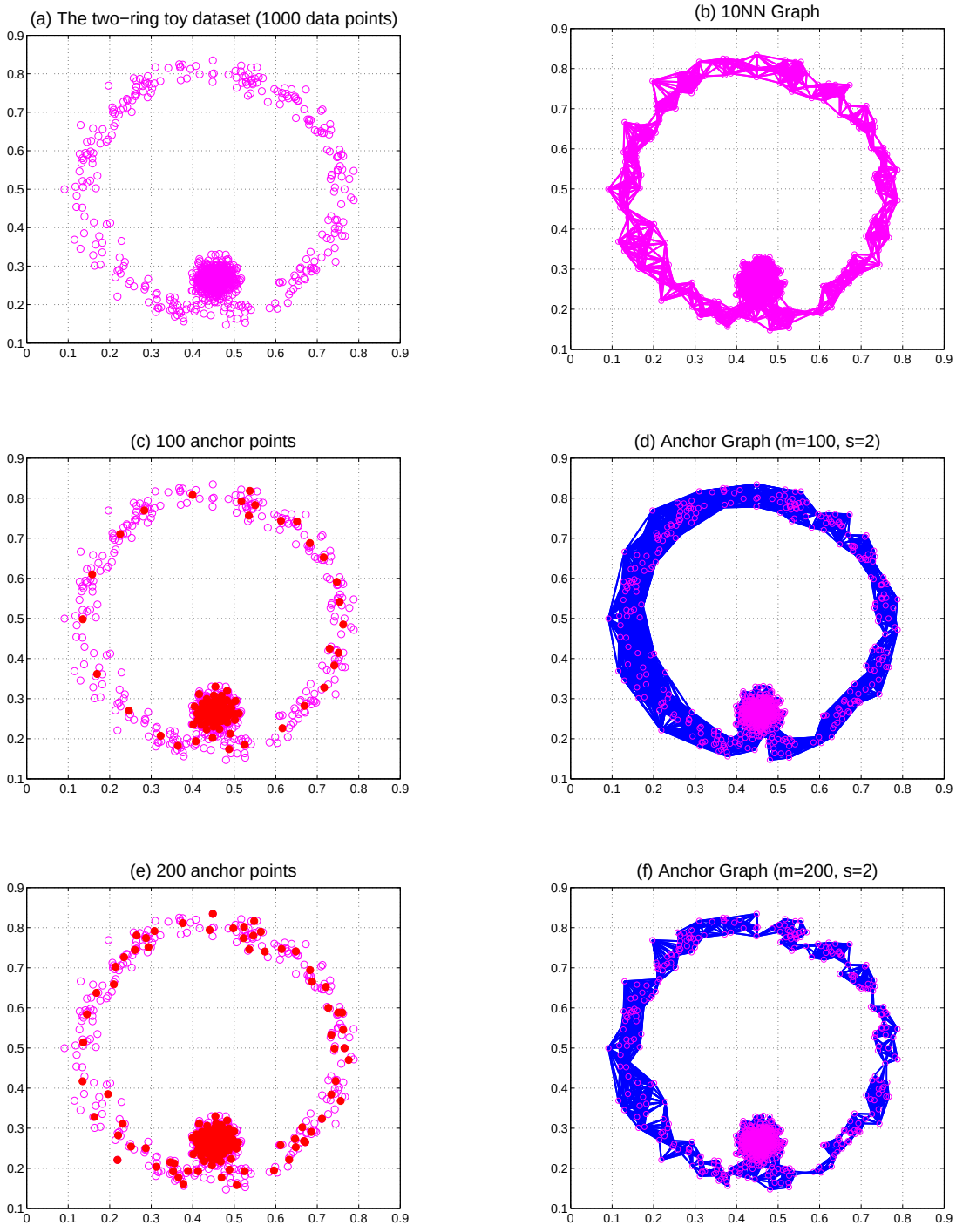


Figure 2.6: Results on the two-ring toy dataset. (a) The original data points; (b) a 10NN graph; (c) 100 anchor points; (d) an Anchor Graph with $m = 100, s = 2$; (e) 200 anchor points; (f) an Anchor Graph with $m = 200, s = 2$.

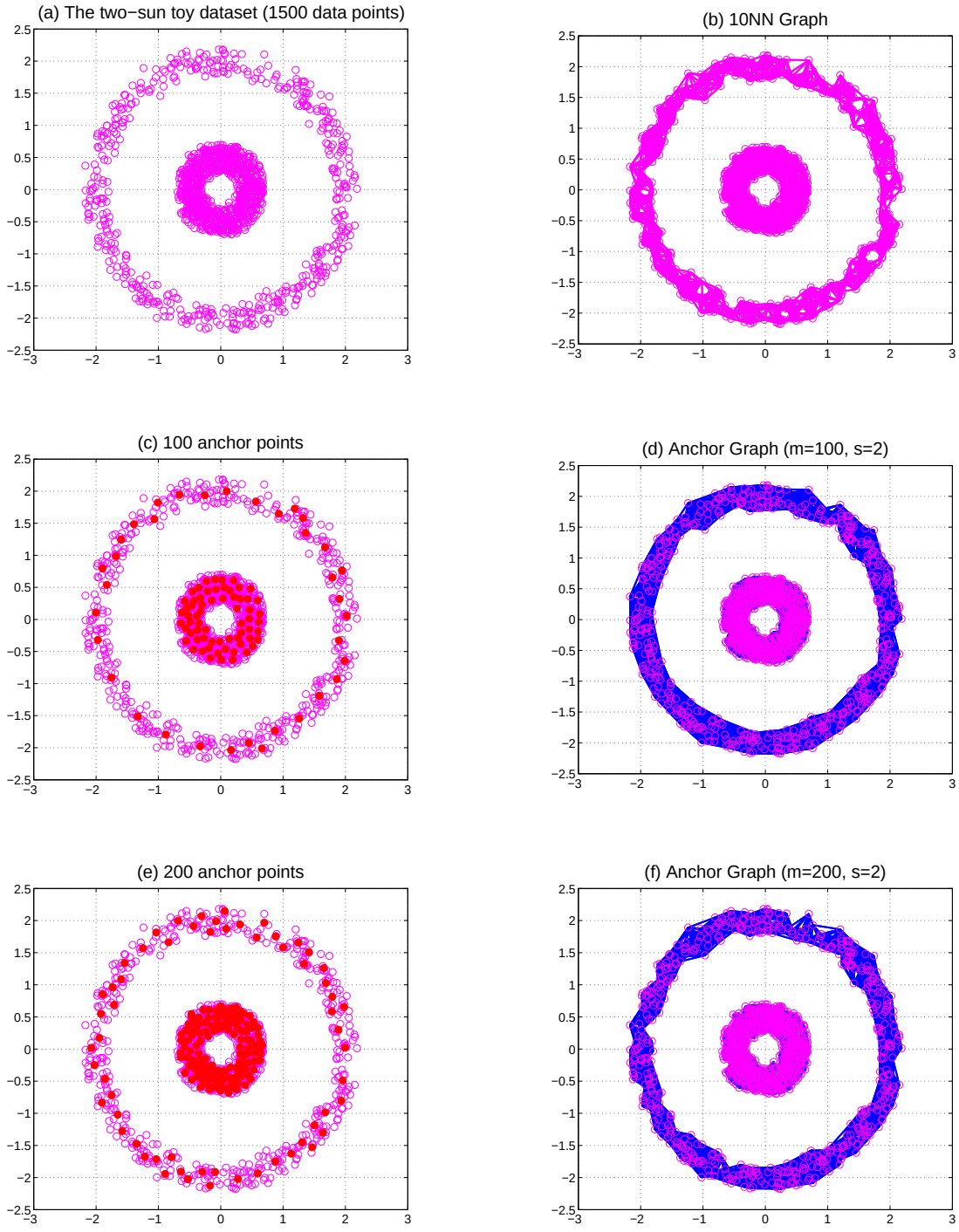


Figure 2.7: Results on the two-sun toy dataset. (a) The original data points; (b) a 10NN graph; (c) 100 anchor points; (d) an Anchor Graph with $m = 100, s = 2$; (e) 200 anchor points; (f) an Anchor Graph with $m = 200, s = 2$.

The three datasets are the **COIL-20** dataset [133] composed of images from 20 object classes, the **SCENE-15** dataset [101] composed of images from 15 natural scene classes, and the **USPS** dataset [1] comprised of images from 10 classes of digits '0', '1', ... ,'9'. Some example images of the three datasets are displayed in Figures 2.8, 2.9, and 2.10.

On **COIL-20** (1,440 samples), to make a SSL environment, we randomly choose $l = 200$ and $l = 400$ labeled samples respectively such that they contain at least one sample from each class. Note that this learning environment introduces skewed class distributions in the labeled samples. The image features of **COIL-20** are in 1,024 dimensions. We do PCA to reduce to 30 dimensions such that higher 1NN classification accuracy is achieved. We evaluate the baseline 1NN, PCA+1NN, three state-of-the-art GSSL algorithms LGC, GFHF and GR using 6NN graphs and 12NN graphs, and AGR^0 and AGR using 20 up to 200 anchors from random exemplars and cluster centers. All GSSL algorithms we are comparing use 30-dimensional PCA features to construct graphs. Averaged over 50 trials, we calculate the classification error rates and report the running time for the referred methods. The results are displayed in Table 2.6 and Figure 2.11. Table 2.6 lists a total running time including three stages K-means clustering, designing Z, and graph regularization for every version of AGR (note that AGR^0 does not do the K-means step). The time cost of graph regularization is quite small and can almost be ignored. From Table 2.6, we know that AGR^0 is much faster than LGC, GFHF and GR, and that AGR is slower because of optimizing Z. However, AGR using 200 anchors achieves comparable classification accuracy to its upper bound GR and GFHF. Figure 2.11 reveals the classification performance of AGR^0 and AGR under various parameters $s \in [2 : 10]$ and $m \in [20 : 200]$. All these results show that 1) the cluster center anchors demonstrate an advantage over the random anchors when using them to build Anchor Graphs, that 2) the increasing anchor size m indeed leads to significant improvement of classification accuracy for every version of AGR, and that 3) AGR with LAE-optimized Z substantially improves the performance of AGR with predefined Z (AGR^0). Actually, Z optimized by LAE induces a sparser adjacency matrix W than predefined rigid Z because Z optimized by LAE is sparser than predefined Z. Using more anchors also leads to sparser W, so the resulting Anchor Graph becomes sparser and closer to the kNN graph. To conclude, we can say that cluster anchors and the geometrical

strategy for designing Z make sense, resulting in high-fidelity Anchor Graphs.

On **SCENE-15** (4,485 samples), we randomly choose $l = 150$ and $l = 600$ labeled samples respectively such that they contain at least one sample from each class. We adopt a 21*1024-dimensional sparse-coding feature vector [189], extracted from dense SIFTs [101], to represent each image in **SCENE-15**. Like **COIL-20**, we do PCA to reduce the very high dimensions to 220 dimensions and subsequently normalize each reduced feature vector to unit ℓ_2 norm. Again, we evaluate the baseline 1NN, PCA+1NN, three state-of-the-art GSSL algorithms LGC, GFHF and GR using 6NN graphs and 12NN graphs, and AGR^0 and AGR using 100 up to 1,000 anchors from random exemplars and cluster centers. All GSSL algorithms in comparison use ℓ_2 normalized 220-dimensional PCA features to construct graphs. Averaged over 50 trials, we report the classification accuracy as well as the running time for the referred methods in Table 2.7 and Figure 2.12. On this larger dataset, both AGR^0 and AGR are faster than LGC, GFHF and GR. AGR^0 and AGR achieve comparable classification accuracy. Figure 2.12 reveals the classification performance of AGR^0 and AGR under various parameters $s \in [2 : 10]$ and $m \in [100 : 1000]$. These results again indicate that for AGR cluster center anchors are more advantageous than random anchors, and that the increasing anchor size m conduces to higher classification accuracy.

On **USPS** (9,298 samples), we randomly choose $l = 100$ and $l = 200$ labeled samples respectively such that they contain at least one sample from each class. The image features are in 256 dimensions which are acceptable, so we do not need to try PCA on this dataset. Like above experiments, we evaluate the baseline 1NN, three state-of-the-art GSSL algorithms LGC, GFHF and GR using 6NN graphs and 12NN graphs, and AGR^0 and AGR using 100 up to 1,000 anchors from random exemplars and cluster centers. Averaged over 50 trials, we report the classification error rates as well as the running time for the referred methods in Table 2.8 and Figure 2.13. Like **SCENE-15**, both AGR^0 and AGR are faster than LGC, GFHF and GR. AGR achieves higher classification accuracy than AGR^0. Figure 2.13 discloses the classification performance of AGR^0 and AGR under various parameter settings $s \in [2 : 10]$ and $m \in [100 : 1000]$. All these results further corroborate that for AGR: i) cluster center anchors are more preferable than random anchors, ii) the growing number of anchors conduce to higher classification accuracy, and iii) AGR using optimized

Figure 2.8: Example images from the **COIL-20** dataset. The original figure is from `http://www1.cs.columbia.edu/CAVE/software/softlib/coil-20.php`.

Z behaves better than using predefined Z.

2.8 Summary and Discussion

The prior GSSL methods scale poorly with the data size because of the quadratic time complexity for neighborhood graph construction, which prevents SSL from being widely applied. This chapter proposes approximate neighborhood graphs, called Anchor Graphs, which are skillfully designed and efficiently constructed over massive data collections, thus making GSSL practical on large-scale problems. Our experiments show that Anchor Graphs exhibit high fidelity to kNN graphs yet with much shorter construction time. Upon Anchor Graphs, the developed novel GSSL algorithm Anchor Graph Regularization (AGR) is simple to understand, easy to implement, and comparable with state-of-the-arts in terms of classification performance. Both time and memory needed by Anchor Graphs as well as AGR grow only linearly with the data size. Therefore, we successfully address the scalabil-

Figure 2.9: Example images from the **SCENE-15** dataset.

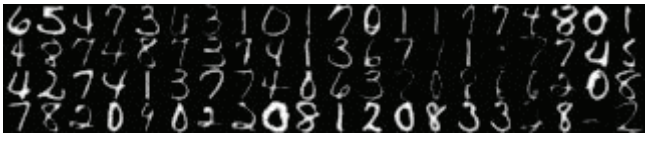


Figure 2.10: Example images from the **USPS** dataset. The original figure is from `http://www.cad.zju.edu.cn/home/dengcai/Data/USPS/images.html`.

Table 2.6: Classification performance on **COIL-20** (1,440 samples). l denotes the number of labeled examples for training GSSL algorithms. All running time is recorded in second. The K-means clustering time for 100 and 200 anchors is 0.049 and 0.083 seconds, respectively. AGR-related methods set $s = 10$. At a fixed l, two lowest error rate values are displayed in boldface type.

	$l = 200$		$l = 400$	
Method	**Error Rate (%)**	**Running Time**	**Error Rate (%)**	**Running Time**
1NN	10.58±1.66	–	4.53±0.90	–
PCA+1NN	8.92±1.49	0.007	3.15±0.93	0.013
LGC with 6NN Graph	3.00±0.93	0.183	1.11±0.61	0.192
LGC with 12NN Graph	8.82±1.15	0.208	6.30±1.43	0.223
GFHF with 6NN Graph	**2.12±0.87**	0.223	**0.75±0.41**	0.208
GFHF with 12NN Graph	7.75±0.98	0.253	4.04±1.00	0.227
GR with 6NN Graph	2.18±0.87	0.163	0.76±0.40	0.167
GR with 12NN Graph	8.57±0.98	0.190	4.87±1.10	0.190
AGR^0 with 100 random anchors	17.62±1.56	0.018	13.65±0.94	0.019
AGR^0 with 200 random anchors	14.56±1.35	0.032	8.81±0.85	0.034
AGR^0 with 100 cluster anchors	17.16±1.46	0.066	12.66±0.99	0.067
AGR^0 with 200 cluster anchors	12.47±1.35	0.114	6.87±0.92	0.116
AGR with 100 random anchors	15.51±1.14	0.262	12.61±1.26	0.289
AGR with 200 random anchors	7.75±1.35	0.270	3.84±0.81	0.291
AGR with 100 cluster anchors	12.73±1.06	0.334	10.49±0.85	0.335
AGR with 200 cluster anchors	**5.16±1.21**	0.370	**1.78±0.67**	0.372

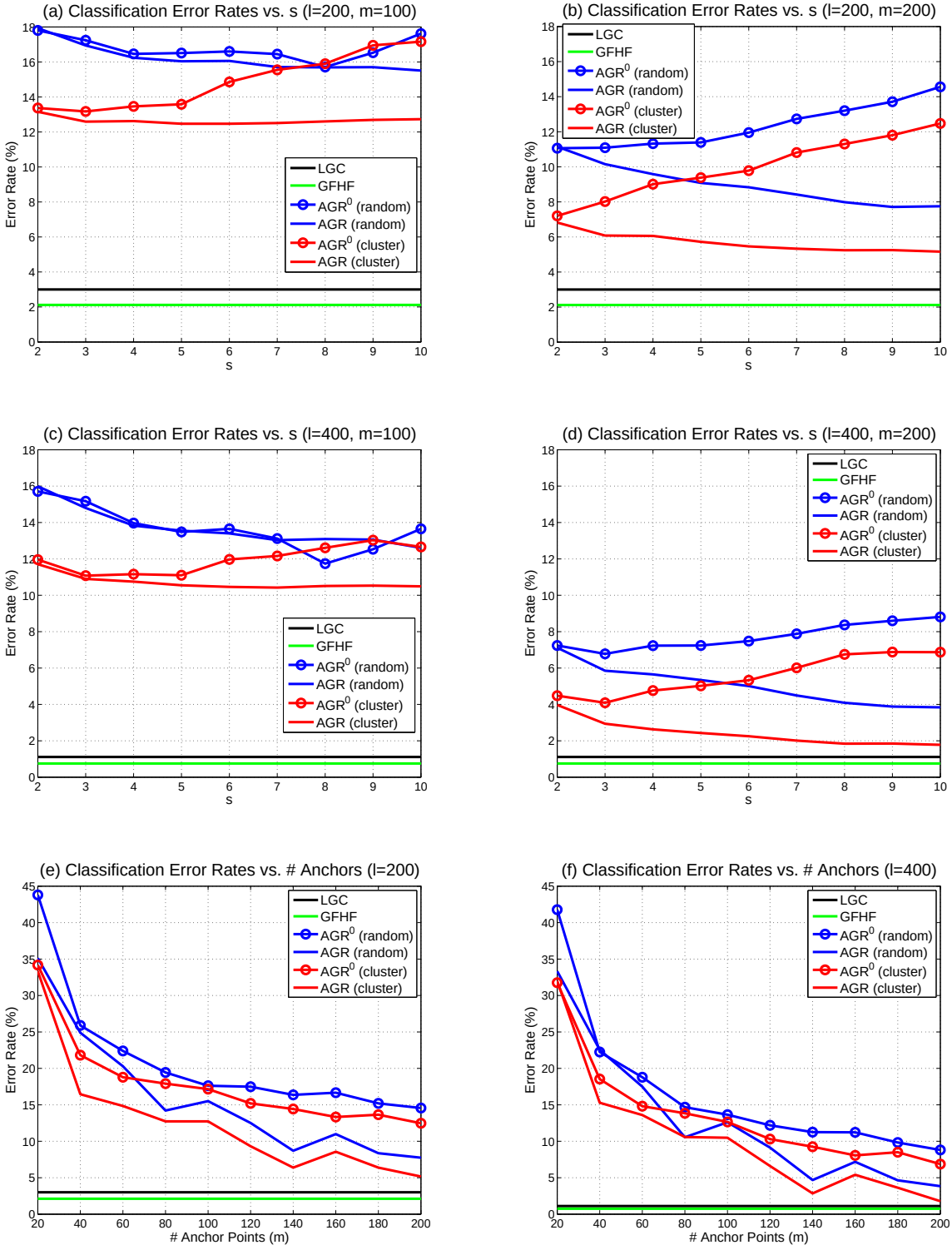


Figure 2.11: Results on **COIL-20**. (a)(b)(c)(d) Classification error rates with increasing s under different l, m settings; (e)(f) error rates with increasing m under $s = 10$.

Table 2.7: Classification performance on **SCENE-15** (4,485 samples). l denotes the number of labeled examples for training GSSL algorithms. All running time is recorded in second. The K-means clustering time for 500 and 1000 anchors is 0.465 and 0.815 seconds, respectively. AGR-related methods set $s = 3$. At a fixed l, two highest accuracy values are displayed in boldface type.

	$l = 150$		$l = 600$	
Method	**Accuracy (%)**	**Running Time**	**Accuracy (%)**	**Running Time**
1NN	54.49±1.56	–	62.49±1.02	–
PCA+1NN	55.76±1.40	0.018	63.86±0.95	0.046
LGC with 6NN Graph	63.80±1.53	2.26	71.12±0.79	2.63
LGC with 12NN Graph	63.50±1.76	2.76	71.22±1.06	3.11
GFHF with 6NN Graph	**66.73±1.03**	1.54	71.93±0.62	1.72
GFHF with 12NN Graph	66.57±1.12	1.94	**72.64±0.66**	2.09
GR with 6NN Graph	66.59±1.16	1.65	71.90±0.54	1.73
GR with 12NN Graph	66.28±1.26	2.00	71.66±0.63	2.14
AGR^0 with 500 random anchors	57.44±1.27	0.200	64.71±0.64	0.209
AGR^0 with 1000 random anchors	58.78±1.42	0.456	65.71±0.59	0.492
AGR^0 with 500 cluster anchors	63.26±1.13	0.665	68.84±0.66	0.674
AGR^0 with 1000 cluster anchors	63.83±1.09	1.27	69.42±0.61	1.31
AGR with 500 random anchors	57.80±1.31	0.401	64.85±0.60	0.410
AGR with 1000 random anchors	59.46±1.37	0.910	65.88±0.61	0.982
AGR with 500 cluster anchors	63.50±1.14	0.866	68.98±0.67	0.875
AGR with 1000 cluster anchors	**64.20±1.07**	1.73	**69.55±0.64**	1.80

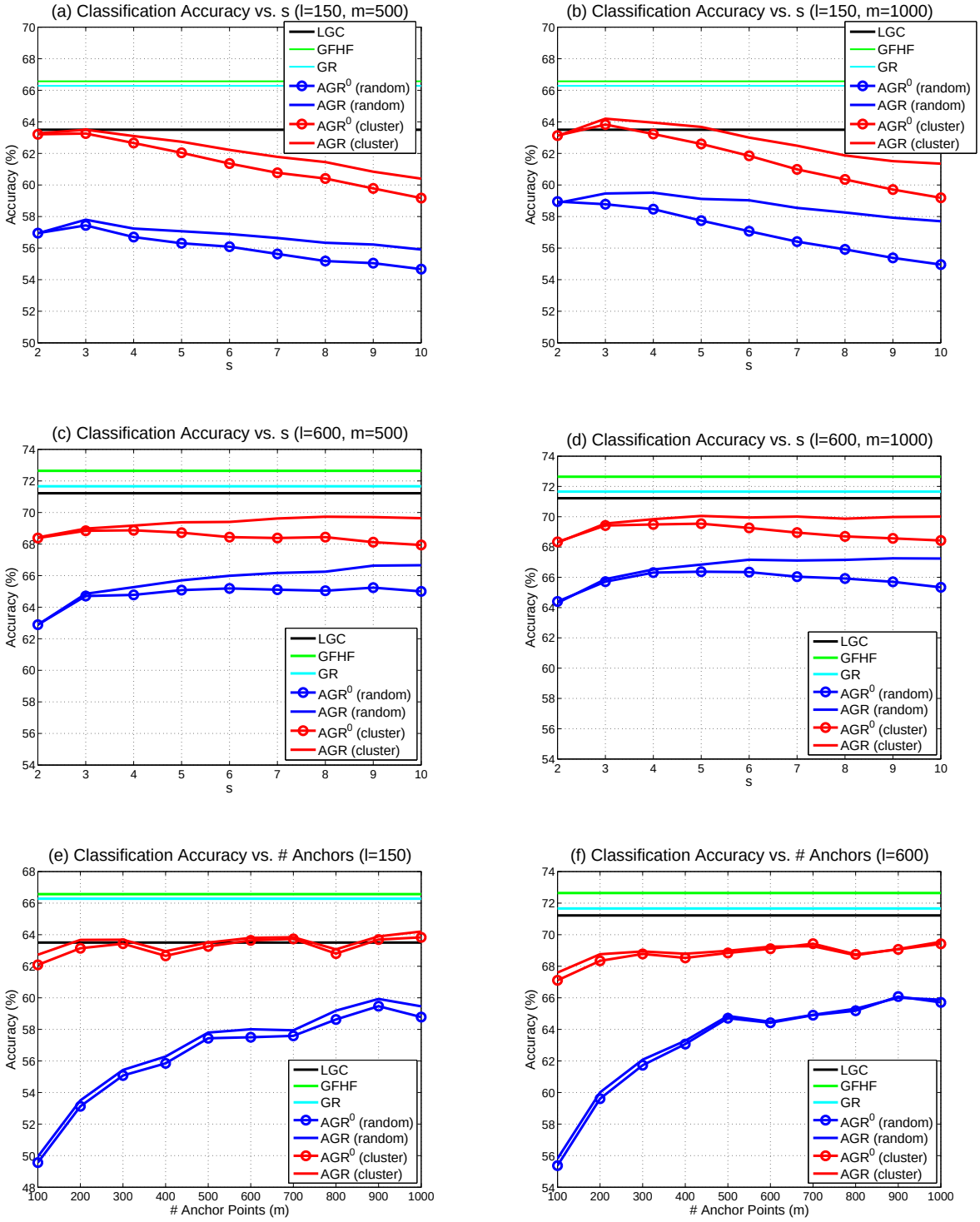


Figure 2.12: Results on **SCENE-15**. (a)(b)(c)(d) Classification accuracy with increasing s under different l, m settings; (e)(f) accuracy with increasing m under $s = 3$.

Table 2.8: Classification performance on **USPS** (9,298 samples). l denotes the number of labeled examples for training GSSL algorithms. All running time is recorded in second. The K-means clustering time for 500 and 1000 anchors is 1.49 and 2.76 seconds, respectively. AGR-related methods set $s = 3$. At a fixed l, two lowest error rate values are displayed in boldface type.

	$l = 100$		$l = 200$	
Method	**Error Rate (%)**	**Running Time**	**Error Rate (%)**	**Running Time**
1NN	21.75±1.41	0.136	16.28±1.11	0.155
LGC with 6NN Graph	8.42±1.25	7.49	6.54±0.78	8.24
LGC with 12NN Graph	9.27±1.34	8.75	7.19±0.85	9.21
GFHF with 6NN Graph	**5.64±0.63**	7.15	**5.04±0.32**	6.50
GFHF with 12NN Graph	6.94±0.75	7.67	6.23±0.40	7.43
GR with 6NN Graph	5.88±0.78	7.30	5.22±0.33	7.36
GR with 12NN Graph	7.40±0.94	7.64	6.55±0.41	7.65
AGR^0 with 500 random anchors	14.41±1.02	0.369	12.49±0.62	0.378
AGR^0 with 1000 random anchors	11.72±0.96	0.670	10.37±0.47	0.707
AGR^0 with 500 cluster anchors	9.09±0.96	1.81	7.78±0.44	1.83
AGR^0 with 1000 cluster anchors	8.04±0.87	3.42	7.07±0.37	3.43
AGR with 500 random anchors	13.56±1.03	1.05	11.79±0.62	1.06
AGR with 1000 random anchors	11.08±0.93	1.07	9.87±0.42	1.08
AGR with 500 cluster anchors	8.56±0.96	2.51	7.35±0.45	2.53
AGR with 1000 cluster anchors	**7.68±0.78**	3.82	**6.82±0.36**	3.83

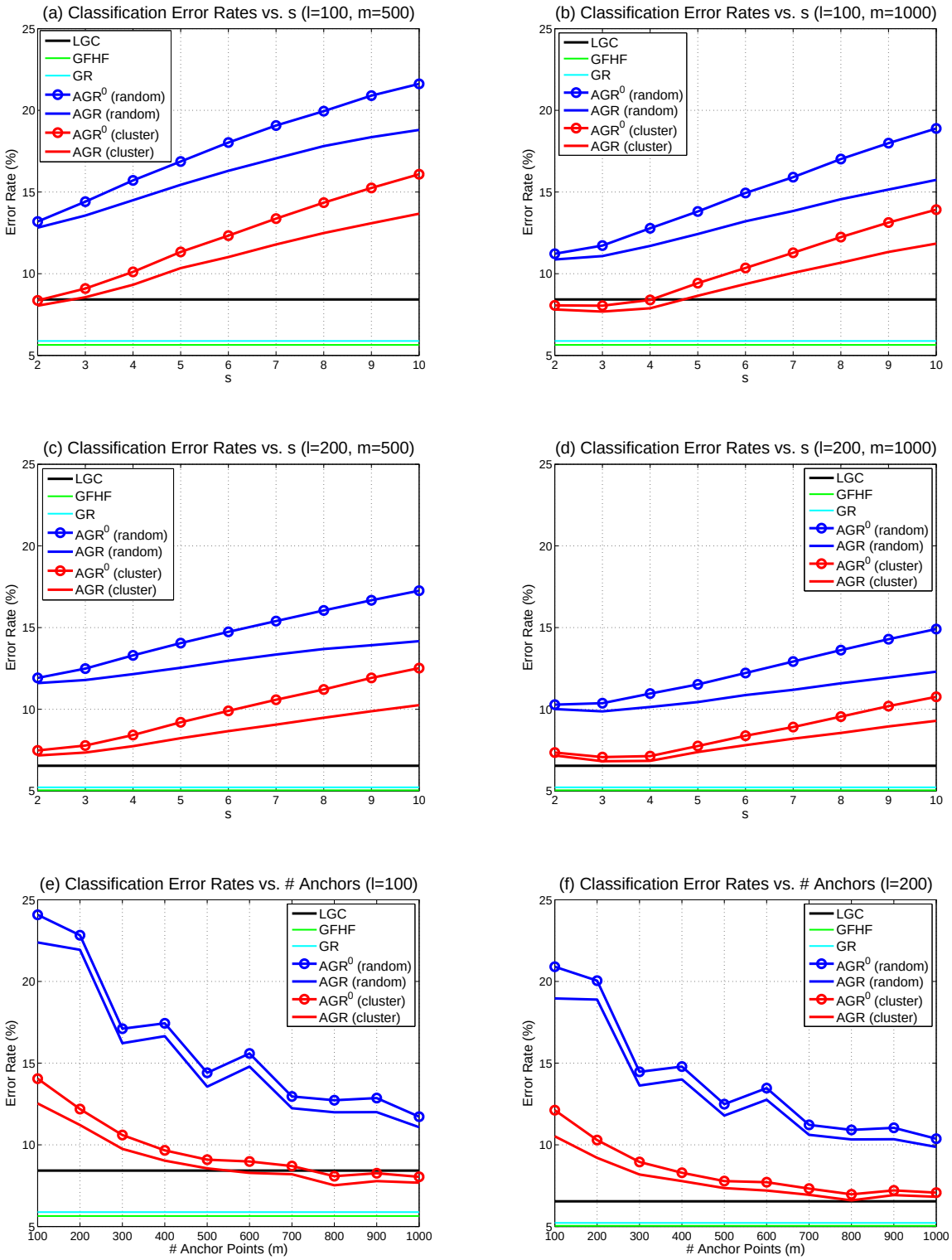


Figure 2.13: Results on **USPS**. (a)(b)(c)(d) Classification error rates with increasing s under different l, m settings; (e)(f) error rates with increasing m under $s = 3$.

ity issue of large graph construction and then make GSSL scalable. In the next chapter, we will apply Anchor Graphs and AGR to larger datasets of up to one million samples.

Having stated in Section 2.6.1, AGR has a natural out-of-sample extension and can easily apply to novel samples once we compute the data-to-anchor mapping $\boldsymbol{z}(\boldsymbol{x})$ for novel sample $\boldsymbol{x}$. For very large datasets (millions or more) K-means clustering may be expensive. To construct Anchor Graphs, we suggest doing clustering on a small subsample of the entire database or trying faster clustering algorithms such as random forest clustering.

For very high-dimensional data, in order to reduce the construction time as well as suppressing noisy data, appropriate dimensionality reduction methods such as PCA, ICA and LDA are strongly suggested before running our Anchor Graphs, referring to what we have done for the **COIL-20** and **SCENE-15** datasets which contain over 1000 dimensional data. What we can do in future is to seek a principled way to further sparsify Anchor Graphs like [161]. Recently, parallel large graph construction such as [187][188] is on the rise. We may engage in setting up Anchor Graphs in a parallel mode, which is very meaningful for faster graph construction over larger scale data.

Lastly, we have to clarify that although in this chapter we select GSSL as the testbed of the proposed large graph construction approach, it is an unsupervised approach in essence and can thus be applied to a large spectrum of machine learning and information retrieval problems. [30] applied Anchor Graphs for large-scale spectral clustering, and [196] made manifold ranking scalable on the foundation of Anchor Graphs, which both witness the power of our proposed Anchor Graphs. Hence, we believe that Anchor Graphs will trigger more applications and make some existing computationally challenging problems tractable. In Chapter 4, we will resolve graph hashing, that was regarded computationally infeasible at a large scale, by employing Anchor Graphs.

Chapter 3

Large-Scale Semi-Supervised Learning

In this chapter, we study large-scale semi-supervised learning deeply. Our purpose is to develop nonlinear and discriminative semi-supervised classifiers from a kernel point of view. We realize that the Anchor Graph Regularization (AGR) algorithm we have developed in the last chapter essentially performs label propagation over Anchor Graphs without optimizing the margins in between different classes. To this end, we aim at learning SVM-like classifiers to maximize the margins. The central idea is to generate a low-rank kernel by leveraging an Anchor Graph into a kernel machine framework. In doing so, the large-scale semi-supervised learning task on all data samples is reduced to a supervised linear classification task carried out on much fewer labeled samples. Therefore, we eventually apply a linear SVM over a new feature space which is derived from decomposing the low-rank kernel. The generated low-rank kernel and its direct linearization succeed in addressing the scalability issue of semi-supervised learning, leading to higher classification accuracy than AGR.

Specifically, we propose three low-rank kernel generation methods. The first method directly takes the low-rank adjacency matrix of the Anchor Graph as a kernel. The second one obtains a low-rank kernel from truncating the pseudo inverse of the Anchor Graph Laplacian on its eigenspectrum. The third one derives a novel kernel through enforcing Anchor Graph based regularization to warp a priori kernel. The resulting Anchor Graph

warped kernel turns out to bear a low-rank expression. Extensive semi-supervised classification experiments performed on three large image datasets of up to one million samples demonstrate the efficacy of the Anchor Graph based low-rank kernel generation techniques. SVMs with the proposed low-rank kernels achieve remarkable performance gains over AGR and state-of-the-art large-scale semi-supervised learning algorithms.

In this chapter, we state the problem background of large-scale semi-supervised learning in Section 3.1, review the related work in Section 3.2, introduce the notations that we will use in Section 3.3, describe the fundamental large-scale learning technique *Kernel Linearization* in Section 3.4, propose our low-rank kernel generation techniques in Section 3.5, show the experimental results in Section 3.6, and finally draw our summary and discussion in Section 3.7.

3.1 Problem Background

In the current age of data divulgence, there is emerging attention in leveraging massive amounts of data available in open sources such as the Web to help solve long standing computer vision, data mining, and information retrieval problems like object recognition, topic detection and discovery, multimedia information retrieval, community detection, collaborative filtering, and so on. How to effectively incorporate and efficiently exploit large-scale data corpora is an open problem.

In this chapter, we further focus on the promising direction *Semi-Supervised Learning* (SSL) [23][211][213] which has gained broad interest: developing the best ways of combining labeled data, often of limited amount, and a huge pool of unlabeled data in forming abundant training resources for optimizing machine learning models. Let us quote one typical application of SSL. Nowadays, massive amounts of data, e.g., images, videos, audio, documents, etc., are present on the Web and may thus be crawled from the Web in demand. For example, images associated with a desired tag, customarily a semantic label, can be gathered via searching this tag at web image search engines such as Google and Bing, or photo sharing websites such as Flickr and Fotolog. However, the large number of images retrieved by search engines are usually noisy and frequently do not match the user-provided

tag entirely [48]. Therefore, users are suggested to give accurate labels for some images and the labels of the rest of images will be predicted by applying an appropriate semi-supervised classifier.

The challenging issue pertaining to SSL for web-scale applications is scalability – how to successfully accommodate millions or billions of data samples in a learning framework. Unfortunately, most SSL methods scale poorly with the data size and become intractable in computation for large scale learning tasks. As far as the popular graph-based SSL (GSSL) algorithms are concerned, they require a quadratic time complexity $O(dn^2)$ for neighborhood graph construction (suppose that n data points live in $\mathbb{R}^d$) and $O(n^\rho)$ $(1 < \rho \leq 3)$ for classifier training over graphs. Hence, the overall time complexity remains $O(dn^2)$ at least. Such an expensive cost is computationally prohibitive for large scale applications, severely preventing the adoption of SSL in practical situations.

Through taking advantage of Anchor Graphs proposed in the previous chapter, we are able to keep a reduced linear computational complexity $O(dn)$ for the graph construction step. Actually, we can benefit further from the Anchor Graphs, leveraging them to develop efficient semi-supervised classifiers which are trained in linear time as well. We have achieved a linear time semi-supervised classifier, i.e., the Anchor Graph Regularization (AGR) algorithm developed in Chapter 2, but it is not discriminative enough to separate multiple classes since AGR essentially performs label propagation over Anchor Graphs without optimizing the margins in between different classes. In this chapter, we would like to do better and develop more discriminative semi-supervised classifiers. A natural idea is to create a kernel machine for margin maximization and make it accommodate the semi-supervised scenario. Motivated from the kernel linearization technique [143] which is critical to scaling up traditional kernel machines, our approach generates and linearizes several low-rank kernels from and with an Anchor Graph. The linearized low-rank kernels give rise to novel feature spaces over which linear SVMs can easily be trained on much fewer labeled samples. Consequently, the challenging large-scale SSL problem is addressed via solving a small-scale linear classification problem. In one sentence, our low-rank kernels are yielded from the kernel viewpoints of Anchor Graphs.

3.2 Related Work

Semi-supervised learning (SSL), a modern machine learning paradigm, deals with learning tasks through utilizing both labeled and unlabeled examples. The learning tasks allow various scenarios including semi-supervised classification [212][208], semi-supervised regression [100][80], and semi-supervised clustering [9][92]. In such learning tasks, one is confronted with the situations where a few labeled data together with large amounts of unlabeled data are available. SSL has been increasingly popular in a lot of practical problems, since it is quite feasible to obtain unlabeled data by an automatic procedure but quite expensive to identify the labels of data.

This chapter concentrates on semi-supervised classification whose paramount foundation necessitates an appropriate assumption about data distributions. Two commonly adopted assumptions are the *cluster assumption* [25] and the *manifold assumption* [12]. The former assumes that samples associated with the same structure, typically a cluster, tend to take similar labels. The latter often implies that close-by samples on data manifolds are very likely to share close labels. Notice that the cluster assumption is assumed in a global view whereas the manifold assumption is often imposed locally.

Among current research on semi-supervised classification, [25][26] implemented the cluster assumption which favors decision boundaries for classification passing through low-density regions in the input sample space. In line with them, Transductive Support Vector Machines (TSVMs) [83][156][36][24] aimed to optimize the margins among both labeled and unlabeled examples. A bunch of GSSL algorithms [212][208][184][112][12][157][124] put forward various graph-based learning frameworks with similar graph regularization terms. Besides these, numerous approaches in the literature have exploited the manifold assumption to pursue smooth classification or prediction functions along data manifolds which were represented by graphs. It is feasible to integrate the two assumptions to develop stronger SSL models like [87][29]. There also exist many other methods engaging in semi-supervised classification from other perspectives or with other assumptions, such as semi-supervised generative models [43][165] and semi-supervised boosting [122][29].

With rapid development of the Internet, now we can collect massive (up to hundreds of millions) unlabeled data such as images and videos, and then the need for large scale SSL

arises. Unfortunately, most SSL methods scale poorly with the data size n. For instance, the classical TSVM [83] is computationally challenging, scaling exponentially with n. Among various versions of TSVM, the Concave-Convex Procedure (CCCP)-TSVM [36] has the lowest complexity, but it scales as at least $O(n^2)$ so it is still difficult to scale up. As for GSSL which is appealing in use and implementation, it incurs a quadratic to cubic time complexity $O(n^2) \sim O(n^3)$ because of neighborhood graph construction and nontrivial manipulations upon large $n \times n$ graph Laplacian matrices.

To temper the high time complexity of GSSL, recent studies seek to reduce the intensive computations upon graph Laplacians. A few solutions have been proposed recently. [40] proposed a nonparametric inductive function for label prediction based on a subset of samples and aggressive truncation in calculating the graph Laplacian. However, the truncation sacrifices the topology structure within the majority of input data and hence will likely lose useful information of the data set. [216] fitted a generative mixture model to the raw data and proposed harmonic mixtures to span the label prediction function, but the key step, i.e., constructing a large sparse graph, needed in estimating the harmonic mixtures remains open. [176] scaled up the manifold regularization method first proposed in [12] by solving the dual optimization problem of manifold regularization subject to a sparsity constraint, but such optimization still requires heavy computation (taking $O(1/\epsilon^8)$ time where $\epsilon > 0$ is the approximation factor) to achieve a good approximate solution. [204] applied the Nyström approximation to build a huge graph adjacency matrix, but there is no guarantee for the positive semidefiniteness of the resulting graph Laplacian, which is required to ensure convexity of the optimization problem and convergence of the solution. [49] approximated the label prediction function by linearly combining smooth eigenfunctions of 1D graph Laplacians calculated from each dimension of data, whose derivation relies on several assumptions about the data, e.g., dimension independence and 1D uniform distributions, which are not true for real-world data.

Almost all semi-supervised learning methods can be categorized to two families: *transductive* and *inductive*. The former aims to infer labels of unlabeled data without developing an explicit classification model, thus lacking the capability of dealing with novel data. The latter takes advantage of both labeled and unlabeled data to train classification mod-

els (i.e., inductive models) which can be used to handle unseen data outside the training data set. Consequently, inductive semi-supervised learning is referred to as *truly* semi-supervised learning [157]. Several classical GSSL methods including GFHF [212], LGC [208] and GTAM [184] are purely transductive, and other methods such as graph min-cuts [15][16][99], spectral graph partitioning [84], random walks [167][7], and local learning [194] also belong to the transductive family since they focus on predicting information associated with the existing unlabeled data. The inductive family consists of all versions of TSVMs [83][36][24], manifold regularization [12][157][124], multi-kernel semi-supervised learning [171], translated kernel logistic regression [147], etc. The recent advance in inductive semi-supervised learning explores semi-supervised relevance ranking [199], structured output semi-supervised learning [125], and multi-label semi-supervised learning [200].

For the scalability issue, most purely transductive methods are not suitable solutions except the Blockwise Supervised Inference method [202] which, however, made a restrictive assumption that the data graph has a block structure. Because of the capability of handling novel data, scalable and inductive semi-supervised learning is more desirable for real-world web-scale data, which usually anticipates dynamic novel data.

The scalable GSSL work mentioned before, including Nonparametric Function Induction [40], Harmonic Mixtures [216], Sparsified Manifold Regularization [176], Prototype Vector Machines [204], and Eigenfunction Combination [49], is actually inductive. As described and analyzed in Chapter 2, our approach Anchor Graph Regularization (AGR) is also inductive and scalable. Akin to AGR, Nonparametric Function Induction, Harmonic Mixtures, and Prototype Vector Machines all exploit the idea of "*anchors*". Table 3.1 summarizes and conceptually compares four scalable yet inductive GSSL models using the anchor idea, most of which can reach a training time complexity $O(m^3 + m^2 n)$ and all of which can attain a test time complexity $O(dm)$ (m is the number of adopted anchors).

To gain a deeper understanding, we point out that besides being applied to construct Anchor Graphs or approximate large kernel matrices (the Nyström method [193]), the idea of anchors is also related to the concept of randomly subsampled "*beacons*" in large-scale network analysis [88], in which node distances are inferred based on the triangulation over the distances between nodes and beacons. In the speech recognition community, the

Table 3.1: Summary of anchor-based GSSL models which share the same form of classification functions $f(\boldsymbol{x}) = \sum_{k=1}^{m} Z_{xk} a_k$. Z_{xk} denotes the affinity between data point $\boldsymbol{x}$ and anchor $\boldsymbol{u}_k$, $S(\cdot)$ is some similarity function, $\kappa(\cdot)$ is a kernel function, and $p(\mathcal{C}_k|\boldsymbol{x})$ is the posterior probability of $\boldsymbol{x}$ assigned to cluster $\mathcal{C}_k$.

GSSL Model	**Anchors** $\{\boldsymbol{u}_k\}$	$\{a_k\}$	**Affinities** $\{Z_{xk}\}$
Nonparametric Function Induction [40]	exemplars	labels on anchors	$\frac{S(\boldsymbol{x}, \boldsymbol{u}_k)}{\sum_{k'=1}^{m} S(\boldsymbol{x}, \boldsymbol{u}_{k'})}$
Harmonic Mixtures [216]	GMM clustering centers	labels on anchors	$p(\mathcal{C}_k\|\boldsymbol{x})$
Prototype Vector Machines [204]	K-means clustering centers	classifier coefficients	$\kappa(\boldsymbol{x}, \boldsymbol{u}_k)$
Anchor Graph Regularization (Chapter 2)	K-means clustering centers	labels on anchors	$\frac{S(\boldsymbol{x}, \boldsymbol{u}_k)}{\sum_{k'=1}^{m} S(\boldsymbol{x}, \boldsymbol{u}_{k'})}$

idea of anchor models has also been employed to prune the number of speaker models needed by the applications in speaker detection and speaker indexing on a large database, achieving a promising tradeoff between detection accuracy and computational efficiency [164]. One important distinction between the prior anchor work and the anchor-based SSL models discussed currently is that the anchor-based SSL models aims at inferring the label information of a large number of original data points rather than pursuing the minimal-distortion reconstruction of the entire network.

3.3 Notations

In this section, we first define the notations and symbols that we will use to describe our approaches in the rest of this chapter. All notations as well as their definitions are listed in Tables 3.2 and 3.3.

Table 3.2: Table of notations.

Notation	Definition
n	The number of data points
l	The number of labeled data
m	The number of anchor points
d	The dimension of data or anchor points
i, j	The indices of data points
k	The index of anchor points
$\boldsymbol{x}_i \in \mathbb{R}^d$	The ith data point
$\boldsymbol{u}_k \in \mathbb{R}^d$	The kth anchor point
$\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^n$	The data set
$\mathcal{X}_l = \{\boldsymbol{x}_i\}_{i=1}^l$	The labeled data set
$\mathcal{U} = \{\boldsymbol{u}_k\}_{k=1}^m$	The anchor set
c	The number of classes
$y_i \in [1:c]$	The class label of $\boldsymbol{x}_i$, $i \in [1:l]$
$\hat{y}_i \in [1:c]$	The estimated class label of $\boldsymbol{x}_i$, $i \in [l+1:n]$
$\mathcal{G}(\mathcal{X}, E, W)$	The Anchor Graph
$E \subseteq \mathcal{X} \times \mathcal{X}$	The set of edges in $\mathcal{G}$
$W = (W_{ij})_{i,j} \in \mathbb{R}^{n \times n}$	The weighted adjacency matrix of $\mathcal{G}$
$Z = (Z_{ij})_{i,j} \in \mathbb{R}^{n \times m}$	The data-to-anchor affinity matrix between $\mathcal{X}$ and $\mathcal{U}$
$\Lambda = \operatorname{diag}(Z^\top \mathbf{1}) \in \mathbb{R}^{m \times m}$	The diagonal anchor-degree matrix of $\mathcal{G}$
s	The number of nearest anchors in $\mathcal{U}$ for each data point
$\langle i \rangle \subset [1:m]$	The set of indices of s nearest anchors in $\mathcal{U}$ for $\boldsymbol{x}_i$
$\mathcal{K}_h : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$	A kernel function with bandwidth h
$L \in \mathbb{R}^{n \times n}$	The Anchor Graph Laplacian matrix
$L^+ \in \mathbb{R}^{n \times n}$	The pseudo inverse of L
r	The truncation rank of L^+
$(\boldsymbol{p}_k, \lambda_k)$	The kth eigenvector-eigenvalue pair of L
$P \in \mathbb{R}^{n \times r}$	The eigenvector matrix of L
$\|\cdot\|_{\mathcal{G}}$	The Anchor Graph Laplacian regularization norm

Table 3.3: Table of notations (continued).

Notation	Definition
$\mathcal{K}^W : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$	The Anchor Graph kernel function
$\mathcal{K}^L : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$	The Anchor Graph Laplacian kernel function
$\kappa : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$	A priori kernel function
$\tilde{\kappa} : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$	The Anchor Graph warped kernel function
$K_{\mathcal{U}\mathcal{U}} \in \mathbb{R}^{m \times m}$	The kernel matrix between $\mathcal{U}$ and $\mathcal{U}$
$K_{\mathcal{X}\mathcal{U}} \in \mathbb{R}^{n \times m}$	The kernel matrix between $\mathcal{X}$ and $\mathcal{U}$
$\boldsymbol{z}(\boldsymbol{x}) \in \mathbb{R}^m$	The data-to-anchor mapping
$\mathbf{F}(\boldsymbol{x}) \in \mathbb{R}^m$	An explicit feature mapping
$f : \mathbb{R}^d \mapsto \mathbb{R}$	A classification function
$\boldsymbol{f} \in \mathbb{R}^n$	The output vector of f on all data points $\mathcal{X}$
$\boldsymbol{a} \in \mathbb{R}^m$	The coefficient vector on anchor points $\mathcal{U}$
$\mathcal{Q}(\boldsymbol{a}) \in \mathbb{R}$	The objective function for optimizing $\boldsymbol{a}$
$Loss(\boldsymbol{x}_i, y_i) \in \mathbb{R}$	A loss function defined on labeled samples $(\boldsymbol{x}_i, y_i)$
$\mathcal{H}_\kappa$	The RKHS induced by κ
$\|\cdot\|_{\mathcal{H}_\kappa}$	The RKHS regularization norm
$\beta > 0$	The RKHS regularization parameter
$\gamma > 0$	The graph regularization parameter

3.4 Kernel Linearization

The challenging issue that both supervised learning and semi-supervised learning suffer from is scalability: how to successfully accommodate millions up to billions of data samples in learning models. Unfortunately, most traditional machine learning models scale poorly with the data size, thus blocking widespread applicability to real-life problems that encounter growing amounts of data.

In this section, we consider the widely used kernel machines and introduce a state-of-the-art technique *Kernel Linearization* [143] which has been corroborated to be able to scale up a kernel machine with a *shift-invariant* kernel at the training expense of a linear time complexity $O(n)$.

[143] proposed to approximate a shift-invariant kernel function $\kappa : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ which

will be plugged into a kernel machine such as a kernel *Support Vector Machine* (SVM) or a kernel *Support Vector Regression* (SVR) [149]. Critically, such an approximation attempts to *linearize* the nonlinear kernel κ as the inner product between two feature vectors:

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') \approx \mathbf{F}^\top(\boldsymbol{x})\mathbf{F}(\boldsymbol{x}'), \tag{3.1}$$

where $\mathbf{F} : \mathbb{R}^d \mapsto \mathbb{R}^m$ is an *explicit feature mapping* that yields *special* features for an original data sample $\boldsymbol{x}$. Once the dimension m of the yielded features is much smaller than the total size n of the input data set $\mathcal{X}$, the kernel machine can be converted to a linear machine via taking $\{\mathbf{F}(\boldsymbol{x}_i)\}_{i=1}^n$ as new training samples to train a linear SVM or a linear SVR in linear time.

The kernel linearization technique has exhibited promising performance: the classification accuracy of a linearized kernel SVM does not drop much apart from that of a kernel SVM, while the training time of a linearized kernel SVM is several orders of magnitude faster than that of a kernel SVM. As an extension, [178][104] developed more feature maps to approximate several shift-variant kernels. It is worth while pointing out that all these explicit feature maps are *data-independent*. They are adequate for supervised learning problems that encounter enough labels, but could lack to some extent for semi-supervised learning problems where labels are usually scarce. Therefore, we intend to develop *data-dependent* feature maps in what follows.

3.5 Generating Low-Rank Kernels

In this section, we address the scalability issue that GSSL suffers from and propose several key techniques to establish scalable kernel machines under the semi-supervised scenario. Importantly, the proposed techniques take advantage of Anchor Graphs from a kernel point of view. While some linear machines such as linear TSVMs [155][123] are scalable, we prefer developing nonlinear kernel machines because kernel-based classifiers have been theoretically and empirically proven to be able to tackle practical data that are mostly linearly inseparable.

Above all, let us define a low-rank kernel which leads to a nonlinear classifier and simultaneously enables linear time training of the classifier.

Definition 2. *A low-rank kernel function* $\mathcal{K} : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ *satisfies*

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = \mathbf{F}^\top(\boldsymbol{x})\mathbf{F}(\boldsymbol{x}'), \tag{3.2}$$

where $\mathbf{F}(\boldsymbol{x})$ *is some vectored representation with* m *dimensions smaller than the total number* n *of data samples.*

Using a low-rank kernel $\mathcal{K}$ for training a nonlinear classifier f is equivalent to using the new data representation $\{\mathbf{F}(\boldsymbol{x}_i)\}_{i=1}^n$ for training a linear classifier:

$$f(\boldsymbol{x}) = \boldsymbol{w}^\top \mathbf{F}(\boldsymbol{x}), \tag{3.3}$$

in which $\boldsymbol{w} \in \mathbb{R}^m$ is the weight vector to be learned from the l labeled examples $\{\mathbf{F}(\boldsymbol{x}_i)\}_{i=1}^l$. Since $l \ll n$ in most semi-supervised learning tasks, in order to alleviate the overfitting issue, we need to deliver a "rich" feature map $\mathbf{F}(\boldsymbol{x})$ such that information from abundant unlabeled examples is absorbed into it. In the following, we generate meaningful low-rank kernels by exploring the underlying data structure, deriving data-dependent feature maps via simple kernel linearization.

3.5.1 Anchor Graph Kernels

Here we propose the first low-rank kernel which is exactly the low-rank adjacency matrix $W = Z\Lambda^{-1}Z^\top$ of the Anchor Graph $\mathcal{G}(\mathcal{X}, E, W)$. By utilizing the data-to-anchor mapping $\boldsymbol{z}(\cdot)$ defined in eq. (2.23) of Chapter 2, we give this low-rank kernel as follows

$$\mathcal{K}^W(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{z}^\top(\boldsymbol{x})\Lambda^{-1}\boldsymbol{z}(\boldsymbol{x}'), \tag{3.4}$$

which we name the *Anchor Graph kernel* since $\mathcal{K}^W(\boldsymbol{x}_i, \boldsymbol{x}_j) = W_{ij}$. Below we give a straightforward proposition, realizing $\Lambda > 0$.

Proposition 3. $\mathcal{K}^W$ *defined in eq. (3.4) is a low-rank kernel.*

In spirit, our Anchor Graph kernel is very similar to the *Random Walk kernel* [190], and can be seen as its anchor-approximated version. In a Random Walk kernel $\mathcal{K}^{rand}$,

$\mathcal{K}^{rand}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{p}^\top(\boldsymbol{x}_i)\boldsymbol{p}(\boldsymbol{x}_j)$ in which $\boldsymbol{p}(\boldsymbol{x}_i) \in \mathbb{R}^n$ represents a vector whose n entries constitute a transition probability distribution of random walks from $\boldsymbol{x}_i$ to all data points in $\mathcal{X}$. Because $\mathbf{1}^\top \boldsymbol{z}(\boldsymbol{x}) = 1$ for any sample $\boldsymbol{x}$, $\boldsymbol{z}(\boldsymbol{x}_i) \in \mathbb{R}^m$ also constructs a transition probability distribution of random walks from $\boldsymbol{x}_i$ to all anchor points in $\mathcal{U}$. Thus, the Anchor Graph kernel can be deemed as the inner product between two probability distributions of random walks across data and anchors, with proper normalization.

3.5.2 Anchor Graph Laplacian Kernels

[85][5] suggested that the pseudo inverse matrices of graph Laplacians are good data-dependent kernels for training semi-supervised kernel machines. As graph Laplacians are positive semidefinite, their pseudo inverse matrices inherit the positive semidefiniteness and thus become valid kernel matrices. Following [85][5], we can derive another low-rank kernel function through inverting the Anchor Graph Laplacian matrix.

In what follows, we first obtain the eigenvectors of the Anchor Graph Laplacian L and then calculate the pseudo inverse L^+. To do that, we give Proposition 4.

Proposition 4. *If a matrix $B^\top B$ has an eigenvalue system $\{(\boldsymbol{v}_k, \sigma_k)\}_{k=1}^r$ ($\sigma_k > 0$) in which $\boldsymbol{v}_k$ is the normalized eigenvector corresponding to eigenvalue σ_k, then the matrix $BB^\top$ has an eigenvalue system $\{(B\boldsymbol{v}_k/\sqrt{\sigma_k}, \sigma_k)\}_{k=1}^r$ in which $B\boldsymbol{v}_k/\sqrt{\sigma_k}$ is the normalized eigenvector corresponding to eigenvalue σ_k.*

Proposition 4 is one of basic properties of eigenvalue systems of matrices [55].

Since $L = I - W$, the Anchor Graph Laplacian eigenvectors $\{\boldsymbol{p}_k\}_k$ are also eigenvectors of $W = Z\Lambda^{-1}Z^T$, but the Anchor Graph Laplacian eigenvalues $\{\lambda_k\}_k$ equal $1 - \sigma_k$ with $\{\sigma_k\}_k$ being the eigenvalues of W. Afterwards, one can easily solve the eigen-system $\{(\boldsymbol{p}_k, \sigma_k)\}_k$ of W by utilizing W's low-rank property and Proposition 4.

In order to achieve the eigen-system of the large $n \times n$ matrix $W = Z\Lambda^{-1}Z^\top$, we solve the eigen-system of a small $m \times m$ matrix $M = \Lambda^{-1/2}Z^\top Z\Lambda^{-1/2}$ guided by Proposition 4, resulting in r ($< m$) eigenvector-eigenvalue pairs $\{(\boldsymbol{v}_k, \sigma_k)\}_{k=1}^r$ where $1 > \sigma_1 \geq \cdots \geq \sigma_r > 0$. Note that we discard 1 eigenvalues of M (W) which correspond to 0 eigenvalues of L. After

expressing $V = [\boldsymbol{v}_1, \cdots, \boldsymbol{v}_r] \in \mathbb{R}^{m \times r}$ (V is column-orthonormal) and $\Sigma = \mathrm{diag}(\sigma_1, \cdots, \sigma_r) \in \mathbb{R}^{r \times r}$, we obtain an eigenvector matrix P of W as

$$P = Z\Lambda^{-1/2}V\Sigma^{-1/2}, \tag{3.5}$$

each column of which corresponds to an eigenvalue $0 < \sigma_k < 1$. Let denote by $\bar{P}$ another eigenvector matrix corresponding to eigenvalues $\{\sigma_k = 0\}_{k=r+1}^{n'}$ ($n' < n$) of W.

Realizing $\lambda_k = 1 - \sigma_k$ for the Anchor Graph Laplacian eigenvalues, we drive the pseudo inverse of the Anchor Graph Laplacian L as follows

$$\begin{aligned}
L^+ &= \sum_{\lambda_k > 0} \frac{1}{\lambda_k} \boldsymbol{p}_k \boldsymbol{p}_k^\top \\
&= \sum_{\sigma_k < 1} \frac{1}{1 - \sigma_k} \boldsymbol{p}_k \boldsymbol{p}_k^\top \\
&= \sum_{k=1}^{r} \frac{1}{1 - \sigma_k} \boldsymbol{p}_k \boldsymbol{p}_k^\top + \sum_{k=r+1}^{n'} \boldsymbol{p}_k \boldsymbol{p}_k^\top \\
&= P(I - \Sigma)^{-1} P^\top + \bar{P}\bar{P}^\top \\
&= Z\Lambda^{-1/2} V \Sigma^{-1/2} (I - \Sigma)^{-1} \Sigma^{-1/2} V^\top \Lambda^{-1/2} Z^\top + \bar{P}\bar{P}^\top \\
&= Z\Lambda^{-1/2} V (\Sigma - \Sigma^2)^{-1} V^\top \Lambda^{-1/2} Z^\top + \bar{P}\bar{P}^\top \\
&= ZQQ^\top Z^\top + \bar{P}\bar{P}^\top,
\end{aligned} \tag{3.6}$$

where $Q = \Lambda^{-1/2} V (\Sigma - \Sigma^2)^{-1/2} \in \mathbb{R}^{m \times r}$.

The derived equation eq. (3.6) inspires us to truncate the pseudo inverse L^+ on its eigenspectrum to acquire a low-rank kernel, which is carried out by keeping the eigencomponents $ZQQ^\top Z^\top$ of L^+ corresponding to r largest eigenvalues of L^+ (i.e., r smallest nonzero eigenvalues $\{\lambda_k = 1 - \sigma_k\}_{k=1}^{r}$ of L).

Accordingly, we give the following Proposition 5.

Proposition 5. *The best rank-r approximation to the pseudo inverse of the Anchor Graph Laplacian is a low-rank kernel* $\mathcal{K}^L(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{z}^\top(\boldsymbol{x}) Q Q^\top \boldsymbol{z}(\boldsymbol{x}')$.

We name $\mathcal{K}^L$ the *Anchor Graph Laplacian kernel* since it stems from the Anchor Graph Laplacian L, and its rank is $r < m$. It is noticeable that the mechanism that we derive

the kernel $\mathcal{K}^L$ is unsupervised without accessing the label information. [214] tried to learn a kernel matrix from the family $P\Omega P^\top$ with Ω being an active diagonal matrix. [214] incorporated the label information associated with the l labeled examples into an optimization framework such that the optimal Ω can be solved. In comparison, we actually fix $\Omega = (I - \Sigma)^{-1}$ to avoid tedious optimization at a large scale. Such a treatment is simple yet effective, and will be verified through our experiments shown in Section 3.6.

3.5.3 Anchor Graph Warped Kernels

It turns out that if the manifold assumption holds, i.e., manifolds exist or nearly exist under input data collections, graph-based approaches work well for various learning problems including clustering, retrieval, and semi-supervised learning [23][211][213]. Accordingly, the low-rank kernels $\mathcal{K}^W$ and $\mathcal{K}^L$, which are directly constructed from Anchor Graphs, are good enough at handling large-scale SSL. Nevertheless, when manifolds do not exist or are not evident due to diverse data distributions or poor feature extraction schemes, merely relying on graphs is not sufficient to deliver effective data-dependent kernels for tackling complex data configurations. Hence, we consider yielding a low-rank kernel from a priori kernel that has manifested beneficial to the data domain. This time the role of the Anchor Graph is to warp a priori kernel like a filter, making the resulting kernel encompass the neighborhood structure unveiled by the Anchor Graph.

Given a priori kernel function $\kappa : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$, we follow the state-of-the-art GSSL method *manifold regularization* [12] to derive our low-rank kernel. Note that even the accelerated version [124] of manifold regularization costs a quadratic training time complexity $O(n^2)$ due to learning a full-size classification function

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \kappa(\boldsymbol{x}, \boldsymbol{x}_i) a_i, \tag{3.7}$$

which originates from the *Representer Theorem* of kernel machines [149]. To compose a scalable kernel machine, we propose an economical classification function based on the small set of anchors:

$$f(\boldsymbol{x}) = \sum_{k=1}^{m} \kappa(\boldsymbol{x}, \boldsymbol{u}_k) a_k = \boldsymbol{a}^\top \boldsymbol{k}(\boldsymbol{x}), \tag{3.8}$$

where $\boldsymbol{a} = [a_1, \cdots, a_m]^\top$ denotes the coefficient vector of f to be learned, and $\boldsymbol{k}(\boldsymbol{x}) = \begin{bmatrix} \kappa(\boldsymbol{x}, \boldsymbol{u}_1) \\ \cdots\cdots \\ \kappa(\boldsymbol{x}, \boldsymbol{u}_m) \end{bmatrix} \in \mathbb{R}^m$ denotes the anchor-supported kernel feature vector.

Subsequently, we formulate a GSSL framework to seek the optimal classification function f of the expression being shown in eq. (3.8), that is

$$\min_{f \in span\{\kappa(\cdot, \boldsymbol{u}_k)\}_{k=1}^m} \sum_{i=1}^{l} Loss(f(\boldsymbol{x}_i), y_i) + \beta \|f\|_{\mathcal{H}_\kappa}^2 + \gamma \|f\|_{\mathcal{G}}^2, \tag{3.9}$$

where $Loss(,)$ is a proper loss function defined on labeled examples, $\beta > 0$ is the regularization parameter for the *reproducing kernel Hilbert space* (RKHS) norm $\|f\|_{\mathcal{H}_\kappa}^2$ ($\mathcal{H}_\kappa$ is the RKHS induced by the given kernel κ), and $\gamma > 0$ is the regularization parameter for the Anchor Graph Laplacian regularization norm $\|f\|_{\mathcal{G}}^2 = \boldsymbol{f}^\top L \boldsymbol{f}$ defined in Chapter 2.

As a matter of fact, the above GSSL framework approximates the manifold regularization framework by means of the introduced anchors $\mathcal{U} = \{\boldsymbol{u_k}\}_{k=1}^m$. Importantly, we can show that eq. (3.9) is equivalent to a supervised learning framework, formulated as follows, using a novel kernel $\tilde{\kappa}$:

$$\min_{f \in span\{\tilde{\kappa}(\cdot, \boldsymbol{u}_k)\}_{k=1}^m} \sum_{i=1}^{l} Loss(f(\boldsymbol{x}_i), y_i) + \beta \|f\|_{\mathcal{H}_{\tilde{\kappa}}}^2. \tag{3.10}$$

We present and prove our crucial finding that $\tilde{\kappa}$ is a low-rank kernel in the following theorem.

Theorem 6. *The GSSL framework formulated in eq. (3.9) results in a low-rank kernel function* $\tilde{\kappa}(\boldsymbol{x}, \boldsymbol{x'}) = \boldsymbol{k}^\top(\boldsymbol{x}) R^{-1} \boldsymbol{k}(\boldsymbol{x'})$ *in which* $R \in \mathbb{R}^{m \times m}$ *is a positive definite matrix.*

Proof. We deduce the RKHS norm as

$$\|f\|_{\mathcal{H}_\kappa}^2 = \langle f, f \rangle_{\mathcal{H}_\kappa} = \sum_{k,k'=1}^{m} a_k \left\langle \kappa(\cdot, \boldsymbol{u}_k), \kappa(\cdot, \boldsymbol{u}_{k'}) \right\rangle_{\mathcal{H}_\kappa} a_{k'} = \sum_{k,k'=1}^{m} a_k \kappa(\boldsymbol{u}_k, \boldsymbol{u}_{k'}) a_{k'} = \boldsymbol{a}^\top K_{\mathcal{U}\mathcal{U}} \boldsymbol{a}, \tag{3.11}$$

where $K_{\mathcal{U}\mathcal{U}} = ((K_{\mathcal{U}\mathcal{U}})_{k,k'})_{1 \le k,k' \le m} = (\kappa(\boldsymbol{u}_k, \boldsymbol{u}_{k'}))_{1 \le k,k' \le m}$ is a kernel matrix computed in between the anchor set $\mathcal{U}$.

The Anchor Graph Laplacian regularization norm is deduced as

$$\begin{aligned}\|f\|_{\mathcal{G}}^2 &= \boldsymbol{f}^\top L \boldsymbol{f} = (K_{\mathcal{U}}\boldsymbol{a})^\top L (K_{\mathcal{U}}\boldsymbol{a}) \\ &= (K_{\mathcal{U}}\boldsymbol{a})^\top (I - Z\Lambda^{-1}Z^\top)(K_{\mathcal{U}}\boldsymbol{a}) \\ &= \boldsymbol{a}^\top \left(K_{\mathcal{U}}^\top K_{\mathcal{U}} - K_{\mathcal{U}}^\top Z\Lambda^{-1}Z^\top K_{\mathcal{U}}\right)\boldsymbol{a}, \end{aligned} \tag{3.12}$$

where $\boldsymbol{f} = \begin{bmatrix} f(\boldsymbol{x}_1) \\ \cdots \\ f(\boldsymbol{x}_n) \end{bmatrix} \in \mathbb{R}^n$ and $K_{\mathcal{U}} = \begin{bmatrix} \kappa(\boldsymbol{x}_1, \boldsymbol{u}_1), \cdots, \kappa(\boldsymbol{x}_1, \boldsymbol{u}_m) \\ \cdots\cdots \\ \kappa(\boldsymbol{x}_n, \boldsymbol{u}_1), \cdots, \kappa(\boldsymbol{x}_n, \boldsymbol{u}_m) \end{bmatrix} \in \mathbb{R}^{n\times m}$.

Putting eq. (3.11)(3.12) together, we have

$$\begin{aligned} &\beta\|f\|_{\mathcal{H}_\kappa}^2 + \gamma\|f\|_{\mathcal{G}}^2 \\ =&\beta\boldsymbol{a}^\top K_{\mathcal{U}\mathcal{U}}\boldsymbol{a} + \gamma\boldsymbol{a}^\top \left(K_{\mathcal{U}}^\top K_{\mathcal{U}} - K_{\mathcal{U}}^\top Z\Lambda^{-1}Z^\top K_{\mathcal{U}}\right)\boldsymbol{a} \\ =&\beta\boldsymbol{a}^\top \left(K_{\mathcal{U}\mathcal{U}} + \frac{\gamma}{\beta}K_{\mathcal{U}}^\top K_{\mathcal{U}} - \frac{\gamma}{\beta}K_{\mathcal{U}}^\top Z\Lambda^{-1}Z^\top K_{\mathcal{U}}\right)\boldsymbol{a} \\ =&\beta\boldsymbol{a}^\top R\boldsymbol{a}, \end{aligned} \tag{3.13}$$

where the matrix

$$R = K_{\mathcal{U}\mathcal{U}} + \frac{\gamma}{\beta}K_{\mathcal{U}}^\top K_{\mathcal{U}} - \frac{\gamma}{\beta}K_{\mathcal{U}}^\top Z\Lambda^{-1}Z^\top K_{\mathcal{U}} \in \mathbb{R}^{m\times m} \tag{3.14}$$

can be calculated in $O(m^2n + m^3)$ time. Obviously, R is positive definite due to the fact $K_{\mathcal{U}}^\top K_{\mathcal{U}} - K_{\mathcal{U}}^\top Z\Lambda^{-1}Z^\top K_{\mathcal{U}} = K_{\mathcal{U}}^\top L K_{\mathcal{U}} \succeq 0$.

Let us define a novel feature map $\mathbf{F}(\boldsymbol{x}) = R^{-1/2}\boldsymbol{k}(\boldsymbol{x})$ and $\boldsymbol{w} = R^{1/2}\boldsymbol{a}$. Then we obtain

$$f(\boldsymbol{x}) = \boldsymbol{a}^\top \boldsymbol{k}(\boldsymbol{x}) = \boldsymbol{w}^\top \mathbf{F}(\boldsymbol{x}), \tag{3.15}$$

$$\beta\|f\|_{\mathcal{H}_\kappa}^2 + \gamma\|f\|_{\mathcal{G}}^2 = \beta\|\boldsymbol{w}\|^2. \tag{3.16}$$

By plugging eqs. (3.15)(3.16) back into eq. (3.9), we achieve a supervised linear machine as follows

$$\min_{\boldsymbol{w}\in\mathbb{R}^m} \sum_{i=1}^{l} Loss(\boldsymbol{w}^\top \mathbf{F}(\boldsymbol{x}_i), y_i) + \beta\|\boldsymbol{w}\|^2. \tag{3.17}$$

Therefore, the original GSSL framework is equivalently reduced to a supervised linear classification problem based on the new feature representation $\{\mathbf{F}(\boldsymbol{x}_i)\}_{i=1}^l$. Immediately, we

Table 3.4: Summary of three low-rank kernels generated from and with Anchor Graphs.

Low-Rank Kernel	**Form**	**Rank**	**Feature Map $\mathbf{F}(\boldsymbol{x})$**	**Property of $\mathbf{F}(\boldsymbol{x})$**
Anchor Graph kernel	$\boldsymbol{z}^\top(\boldsymbol{x})\Lambda^{-1}\boldsymbol{z}(\boldsymbol{x}')$	m	$\Lambda^{-1/2}\boldsymbol{z}(\boldsymbol{x})$	sparse
Anchor Graph Laplacian kernel	$\boldsymbol{z}^\top(\boldsymbol{x})QQ^\top\boldsymbol{z}(\boldsymbol{x}')$	$< m$	$Q^\top\boldsymbol{z}(\boldsymbol{x})$	dense
Anchor Graph warped kernel	$\boldsymbol{k}^\top(\boldsymbol{x})R^{-1}\boldsymbol{k}(\boldsymbol{x}')$	m	$R^{-1/2}\boldsymbol{k}(\boldsymbol{x})$	dense

acquire a "linear" kernel in terms of $\mathbf{F}(\boldsymbol{x})$:

$$\begin{aligned}\tilde{\kappa}(\boldsymbol{x},\boldsymbol{x}') &= \mathbf{F}^\top(\boldsymbol{x})\mathbf{F}(\boldsymbol{x}') \\ &= \boldsymbol{k}^\top(\boldsymbol{x})R^{-1}\boldsymbol{k}(\boldsymbol{x}'),\end{aligned} \tag{3.18}$$

which completes the proof. □

To gain an in-depth understanding, the low-rank kernel $\tilde{\kappa}$ is obtained by warping a priori kernel κ using the Anchor Graph based regularization, and the matrix R defined in eq. (3.14) which is composing the warped kernel $\tilde{\kappa}$ has absorbed the information from the Anchor Graph Laplacian L. Thus, we can say that $\tilde{\kappa}$ assimilates the neighborhood structure unveiled by the Anchor Graph, bearing a geometrical meaning compared to the raw kernel κ. To shed light on kernel generation, [157] has initially proven that the original manifold regularization framework using a kNN graph can yield a full-rank new kernel which is warped by the graph Laplacian of the kNN graph. In contrast, we generate a low-rank kernel to fulfill the goal of scalable graph-based semi-supervised learning.

In the sequel, we summarize the three low-rank kernels we have proposed in Table 3.4.

3.6 Experiments

In this section, we conduct experiments on three large image databases which vary in size from 60,000 to 1,000,000 samples. We evaluate the semi-supervised classification performance of the three proposed low-rank kernels which are used in conjunction with linear

SVMs after simple linearization. We compare these **linearized low-rank kernels+linear SVMs** against AGR developed in Chapter 2 and two recent scalable GSSL algorithms *Eigenfunctions* [49] and *Prototype Vector Machines* (PVMs) [204]. We also report the performance of several baseline methods including 1NN, linear SVMs and kernel SVMs. We feed the same Gaussian kernel $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\|\boldsymbol{x} - \boldsymbol{x}'\|^2/2\epsilon^2)$ to kernel SVMs, PVMs, and the initial kernel for our Anchor Graph warped kernel on every dataset. The parameter ϵ is tuned via cross-validation on every dataset. To construct an Anchor Graph, we also use a Gaussian kernel $\mathcal{K}_h(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\|\boldsymbol{x} - \boldsymbol{x}'\|^2/2h^2)$ to define the regression matrix Z. The value of the width parameter h is empirically chosen according to the trick suggested by [112]. Since PVMs also use anchors, we assign the same set of K-means clustering centers to PVMs and our Anchor Graph-related methods on every dataset. The iteration number of K-means clustering is set to five for all datasets. We use LIBLINEAR [46] for running all linear SVMs. All our experiments are run on a workstation with a 2.53 GHz Intel Xeon CPU and 48GB RAM.

3.6.1 CIFAR-10

CIFAR-10 is a labeled subset of the 80 million tiny image collection [174], which consists of a total of 60,000 32×32 color images from ten object classes. This dataset [91] is partitioned into two parts: a training set of 50,000 images and a test set of 10,000 images, all of which cover ten classes. Each image in this dataset is represented by a 512-dimensional GIST feature vector [138]. Some image examples selected from **CIFAR-10** are displayed in Figure 3.1. The parameter s in the Anchor Graphs built on this dataset is set to 10.

To set up SSL trials, we sample $l = 100$ and $l = 1,000$ labeled samples respectively uniformly at random from the training set such that they contain at least one sample coming from each class. The one-versus-all strategy is carried out to run all SVM-related methods. The classification accuracy, averaged over 20 SSL trials, is computed over the test set for each method under comparison. All of the evaluation results are shown in Table 3.5, which indicate that our Anchor Graph warped kernel with 1,000 anchors gives rise to the highest classification accuracy when cooperated with linear SVMs through kernel linearization.

Table 3.5: Classification accuracy (%) on **CIFAR-10** (60,000 samples).

Method	$l = 100$	$l = 1,000$
1NN	29.86	36.71
Linear SVM	32.60	38.20
Kernel SVM	35.20	42.10
Eigenfunctions	31.35	41.10
PVM(square loss) with 500 anchors	35.71	42.88
PVM(square loss) with 1,000 anchors	37.21	44.31
PVM(hinge loss) with 500 anchors	36.19	43.43
PVM(hinge loss) with 1,000 anchors	38.92	45.88
AGR with 500 anchors	36.25	44.45
AGR with 1,000 anchors	38.25	46.78
Anchor Graph kernel with 500 anchors+SVM	34.92	42.80
Anchor Graph kernel with 1,000 anchors+SVM	37.21	45.88
Anchor Graph Laplacian kernel with 500 anchors+SVM	36.62	44.70
Anchor Graph Laplacian kernel with 1,000 anchors+SVM	39.21	47.90
Anchor Graph warped kernel with 500 anchors+SVM	37.68	46.60
Anchor Graph warped kernel with 1,000 anchors+SVM	**40.35**	**50.33**

Table 3.6: Classification error rates (%) on **MNIST** (70,000 samples).

Method	$l = 100$	$l = 1,000$
1NN	28.86	11.96
Linear SVM	27.60	14.22
Kernel SVM	23.70	8.58
Eigenfunctions	22.35	12.91
PVM(square loss) with 500 anchors	21.12	9.75
PVM(square loss) with 1,000 anchors	20.21	8.88
PVM(hinge loss) with 500 anchors	20.33	9.18
PVM(hinge loss) with 1,000 anchors	19.55	8.21
AGR with 500 anchors	13.21	8.10
AGR with 1,000 anchors	12.11	7.35
Anchor Graph kernel with 500 anchors+SVM	15.51	10.80
Anchor Graph kernel with 1,000 anchors+SVM	14.35	9.69
Anchor Graph Laplacian kernel with 500 anchors+SVM	12.11	7.28
Anchor Graph Laplacian kernel with 1,000 anchors+SVM	11.20	6.80
Anchor Graph warped kernel with 500 anchors+SVM	10.11	6.86
Anchor Graph warped kernel with 1,000 anchors+SVM	**9.30**	**5.66**

Table 3.7: Classification error rates (%) on **Extended MNIST** (1,030,000 samples).

Method	$l = 100$	$l = 1,000$
1NN	40.65	35.36
Linear SVM	43.60	37.13
Kernel SVM	38.70	32.58
Eigenfunctions	37.94	33.91
PVM(square loss) with 500 anchors	30.21	26.36
PVM(square loss) with 1,000 anchors	27.21	23.56
PVM(hinge loss) with 500 anchors	29.31	25.37
PVM(hinge loss) with 1,000 anchors	26.32	22.20
AGR with 500 anchors	25.71	21.90
AGR with 1,000 anchors	22.46	18.26
Anchor Graph kernel with 500 anchors+SVM	28.21	24.90
Anchor Graph kernel with 1,000 anchors+SVM	26.19	22.78
Anchor Graph Laplacian kernel with 500 anchors+SVM	23.28	19.92
Anchor Graph Laplacian kernel with 1,000 anchors+SVM	20.79	16.58
Anchor Graph warped kernel with 500 anchors+SVM	21.35	17.82
Anchor Graph warped kernel with 1,000 anchors+SVM	**18.19**	**14.28**

Figure 3.1: Example images from the **CIFAR-10** dataset.

3.6.2 MNIST

The **MNIST** dataset [102] contains handwritten digit images from '0' to '9'. It has a training set of 60,000 samples and a test set of 10,000 samples. Some image examples from **MNIST** are plotted in Figure 3.2. The parameter s in the Anchor Graphs built on this dataset is set to 3.

Similar to the **CIFAR-10** experiments, we sample $l = 100$ and $l = 1,000$ labeled samples respectively uniformly at random from the training set such that they contain at least one sample coming from each class, thus setting up the SSL trials. Like the **CIFAR-10** experiments, the SSL settings on **MNIST** also introduce skewed class distributions in the labeled samples.

Averaged over 20 SSL trials, we calculate the error rates over the test set for all referred methods, with the number of labeled samples being 100 and 1,000 respectively. The results are listed in Table 3.6. Again, we observe that the Anchor Graph warped kernel ($m =$

Figure 3.2: Example images from the **MNIST** dataset. The original figure is from `http://www.cad.zju.edu.cn/home/dengcai/Data/MNIST/images.html`.

1,000) working with linear SVMs is superior to the other methods in comparison, which demonstrates that the generated low-rank kernel $\tilde{\kappa}$ via Anchor Graph warping makes sense and leads to more discriminative classifiers, thereby enabling more accurate graph-based SSL at a large scale.

3.6.3 Extended MNIST

In order to test the performance at a larger scale, we construct an **Extended MNIST** by translating the original training images by one and two pixels in eight directions, and then obtain 17×60,000 training images like [87]. The parameter s in the Anchor Graphs built on this dataset is set to 3.

By repeating the similar evaluation process as **MNIST**, we report the average classification error rates over 20 SSL trials and on the test set for all referred methods in Table 3.7, given 100 and 1,000 labeled samples respectively. The results further confirm the superior classification performance of the proposed GSSL framework: linearized Anchor Graph warped kernels + linear SVMs.

3.7 Summary and Discussion

Through this chapter, we have addressed the scalability issue that GSSL suffers from and proposed several key techniques to establish scalable kernel machines under the semi-supervised scenario. The proposed techniques take advantage of the Anchor Graph from a kernel point of view, generating a group of low-rank kernels directly from or utilizing the Anchor Graph. Such low-rank kernels enable simple linearization, by which new fea-

ture spaces are produced and original semi-supervised kernel machines are converted to supervised linear machines so that linear SVMs can be applied.

Why we prefer developing nonlinear kernel machines is because kernel-based classifiers have been theoretically and empirically proven to be able to tackle practical data that are mostly linearly inseparable. Our proposed low-rank kernels all bear closed-form solutions, costing linear time $O(n)$ to generate them. Since training nonlinear kernel machines in semi-supervised settings can be transformed to training linear machines, typically linear SVMs, in supervised settings, the cost for classifier training is substantially reduced, only costing $O(l)$ time where $l \ll n$. Our experiments corroborate that linear SVMs using the proposed linearized low-rank kernels, especially the Anchor Graph warped kernels, exhibit superior classification accuracy over the state-of-the-art scalable GSSL methods, and are more discriminative than AGR which only imposes label propagation and does not emphasize margin maximization.

For the parameter settings of Anchor Graphs, we simply keep $m \leq 1000$ to bound the time complexity, although we are aware that more anchors (larger m) are very likely to boost the classification accuracy for all Anchor Graph-related approaches. The experimental results presented in Chapter 2 have shown that when $m \geq n/10$ AGR can achieve the classification accuracy comparable to that of classical GSSL methods using kNN graphs. For the parameter s, we set it to a small integer that is smaller than m/c. A typical value for s is in $[2, 10]$. For the regression matrix Z, we have found that in most datasets we have tried, LAE-optimized Z is better than predefined Z but the latter is good enough when using sufficient anchors (e.g., $m \geq 500$). In addition, to save the construction time, we usually use predefined Z to construct large scale Anchor Graphs, referring to the experiments conducted in this chapter.

There is a problem being worthy to be considered that when the underlying manifolds may not exist or are not evident, how well our Anchor Graph-related approaches can work. Since all of the developed techniques including Anchor Graphs, AGR, and low-rank kernel generation methods are based on the manifold assumption, they could behave poor if such an assumption does not hold on some datasets. Then, other assumptions such as the cluster assumption introduced in Section 3.2 should be incorporated to cope with the "difficult"

circumstances. Motivated by [29] that combined multiple semi-supervised assumptions, we are thinking of integrating our proposed Anchor Graph-related methods with proper cluster-based ideas or any other sensible assumptions about the data to develop stronger SSL models which are expected to accommodate a broader range of data structures.

Lastly, we discuss the limits of semi-supervise learning. [158] pointed out the particular cases where unlabeled data do not help improve the performance of supervised learning when adding them for co-training. [132] thought that when unlabeled data increase to infinity the graph Laplacian eigenvectors tend to become less informative. All these are valuable explorations to shed light on the theoretical value of unlabeled data. We would also like to investigate and disclose the value of unlabeled data under our Anchor Graph's framework.

Part II

Nearest Neighbor Search with Hashing

Chapter 4

Unsupervised Hashing

Hashing is becoming increasingly popular for time-efficient nearest neighbor search in massive databases. However, learning compact hash codes that yield good search performance is still a challenge. Moreover, in many cases real-world data often live on low-dimensional manifolds, which should be taken into account in order to capture meaningful nearest neighbors in the hash code learning process.

In this chapter, we present a novel graph-based hashing approach that we name *Anchor Graph Hashing* (AGH) and has been described in our recent paper [117]. AGH is fully unsupervised but can automatically discover the neighborhood structure inherent in the data to learn appropriate compact codes. To make such an approach computationally feasible for large-scale databases, we utilize *Anchor Graphs* that have been presented in Chapter 2 to obtain tractable low-rank adjacency matrices and derive the nonlinear hash functions from the eigenspectra of such low-rank matrices. The formulation of our Anchor Graph-driven hash functions allows constant time hashing of a new data point by extrapolating graph Laplacian eigenvectors to eigenfunctions. Finally, we describe a hierarchical threshold learning procedure in which each eigenfunction yields multiple bits, leading to higher search accuracy. Experimental comparison with the other state-of-the-art hashing methods on two large datasets demonstrates the efficacy of the presented AGH approach.

In the remainder of this chapter, we state the problem background of fast nearest neighbor search in Section 4.1, review the related work in Section 4.2, introduce the notations that we will use in Section 4.3, present our approach AGH in Section 4.4, show the ex-

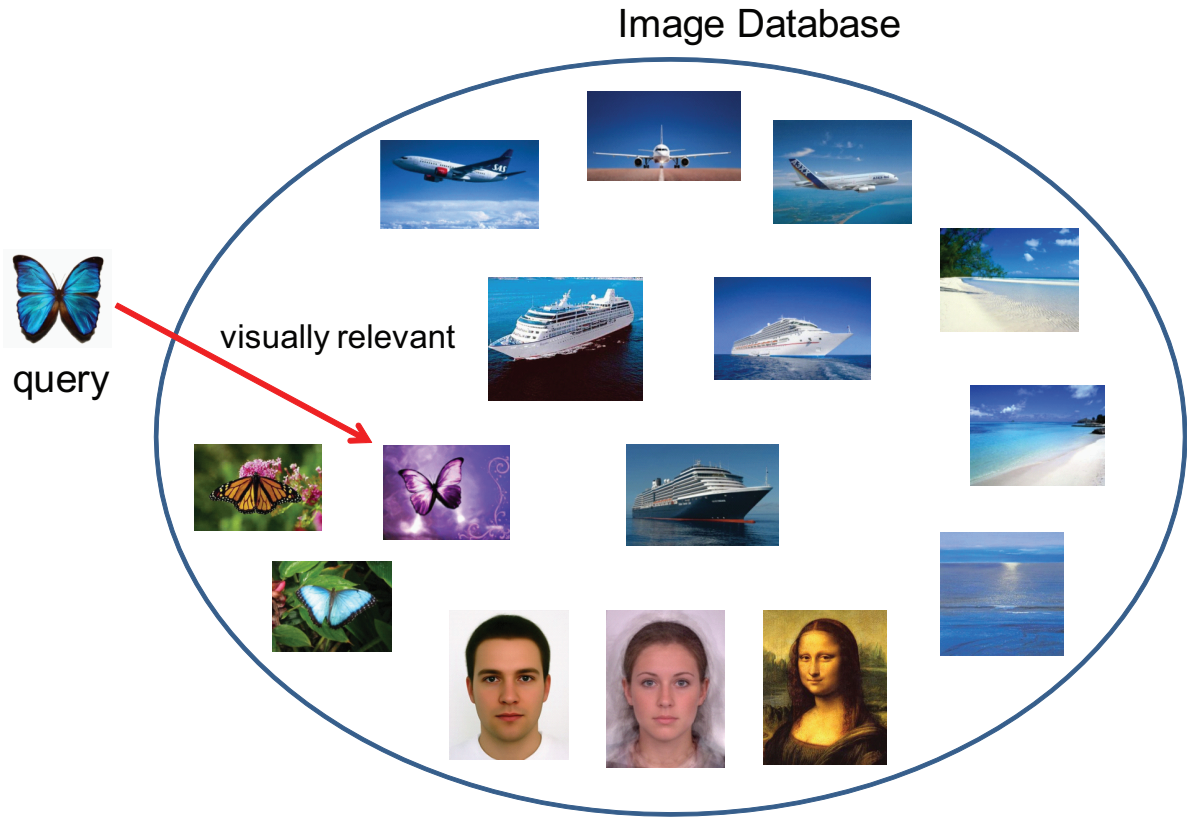


Figure 4.1: A nearest neighbor search example in image retrieval. The query image is a butterfly image, and the most visually relevant images, i.e., images containing at least a butterfly, are expected to be retrieved from the given image database.

perimental results in Section 4.5, and finally give our summary and discussion in Section 4.6.

4.1 Problem Background

Nearest neighbor (NN) search is a fundamental problem that arises commonly in a variety of fields including computer vision, machine learning, database systems, data mining, multimedia, and information retrieval. Figure 4.1 displays a standard NN search task in the context of image retrieval which has attracted broad interest from researchers in computer vision, multimedia, and information retrieval.

From the conceptually algorithmic perspective, searching nearest neighbors of a query $\boldsymbol{q} \in \mathbb{R}^d$ requires scanning all n items in a database $\mathcal{X} = \{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^n$ (suppose that data dimension is d and data size is n), which has a linear time complexity $O(n)$ (more precisely, $O(dn)$). For large n, e.g., millions, exhaustive linear search is expensive and therefore impractical for large-scale applications. Consequently, a great number of techniques have been proposed in the past few decades, focusing on fast *approximate nearest neighbor* (ANN) (or near neighbor) search [151].

One classical paradigm to address the fast ANN search problem is building tree-based indexing structures, such as *kd*-trees [14][50] which provide a logarithmic query (i.e., search) time complexity $O(\log n)$ in expectation. However, for very high-dimensional data sets, the worst-case search time for a *kd*-tree is $O(dn^{1-\frac{1}{d}})$ that exponentially approaches the exhaustive search time $O(dn)$, as discovered in [103]. As a matter of fact, almost all tree-based methods theoretically suffer from the dimensionality issue with their performance typically degrading to exhaustive linear scan for high-dimensional data. Especially for a *kd*-tree, it only works fine, in both theory and practice, with dimensions up to 20 [50].

To overcome the dimensionality issue, hashing-based methods coping with ANN search have attracted considerable attention since the advent of the well-known *Locality-Sensitive Hashing* (LSH) [52]. These methods convert each database item into a binary code and save all codes in single or multiple hash lookup tables. Through looking up the code of a given query in these hash tables, hashing algorithms can provide sub-linear $O(n^\rho)$ $(0 < \rho < 1)$ or even constant $O(1)$ search time.

Figure 4.2 shows a schematic diagram for a tree and a hash table, respectively. Shown in Figure 4.2(a), in the search stage of a tree-based algorithm, some proper search strategies such as *Backtracking* and *Branch-and-Bound* are exploited to browse the tree and then locate the leaf node in which the database point is found closest to the query $\boldsymbol{q}$ among the visited tree nodes. Revealed in Figure 4.2(b), in the search stage of a hashing algorithm, the code of the query $\boldsymbol{q}$ is first extracted and then used to look up the hash table to locate the matched hash bucket that contains the near neighbors.

From Figure 4.2, we can see that hash tables have much simpler indexing structures than trees. Tree-based methods require a hierarchical scheme for both indexing and search

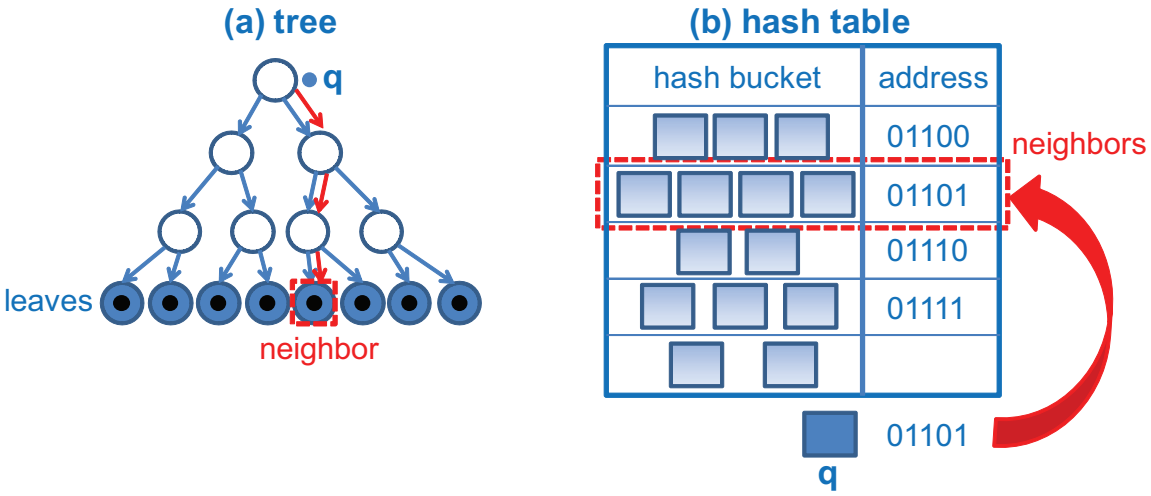


Figure 4.2: Schematic diagrams for a tree and a hash table. $\boldsymbol{q}$ represents a query point. (a) A binary space partitioning tree (e.g., *kd*-tree) where each node contains a database point. (b) A linear hash table in which the binary codes of database points are treated as addresses (or indices) into the hash table and every address points to a hash bucket that stores corresponding database points.

stages, while hashing methods work in parallel and their generated codes can be directly used to point to the addresses of data during table lookups. Notably, hashing requires less storage than tree-based methods, because trees need to store the original data points in memory in order to perform tree browsing during indexing and search stages, whereas hash tables only need to save the data IDs if the exact NN ranking is not necessary.

In practice, rigorous large-scale performance comparison between *kd*-tree and hashing for fast NN search still remains missing in the literature. Through some performance evaluations carried out on some benchmark databases such as local descriptor SIFTs [119] of images, [3] showed that when using NN search for nonparametric classification, LSH is generally better than *kd*-tree at the similar storage expense in terms of recognition rate but inferior in terms of search time (the total running time for returning neighbors given a query), and that the search time of LSH rises sharply with the database size while that of *kd*-tree rises slowly (logarithmic increase). However, [3] did not supply the comparison between *kd*-tree and some recent advanced hashing algorithms, particularly compact hashing that

provides nearly constant search time. It did not include experiments for high-dimensional data (e.g., ≥ 2000 dimensions) either.

In this chapter, we focus on hashing with compact codes of less than 100 binary bits, which is called *Compact Hashing* [175] in the literature. It turns out that ANN search under such a scenario is both time and memory efficient. What's more, we suppose that only a single hash table is used in the offline indexing phase and the online search phase.

Let us define the basic setting for hashing with compact codes, which we call *compact hashing mode* and is the infrastructure of all compared hashing algorithms we will evaluate in this chapter. Under such a mode, only one hash table is employed to index database items (then the indexing structure is a simple linear table), and $8 \sim 64$ binary hash bits (within the storage size for a double-precision value) are adopted to represent each database item or a query in the Hamming space. The hash table construction time complexity is $O(n)$ in general. Here we concentrate on the query (search) time complexity since it is the key to the true speed of hashing-based ANN search. This query time consists of two components: 1) *hashing time* that is the time of converting a query to its binary hash code via hash functions (usually multiple hash functions), and 2) *hash table lookup time* that is the time of looking up, in a hash table, the target code addresses which are within a small Hamming radius to the query's code. Throughout this chapter, we use Hamming radius 2 to retrieve potential near neighbors for each query. For high-dimensional data, hashing time may dominate the total query time. Very importantly, hash table lookup time (hash lookup time in abbreviation) can be substantially fast by using a simple implementation. That is, flipping zero up to two bits into the query's hash code and then searching the new code by addressing the hash table, which results in constant time querying. Table 4.1 provides the times of Hamming-radius-2 hash lookups for varied hash bit length, from which we find that the hash lookup times are $\sum_{k=0}^{2} \binom{r}{k} \leq 2081$ for hash code length $r \leq 64$. Hence, we are able to regard the hash lookup time as a constant $O(1)$.

As a critical finding from Table 4.1, the hash lookup time can be ignored with respect to the hashing time for high-dimensional data of more than 500 dimensions. For example, a LSH algorithm using 64 hash bits needs $2dr = 64,000$ floating-point operations for hashing a 500-dimensional query, but only takes 2081 addressing operations for hash lookups within

Table 4.1: Compact hashing mode: hash table lookup times within Hamming radius 2.

Hash Code Length	**Hash Lookup Times**
8	37
12	79
16	137
24	301
32	529
48	1177
64	2081

Hamming radius 2. The latter is at most 3.25% of the former (note that the speed of addressing operations is much faster than floating-point operations), so it can be completely skipped compared with the hashing time (i.e., hash function evaluation time) which actually constitutes the speed bottleneck of NN search via compact hashing.

4.2 Related Work

Beyond *kd*-trees, various variants of trees have been invented to deal with fast NN search, such as ball trees [139], PCA trees [179], metric trees [201][111], random projection trees [38], vantage-point trees [96], hierarchical K-means trees [136][130], etc. Very few of them have theoretic guarantees about the accuracy of returned neighbors, and most of them demonstrate their empirical advantages over *kd*-trees. Even when we only consider classical *kd*-trees, there have been considerable improvements in both theory and practical performance. As disclosed by the previous section, the curse of dimensionality causes that a *kd*-tree search algorithm has to visit many more branches in very high-dimensional spaces than in lower dimensional spaces. The worst case is that when the number of data points n is slightly larger than the number of dimensions d, the search algorithm evaluates most of the data points in the tree and is thus no faster than the exhaustive scan of all of the points. With in-depth analysis, [74] indicated a general rule: a *kd*-tree search algorithm is efficient when $n \gg 2^d$. To enable *kd*-trees to work with high-dimensional data, a lot of modifications

to *kd*-trees have been devised in order to remedy the curse of dimensionality by applying PCA or random projections first [162][154][38], speed up the query time by limiting the backtracking times [6], or enhance the practical performance by leveraging multiple trees [154][130] or distributed tree organization [2]. All these alternatives modulate *kd*-trees for specific applications such as indexing large-scale document and image collections [130][2], fast image descriptor matching [154], and so on. [130] provides a comprehensive and systematic work which also gives the guidance to choose the best tree search method for particular data. In addition, [130] developed a software system that can take any given dataset and a desired degree of precision to automatically determine the best tree algorithm and the corresponding parameter values. [130] concluded that the hierarchical K-means tree behaves best on many datasets, while the multiple-randomized *kd*-tree shows the best performance on other datasets.

While there exist many new tree-based methods better than *kd*-trees, they are not trying to address the dimensionality issue explicitly, and lack elegant formulations to the original nearest neighbor search problem. In this chapter, we study the hashing technology which tends to remove the curse of dimensionality [39][4] and formulates the NN search problem in a binary code space.

Hashing methods essentially establish Hamming embeddings of data points, which map real-valued data to binary codes. The pioneering work on *Locality-Sensitive Hashing* (LSH) [52][4] uses simple random projections for such mappings. It has been extended to a variety of similarity measures including the cosine similarity [27], p-norm distances for $p \in (0, 2]$ [39], the Mahalanobis distance [95], and the kernel similarity [94]. There are also a few variants originating from LSH, including *Spherical LSH* [170] for dealing with ℓ_2-normalized data vectors and *Comparison Hadamard Random Projection* [181] (one version of sparse random projection) based LSH for accelerating dot-product computations during enforcing random projections in LSH. Another related technique named *Shift Invariant Kernel Hashing* (SIKH) was proposed in [142]. Although enjoying asymptotic theoretical properties, LSH-related methods require long binary codes to achieve good precision. Nonetheless, long codes result in low recall when used for creating a hash lookup table, as the collision probability decreases exponentially with the code length [27][186]. Hence, one usually needs

to set up multiple hash tables to achieve reasonable recall, which leads to longer query time as well as significant increase in storage.

Unlike the data-independent hash generation in LSH-based algorithms, more recent methods have focused on learning *data-dependent hash functions*. They try to learn compact hash codes [175] for all database items, leading to faster query time with much less storage. Several methods such as *Boosted Similarity Sensitive Coding* [152], *Restricted Boltzmann Machines* (RBMs) [71][145] (or *Semantic Hashing* [146]), *Spectral Hashing* (SH) [191], *Binary Reconstruction Embeddings* (BRE) [93], and *Semi-Supervised Hashing* (SSH) [186] have been proposed, but learning short codes that yield high search accuracy, especially under an *unsupervised* setting, is still an open question.

Perhaps the most critical shortcoming of the existing unsupervised hashing methods is the need to specify a *global* distance measure. On the contrary, in many real-world applications data nearly live on low-dimensional manifolds, which should be taken into account to capture meaningful nearest neighbors during the code learning process. For these, one can only specify *local* distance measures, while the global distances are automatically determined by the underlying manifolds. In this chapter, we present a graph-based hashing method which automatically discovers the neighborhood structure inherent in the data to learn appropriate compact codes in an unsupervised manner. Our basic idea is motivated by [191] in which the hashing purpose is to embed the input data in a Hamming space such that the neighbors in the original data space remain neighbors in the produced Hamming space.

4.3 Notations

In this section, we first define the notations and symbols that we will use in the rest of this chapter. All notations as well as their definitions are listed in Tables 4.2 and 4.3.

Table 4.2: Table of notations.

Notation	Definition
n	The number of database points
m	The number of anchor points
d	The dimension of database, anchor or query points
i, j	The indices of database or anchor points
$\boldsymbol{x}_i \in \mathbb{R}^d$	The ith database point
$\boldsymbol{u}_j \in \mathbb{R}^d$	The jth anchor point
$\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^n$	The data set
$\mathcal{U} = \{\boldsymbol{u}_j\}_{j=1}^m$	The anchor set
$\boldsymbol{q} \in \mathbb{R}^d$	The query point
$h : \mathbb{R}^d \mapsto \{1, -1\}$	A hash function for a single bit
$\operatorname{sgn}(x) \in \{1, -1\}$	The sign function that returns 1 for $x > 0$ and -1 otherwise
$A = (A_{ij})_{i,j} \in \mathbb{R}^{n \times n}$	The adjacency (affinity) matrix of a neighborhood graph
$D = \operatorname{diag}(A\mathbf{1}) \in \mathbb{R}^{n \times n}$	The diagonal node-degree matrix of a neighborhood graph
$L = D - A \in \mathbb{R}^{n \times n}$	The graph Laplacian matrix
$\hat{A} = (\hat{A}_{ij})_{i,j} \in \mathbb{R}^{n \times n}$	The adjacency (affinity) matrix of an Anchor Graph
$\hat{L} = I - \hat{A} \in \mathbb{R}^{n \times n}$	The Anchor Graph Laplacian matrix
$\mathcal{D} : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$	A distance function defined in $\mathbb{R}^d$
t	The bandwidth parameter
s	The number of nearest anchors in $\mathcal{U}$ for each data point
$\langle i \rangle \subset [1 : m]$	The set of indices of s nearest anchors in $\mathcal{U}$ for $\boldsymbol{x}_i$
$Z = (Z_{ij})_{i,j} \in \mathbb{R}^{n \times m}$	The data-to-anchor affinity matrix between $\mathcal{X}$ and $\mathcal{U}$
$\Lambda = \operatorname{diag}(Z^\top \mathbf{1}) \in \mathbb{R}^{m \times m}$	The diagonal anchor-degree matrix of an Anchor Graph
r	The number of hash bits
k	The index of hash bits
$Y \in \{1, -1\}^{n \times r}$	The Hamming embedding matrix
$h_k : \mathbb{R}^d \mapsto \{1, -1\}$	The hash function for the kth bit
$\boldsymbol{y}_k \in \mathbb{R}^n$	The kth Anchor Graph eigenvector
$\boldsymbol{w}_k \in \mathbb{R}^m$	The kth projection vector in the anchor space
$Y = [\boldsymbol{y}_1, \cdots, \boldsymbol{y}_r] \in \mathbb{R}^{n \times r}$	The eigenvector matrix of the Anchor Graph Laplacian
$W = [\boldsymbol{w}_1, \cdots, \boldsymbol{w}_r] \in \mathbb{R}^{m \times r}$	The projection matrix in the anchor space

Table 4.3: Table of notations (continued).

Notation	Definition
$\boldsymbol{z}(\boldsymbol{x}) \in \mathbb{R}^m$	The data-to-anchor mapping
$\phi_k(\boldsymbol{x}) = \boldsymbol{w}_k^\top \boldsymbol{z}(\boldsymbol{x}) \in \mathbb{R}$	The kth Anchor Graph eigenfunction
$b_k^+ \in \mathbb{R}$	The kth positive threshold for positive elements in $\boldsymbol{y}_k$
$b_k^- \in \mathbb{R}$	The kth negative threshold for negative elements in $\boldsymbol{y}_k$

4.4 Anchor Graph Hashing (AGH)

4.4.1 Preview of AGH

Solving the graph hashing problem mentioned in Section 4.2 requires three main steps: (i) building a neighborhood graph, e.g., a kNN graph, involving all n data points from the database $\mathcal{X}$, which costs $O(dn^2)$ time, (ii) computing r eigenvectors of the graph Laplacian of the built sparse graph, which costs $O(rn)$ time, and (iii) extending r eigenvectors to any unseen (novel) data point, which also costs $O(rn)$ time. Unfortunately, step (i) is intractable for offline training while step (iii) is infeasible for online hashing given very large n up to millions. To avoid these computational bottlenecks, Spectral Hashing (SH) [191] made a strong assumption that the input data are uniformly distributed at each dimension. This leads to a simple analytical eigenfunction solution of 1-D Laplacians, but the manifold structure of the original data is almost ignored, substantially weakening the claimed "graph/manifold" theme of SH.

On the contrary, in this section, we present a novel unsupervised hashing approach named *Anchor Graph Hashing* (AGH) [117] to address both of the above bottlenecks. We build an approximate neighborhood graph using *Anchor Graphs* (see Chapter 2), in which the similarity between a pair of data points is measured with respect to a small number of anchors (typically a few hundred). The resulting approximate neighborhood graph is built in $O(n)$ time and is sufficiently sparse with performance approaching to the exact neighborhood graph – kNN graph as the number of anchors increases. Because of the low-rank property of an Anchor Graph's adjacency matrix, our approach can solve the graph Laplacian eigenvectors in linear time. One critical requirement to make graph-based hashing

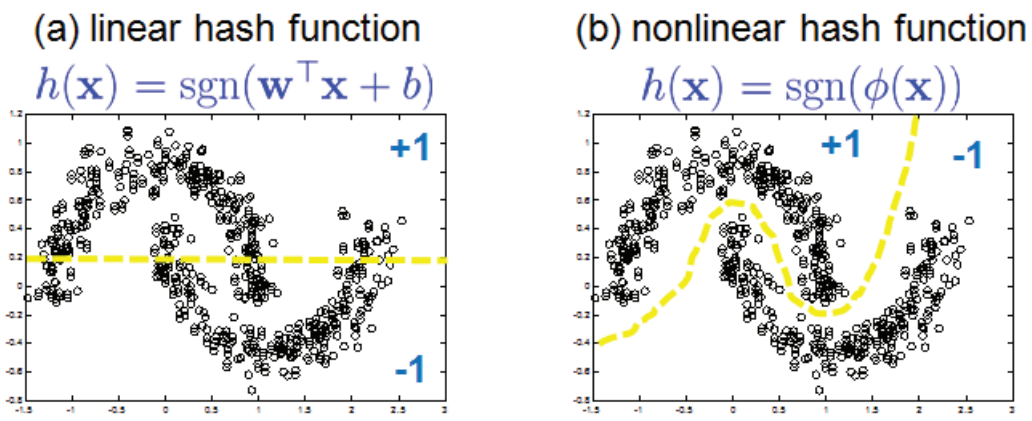


Figure 4.3: Linear vs. nonlinear hash function. (a) A standard linear hash function $h(\boldsymbol{x}) = \mathrm{sgn}(\boldsymbol{w}^\top \boldsymbol{x} + b)$ used by many hashing algorithms, which cuts off the two manifolds. (b) The nonlinear hash function $h(\boldsymbol{x}) = \mathrm{sgn}(\phi(\boldsymbol{x}))$ used by our proposed AGH, which adaptively yields hash bits along the two manifolds.

practical is the ability to generate hash codes for unseen data points. This is known as *out-of-sample extension* in the literature. In this section, we show that the eigenvectors of the Anchor Graph Laplacian can be extended to the generalized eigenfunctions in constant time, thus leading to fast code generation for any incoming data. Finally, to deal with the problem of poor quality of hash functions associated with the higher (less smooth) eigenfunctions of the graph Laplacian, we develop a hierarchical threshold learning procedure in which each eigenfunction yields multiple bits. Therefore, one avoids picking higher eigenfunctions to generate more bits, and bottom (smoother) few eigenfunctions are visited multiple times. We propose a simple method for optimizing the thresholds to obtain multiple bits.

One interesting characteristic of the proposed novel hashing method AGH is that it tends to capture semantic neighborhoods. In other words, data points that are close in the Hamming space, produced by AGH, tend to share similar semantic labels. This is because for many real-world applications close-by points on a manifold tend to share similar semantic labels, and AGH preserves the neighborhood structure found by a graph which reveals underlying manifolds, especially for large-scale data collections. Figure 4.3 visually illustrates the prominent trait of AGH. AGH provides the nonlinear hash function $h(\boldsymbol{x}) =$

$\text{sgn}(\phi(\boldsymbol{x}))$ ($\phi(\boldsymbol{x})$ is an eigenfunction of the graph Laplacian) that yields the same hash bit for points on the same manifold, thereby capturing the existing semantic neighborhoods residing on the underlying manifolds. In contrast, traditional linear hash functions with the form $h(\boldsymbol{x}) = \text{sgn}(\boldsymbol{w}^\top \boldsymbol{x} + b)$ obviously lose the manifold structure, although widely employed by Locality-Sensitive Hashing (LSH) [27][4], PCA Hashing (PCAH) [186], Metric Hashing [95], Semi-Supervised Hashing (SSH) [186], Iterative Quantization (ITQ) [56], etc.

The key characteristic of AGH, i.e., preferring semantically similar neighbors, is validated by extensive experiments carried out on two datasets. The experimental results show that AGH even outperforms exhaustive linear scan (using the commonly used ℓ_2 distance in the feature space) in terms of search accuracy, while almost all of the other hashing methods are inferior to exhaustive linear scan.

4.4.2 Formulation

As stated before, the goal of graph hashing is to learn binary codes such that neighbors found by a neighborhood graph built in the input feature space are mapped to nearby codes in the Hamming space. Suppose that $A_{ij} \geq 0$ is the similarity (or affinity) between a data pair $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ in the input space $\mathbb{R}^d$. Then, similar to Spectral Hashing (SH) [191], our method seeks an r-bit Hamming embedding $Y \in \{1, -1\}^{n \times r}$ [1] for n points in the data set $\mathcal{X}$ by optimizing the following constrained problem:

$$\begin{aligned} \min_{Y} \quad & \frac{1}{2} \sum_{i,j=1}^{n} \|Y_i - Y_j\|^2 A_{ij} = \text{tr}(Y^\top L Y) \\ \text{s.t.} \quad & Y \in \{1, -1\}^{n \times r} \\ & \mathbf{1}^\top Y = 0 \\ & Y^\top Y = n I_{r \times r} \end{aligned} \tag{4.1}$$

where Y_i is the ith row of Y representing the r-bit code for point $\boldsymbol{x}_i$, A is the $n \times n$ graph adjacency (affinity) matrix, and $D = \text{diag}(A\mathbf{1})$ (where $\mathbf{1} = [1, \cdots, 1]^\top \in \mathbb{R}^n$) is the diagonal node-degree matrix of the graph. The graph Laplacian matrix is then defined by $L = D - A$.

[1] We treat '0' bit as '-1' bit for formulation and training; in the implementations of data coding and hashing, we use '0' bit back since converting -1/1 codes to 0/1 codes is a trivial shift and scaling operation.

The constraint $\mathbf{1}^\top Y = 0$ is imposed to maximize the information of each bit, which occurs when each bit leads to balanced partitioning of the data. Another constraint $Y^\top Y = nI_{r\times r}$ forces r bits to be mutually uncorrelated in order to minimize redundancy among bits.

The above problem is an integer program, equivalent to balanced graph partitioning even for a single bit. This is known to be *NP-hard*. To make eq. (4.1) tractable, one can apply *spectral relaxation* [153] to drop the integer constraint and allow $Y \in \mathbb{R}^{n\times r}$. With this, the solution Y is given by r eigenvectors of ℓ_2 norm $\sqrt{n}$ corresponding to r smallest eigenvalues (ignoring eigenvalue 0) of the graph Laplacian L. Y thereby forms an r-dimensional spectral embedding in analogy to *Laplacian Eigenmap* [10]. Note that the excluded bottom most eigenvector associated with eigenvalue 0 is $\mathbf{1}$ if the underlying graph is connected. Since all of the remaining eigenvectors are orthogonal to it, $\mathbf{1}^\top Y = 0$ holds. Even though the graph is not connected, by discarding the eigenvectors associated with all zero eigenvalues one can still achieve $\mathbf{1}^\top Y = 0$. An approximate solution simply given by $\mathrm{sgn}(Y)$ generates the final desired hash codes, forming a Hamming embedding from $\mathbb{R}^d$ to $\{1, -1\}^r$.

Although conceptually simple, the main bottleneck in the above formulation is computation. The cost of building the needed neighborhood graph and the associated graph Laplacian is $O(dn^2)$, which is intractable for large n up to millions. To avoid the computational bottleneck, unlike the restrictive assumption of a separable uniform data distribution made by SH, in the next subsection, we present a more general approach based on Anchor Graphs that have been presented in Chapter 2. The core idea is to directly approximate the exact neighborhood graph as well as its associated sparse adjacency matrix.

4.4.3 Anchor Graphs

Above all, let us quickly review and sketch our proposed Anchor Graphs. An Anchor Graph uses a small set of m data points called *anchors*, not necessarily belonging to the raw database $\mathcal{X}$, to approximate the neighborhood structure existing over $\mathcal{X}$. Similarities between all n database points are measured with respect to these m anchors, and the true adjacency (or affinity) matrix A is approximated using the data-to-anchor similarities. First, K-means clustering is performed on n data points to obtain m ($m \ll n$) cluster centers $\mathcal{U} = \{\boldsymbol{u}_j \in \mathbb{R}^d\}_{j=1}^m$ that act as anchor points. In practice, running K-means on a

small subsample of the database with very few iterations (less than 10) is sufficient. This makes clustering very fast, thus speeding up training significantly. One can also alternatively use any other efficient clustering algorithms instead of K-means. Next, the Anchor Graph defines the *truncated* similarities Z_{ij}'s between all n data points and m anchors as,

$$Z_{ij} = \begin{cases} \frac{\exp(-\mathcal{D}^2(\boldsymbol{x}_i, \boldsymbol{u}_j)/t)}{\sum_{j' \in \langle i \rangle} \exp(-\mathcal{D}^2(\boldsymbol{x}_i, \boldsymbol{u}_{j'})/t)}, & \forall j \in \langle i \rangle \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

where $\langle i \rangle \subset [1:m]$ denotes the indices of s ($s \ll m$) nearest anchors of point $\boldsymbol{x}_i$ in $\mathcal{U}$ according to a distance function $\mathcal{D}(\cdot)$ such as ℓ_2 distance, and t denotes the bandwidth parameter. Note that the matrix $Z \in \mathbb{R}^{n \times m}$ is highly sparse. Each row Z_i contains only s nonzero entries which sum to 1.

Derived from random walks across data points and anchors, the Anchor Graph provides a powerful approximation to the adjacency matrix A as $\hat{A} = Z\Lambda^{-1}Z^\top$ where $\Lambda = \text{diag}(Z^\top \mathbf{1}) \in \mathbb{R}^{m \times m}$ is the diagonal anchor-degree matrix of the Anchor Graph. The approximate graph adjacency matrix $\hat{A}$ has three key properties: 1) $\hat{A}$ is nonnegative and sparse since Z is very sparse; 2) $\hat{A}$ is low-rank (its rank is at most m), so an Anchor Graph does not compute $\hat{A}$ explicitly but instead keeps its low-rank form; 3) $\hat{A}$ is a doubly stochastic matrix, i.e., has unit row and column sums, so the resulting Anchor Graph Laplacian is $\hat{L} = I - \hat{A}$. Properties 2) and 3) are critical, which allow efficient eigenfunction extensions of Anchor Graph Laplacians, as shown in the next subsection. The memory cost of an Anchor Graph is $O(sn)$ for storing Z, and the time cost is $O(dmnT + dmn)$ in which $O(dmnT)$ originates from K-means clustering with T iterations. Since $m \ll n$, the cost for constructing an Anchor Graph is linear in n, which is far more efficient than constructing a kNN graph that has a quadratic cost $O(dn^2)$ though provides exact neighborhoods over $\mathcal{X}$.

Since the graph Laplacian matrix of the Anchor Graph is $\hat{L} = I - \hat{A}$, the required r Anchor Graph Laplacian eigenvectors are also eigenvectors of $\hat{A}$ but associated with the r largest eigenvalues (ignoring eigenvalue 1 which corresponds to eigenvalue 0 of $\hat{L}$). One can easily solve the eigenvectors of $\hat{A}$ by utilizing its low-rank property and Proposition 4 presented in Chapter 3.

Specifically, in order to achieve the eigen-system of the big $n \times n$ matrix $\hat{A} = Z\Lambda^{-1}Z^\top$, we

solve the eigen-system of a small $m \times m$ matrix $M = \Lambda^{-1/2} Z^\top Z \Lambda^{-1/2}$ accordingly, resulting in r ($< m$) eigenvector-eigenvalue pairs $\{(\boldsymbol{v}_k, \sigma_k)\}_{k=1}^r$ where $1 > \sigma_1 \geq \cdots \geq \sigma_r > 0$. After expressing $V = [\boldsymbol{v}_1, \cdots, \boldsymbol{v}_r] \in \mathbb{R}^{m \times r}$ (V is column-orthonormal) and $\Sigma = \text{diag}(\sigma_1, \cdots, \sigma_r) \in \mathbb{R}^{r \times r}$, we obtain the desired spectral embedding matrix Y as

$$Y = \sqrt{n} Z \Lambda^{-1/2} V \Sigma^{-1/2} = ZW, \tag{4.3}$$

which satisfies $\mathbf{1}^\top Y = 0$ and $Y^\top Y = n I_{r \times r}$. It is interesting to find out that hashing based on Anchor Graphs can be interpreted as first nonlinearly transforming each input data point $\boldsymbol{x}_i$ to Z_i by computing its sparse similarities to anchor points and second linearly projecting Z_i onto the vectors in $W = \sqrt{n} \Lambda^{-1/2} V \Sigma^{-1/2} = [\boldsymbol{w}_1, \cdots, \boldsymbol{w}_r] \in \mathbb{R}^{m \times r}$ where $\boldsymbol{w}_k = \sqrt{n/\sigma_k} \Lambda^{-1/2} \boldsymbol{v}_k$. Finally, we accomplish the target Hamming embedding as $\text{sgn}(Y)$.

To verify the quality of the Hamming embedding solved from the Anchor Graph Laplacian eigenvectors, we use the expanded two-moon toy dataset, which replicates the original dataset introduced in Chapter 2 ten times with random small shifts, to exhibit the quality of Hamming embeddings acquired by (a) Idealized Graph Hashing that constructs a 10NN graph and computes two smoothest graph Laplacian eigenvectors, (b) our Anchor Graph Hashing that constructs an Anchor Graph with $m = 100, s = 2$ and computes two smoothest Anchor Graph Laplacian eigenvectors, and (c) Spectral Hashing that computes two smoothest pseudo graph Laplacian eigenfunctions. The three Hamming embeddings are shown in Figure 4.4. Figure 4.4 reveals that our proposed AGH produces a remarkably better embedding than Idealized Graph Hashing because at the second dimension AGH yields more balanced bits than the latter which is impractical at large scale. The Hamming embedding generated by Spectral Hashing is very poor, losing smoothness along manifolds.

4.4.4 Eigenfunction Generalization

The procedure given in eq. (4.3) generates binary codes only for those database points that are available during training. But, for the purpose of hashing, one needs to seek a generic hash function $h : \mathbb{R}^d \mapsto \{1, -1\}$ which can take any arbitrary data point as input. For this, one needs to generalize the eigenvectors of the Anchor Graph Laplacian to the eigenfunctions $\{\phi_k : \mathbb{R}^d \mapsto \mathbb{R}\}_{k=1}^r$ such that the required hash functions can be simply

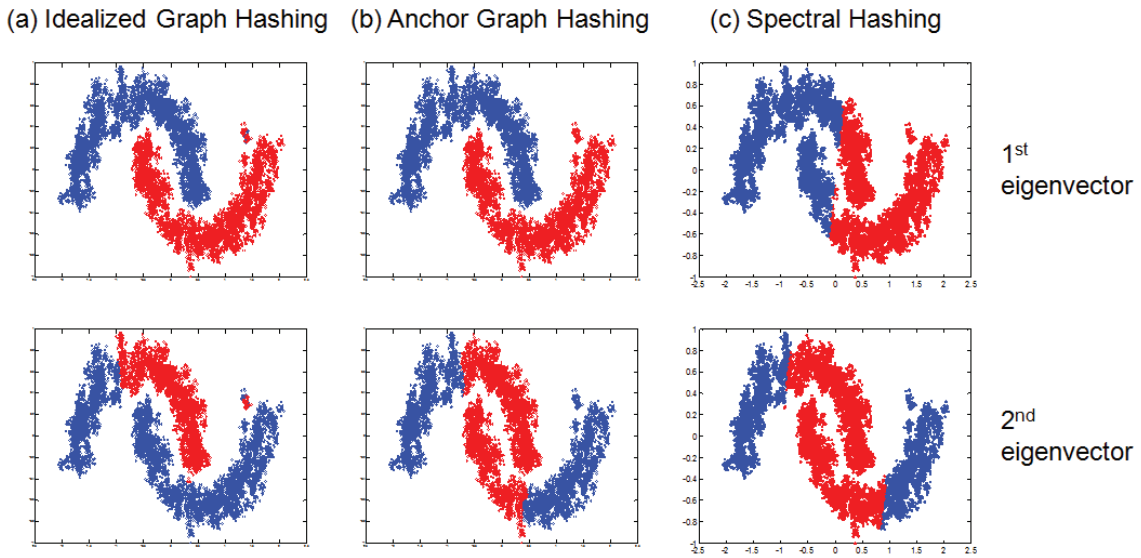


Figure 4.4: Two-bit Hamming embeddings of two-moon toy data consisting of 12,000 2D points. The trivial eigenvector $\mathbf{1}$ has been excluded in each embedding. The blue color denotes '1' bits (positive eigenvector entries) and the red color denotes '-1' bits (negative eigenvector entries). (a) The Hamming embedding of Idealized Graph Hashing, (b) the Hamming embedding of Anchor Graph Hashing, and (c) the Hamming embedding of Spectral Hashing.

obtained as $h_k(\boldsymbol{x}) = \text{sgn}(\phi_k(\boldsymbol{x}))$ $(k = 1, \cdots, r)$. We create the "out-of-sample" extension of the Anchor Graph Laplacian eigenvectors Y to their corresponding eigenfunctions using the Nyström method [193][13] which is depicted by Proposition 7.

Proposition 7. *Suppose a valid kernel function* $\kappa : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ *and a kernel matrix* $K = (\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j))_{i,j} \in \mathbb{R}^{n \times n}$. *If* $(\boldsymbol{y}, \sigma)$ *is an eigenvector-eigenvalue pair of* K *such that* $\|\boldsymbol{y}\|^2 = n$, *then the Nyström-approximated eigenfunction of* κ *extended from* $\boldsymbol{y}$ *is*

$$\phi(\boldsymbol{x}) = \frac{1}{\sigma}[\kappa(\boldsymbol{x}, \boldsymbol{x}_1), \cdots, \kappa(\boldsymbol{x}, \boldsymbol{x}_n)]\boldsymbol{y}. \tag{4.4}$$

Through applying Proposition 7, we give Theorem 8 which derives an analytical form to each Anchor Graph Laplacian eigenfunction ϕ_k.

Theorem 8. *Given m anchor points $\mathcal{U} = \{\boldsymbol{u}_j\}_{j=1}^m$ and any sample $\boldsymbol{x}$, define a data-to-anchor mapping $\boldsymbol{z} : \mathbb{R}^d \mapsto \mathbb{R}^m$ as follows*

$$\boldsymbol{z}(\boldsymbol{x}) = \frac{\left[\delta_1 \exp(-\frac{\mathcal{D}^2(\boldsymbol{x},\boldsymbol{u}_1)}{t}), \cdots, \delta_m \exp(-\frac{\mathcal{D}^2(\boldsymbol{x},\boldsymbol{u}_m)}{t})\right]^\top}{\sum_{j=1}^m \delta_j \exp(-\frac{\mathcal{D}^2(\boldsymbol{x},\boldsymbol{u}_j)}{t})}, \tag{4.5}$$

where $\delta_j \in \{1, 0\}$ and $\delta_j = 1$ if and only if anchor $\boldsymbol{u}_j$ is one of s nearest anchors of sample $\boldsymbol{x}$ in $\mathcal{U}$ according to the distance function $\mathcal{D}(\cdot)$. Then the Nyström eigenfunction extended from the Anchor Graph Laplacian eigenvector $\boldsymbol{y}_k = Z\boldsymbol{w}_k$ is

$$\phi_k(\boldsymbol{x}) = \boldsymbol{w}_k^\top \boldsymbol{z}(\boldsymbol{x}). \tag{4.6}$$

Proof. First, we check that ϕ_k and $\boldsymbol{y}_k$ (the kth column of Y) overlap on all n training samples. If $\boldsymbol{x}_i$ is in the training set $\mathcal{X}$, then $\boldsymbol{z}(\boldsymbol{x}_i) = Z_i^\top$ where Z_i is the ith row of Z. Thus, $\phi_k(\boldsymbol{x}_i) = \boldsymbol{w}_k^\top Z_i^\top = Z_i \boldsymbol{w}_k = Y_{ik}$. Note that Y solved by eq. (4.3) satisfies $Y^\top Y = nI_{r\times r}$ and then $\|\boldsymbol{y}_k\|^2 = n$.

The Anchor Graph's adjacency matrix $\hat{A} = Z\Lambda^{-1}Z^\top$ is positive semidefinite with each entry defined as $\hat{A}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{z}^\top(\boldsymbol{x}_i)\Lambda^{-1}\boldsymbol{z}(\boldsymbol{x}_j)$, so $\hat{A}(\cdot)$ forms a valid kernel. For any unseen sample $\boldsymbol{x}$, the Nyström method extends $\boldsymbol{y}_k$ to $\phi_k(\boldsymbol{x})$ as the weighted summation over n elements of $\boldsymbol{y}_k$, that is,

$$\phi_k(\boldsymbol{x}) = \frac{1}{\sigma_k}\left[\hat{A}(\boldsymbol{x}, \boldsymbol{x}_1), \cdots, \hat{A}(\boldsymbol{x}, \boldsymbol{x}_n)\right]\boldsymbol{y}_k.$$

Since $\boldsymbol{w}_k = \sqrt{n/\sigma_k}\Lambda^{-1/2}\boldsymbol{v}_k$ and $M\boldsymbol{v}_k = \sigma_k\boldsymbol{v}_k$, we can show that

$$\begin{aligned}
\phi_k(\boldsymbol{x}) &= \frac{1}{\sigma_k}\boldsymbol{z}^\top(\boldsymbol{x})\Lambda^{-1}[\boldsymbol{z}(\boldsymbol{x}_1), \cdots, \boldsymbol{z}(\boldsymbol{x}_n)]\boldsymbol{y}_k \\
&= \frac{1}{\sigma_k}\boldsymbol{z}^\top(\boldsymbol{x})\Lambda^{-1}Z^\top\boldsymbol{y}_k \\
&= \frac{1}{\sigma_k}\boldsymbol{z}^\top(\boldsymbol{x})\Lambda^{-1}Z^\top Z\boldsymbol{w}_k \\
&= \frac{1}{\sigma_k}\boldsymbol{z}^\top(\boldsymbol{x})\Lambda^{-1}Z^\top Z\sqrt{\frac{n}{\sigma_k}}\Lambda^{-1/2}\boldsymbol{v}_k \\
&= \sqrt{\frac{n}{\sigma_k^3}}\boldsymbol{z}^\top(\boldsymbol{x})\Lambda^{-1/2}\left(\Lambda^{-1/2}Z^\top Z\Lambda^{-1/2}\boldsymbol{v}_k\right) \\
&= \sqrt{\frac{n}{\sigma_k^3}}\boldsymbol{z}^\top(\boldsymbol{x})\Lambda^{-1/2}\left(M\boldsymbol{v}_k\right) \\
&= \boldsymbol{z}^\top(\boldsymbol{x})\left(\sqrt{\frac{n}{\sigma_k}}\Lambda^{-1/2}\boldsymbol{v}_k\right) \\
&= \boldsymbol{z}^\top(\boldsymbol{x})\boldsymbol{w}_k = \boldsymbol{w}_k^\top\boldsymbol{z}(\boldsymbol{x}),
\end{aligned}$$

which completes the proof. □

By making use of Theorem 8, Anchor Graph Hashing (AGH) offers r hash functions as:

$$h_k(\boldsymbol{x}) = \mathrm{sgn}(\phi_k(\boldsymbol{x})) = \mathrm{sgn}(\boldsymbol{w}_k^\top\boldsymbol{z}(\boldsymbol{x})),\ k = 1, \cdots, r. \tag{4.7}$$

In addition to the time for Anchor Graph construction, AGH needs $O(m^2n + srn)$ time for solving r Anchor Graph Laplacian eigenvectors retained in the spectral embedding matrix Y, and $O(rn)$ time for compressing Y into binary codes. Under the online search scenario, AGH needs to save the binary codes $\mathrm{sgn}(Y)$ of n training samples, m anchors $\mathcal{U}$, and the projection matrix W in memory. Hashing any test sample $\boldsymbol{x}$ only costs $O(dm + sr)$ time which is dominated by the construction of a sparse vector $\boldsymbol{z}(\boldsymbol{x})$ in the anchor space $\mathbb{R}^m$.

Remarks. 1) Though graph Laplacian eigenvectors of the Anchor Graph are not as accurate as those of an exact neighborhood graph, e.g., kNN graph, they provide good performance when used for hashing. Exact neighborhood graph construction is infeasible at large scale. Even if one could get r graph Laplacian eigenvectors of an exact graph, the cost of calculating their Nyström extensions to a novel sample is $O(rn)$, which is still infeasible for online hashing requirement. 2) Free from any restrictive data distribution assumption, AGH solves Anchor Graph Laplacian eigenvectors in linear time and extends them to eigenfunctions in constant time depending only on constants m and s.

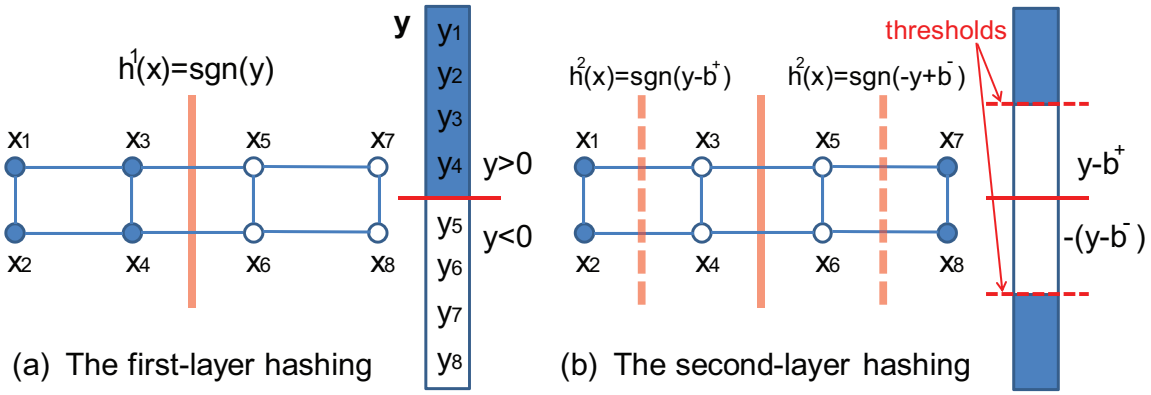


Figure 4.5: Hierarchical hashing on a data graph. $x_1, \cdots, x_8$ are data points and $\boldsymbol{y}$ is a graph Laplacian eigenvector. The data points of filled circles take '1' hash bit and the others take '-1' hash bit. The entries with dark color in $\boldsymbol{y}$ are positive and the others are negative. (a) The first-layer hash function h^1 uses threshold 0 ; (b) the second-layer hash functions h^2 use thresholds b^+ and b^-.

4.4.5 Hierarchical Hashing

As illustrated before, to generate r-bit hash codes, we need to access r graph Laplacian eigenvectors, but not all eigenvectors are equally suitable for hashing especially when r increases. From a geometric point of view, the intrinsic dimension of data manifolds is usually low, so a low-dimensional spectral embedding containing the lower graph Laplacian eigenvectors is desirable. Moreover, [153] discussed that the error made in converting the real-valued eigenvector $\boldsymbol{y}_k$ to the optimal integer solution $\boldsymbol{y}_k^* \in \{1, -1\}^n$ accumulates rapidly as k increases. In this subsection, we develop a simple hierarchical scheme that gives the priority to the lower graph Laplacian eigenvectors/eigenfunctions by revisiting them to generate multiple bits.

To explain the basic idea, let us look at a toy example shown in Figure 4.5. To generate the first bit, the graph Laplacian eigenvector $\boldsymbol{y}$ partitions the graph by the red line using threshold zero. Due to thresholding, there is always a possibility that neighboring points close to the boundary (i.e., threshold) are hashed to different bits (e.g., points x_3 and x_5). To address this issue, we conduct hierarchical hashing of two layers in which the second-layer hashing tries to correct the boundary errors caused by the previous hashing. Intuitively, we

form the second layer by further dividing each partition created by the first layer. In other words, the positive and negative entries in $\boldsymbol{y}$ are thresholded at b^+ and b^-, respectively. Hence, the hash bits at the second layer are generated by $\text{sgn}(y_i - b^+)$ when $y_i > 0$ and $\text{sgn}(-y_i + b^-)$ otherwise. Figure 4.5(b) shows that x_3 and x_5 are hashed to the same bit at the second layer. Next we describe how one can learn the optimal thresholds for the second-layer hashing.

We propose to optimize the two thresholds b^+ and b^- from the perspective of balanced graph partitioning. Let us form a thresholded vector $\begin{bmatrix} \boldsymbol{y}^+ - b^+\mathbf{1}^+ \\ -\boldsymbol{y}^- + b^-\mathbf{1}^- \end{bmatrix}$ whose sign gives a hash bit for each training sample during the second-layer hashing. Two vectors $\boldsymbol{y}^+$ of length n^+ and $\boldsymbol{y}^-$ of length n^- correspond to the positive and negative entries in $\boldsymbol{y}$, respectively. Two constant vectors $\mathbf{1}^+$ and $\mathbf{1}^-$ contain n^+ and n^- 1 entries accordingly ($n^+ + n^- = n$). Similar to the first layer, we would like to find such thresholds that minimize the cut value of the graph Laplacian with the target thresholded vector while maintaining a balanced partitioning, i.e.,

$$\begin{aligned} \min_{b^+,b^-} \ \Gamma(b^+,b^-) &= \begin{bmatrix} \boldsymbol{y}^+ - b^+\mathbf{1}^+ \\ -\boldsymbol{y}^- + b^-\mathbf{1}^- \end{bmatrix}^\top L \begin{bmatrix} \boldsymbol{y}^+ - b^+\mathbf{1}^+ \\ -\boldsymbol{y}^- + b^-\mathbf{1}^- \end{bmatrix} \\ \text{s.t.} \ \ \mathbf{1}^\top &\begin{bmatrix} \boldsymbol{y}^+ - b^+\mathbf{1}^+ \\ -\boldsymbol{y}^- + b^-\mathbf{1}^- \end{bmatrix} = 0. \end{aligned} \tag{4.8}$$

Defining vector $\overline{\boldsymbol{y}} = \begin{bmatrix} \boldsymbol{y}^+ \\ -\boldsymbol{y}^- \end{bmatrix}$ and arranging L into $\begin{bmatrix} L_{+\cdot} \\ L_{-\cdot} \end{bmatrix} = \begin{bmatrix} L_{++} & L_{+-} \\ L_{-+} & L_{--} \end{bmatrix}$ corresponding to the positive and negative entries in $\boldsymbol{y}$, we optimize b^+ and b^- by zeroing the derivatives of the objective in eq. (4.8). After simple algebraic manipulation, one can show that

$$b^+ + b^- = \frac{(\mathbf{1}^+)^\top L_{+\cdot}\overline{\boldsymbol{y}}}{(\mathbf{1}^+)^\top L_{++}\mathbf{1}^+} \equiv \beta. \tag{4.9}$$

On the other hand, combining the fact that $\mathbf{1}^\top\boldsymbol{y} = 0$ with the constraint in eq. (4.8) leads to:

$$n^+b^+ - (n - n^+)b^- = (\mathbf{1}^+)^\top\boldsymbol{y}^+ - (\mathbf{1}^-)^\top\boldsymbol{y}^- = 2(\mathbf{1}^+)^\top\boldsymbol{y}^+. \tag{4.10}$$

We use the Anchor Graph's adjacency matrix $\hat{A} = Z\Lambda^{-1}Z^\top$ for the computations involving the graph Laplacian L. Suppose, $\boldsymbol{y}$ is an eigenvector of $\hat{A}$ with eigenvalue σ such

that $\hat{A}\boldsymbol{y} = \sigma\boldsymbol{y}$. Then, we have $\hat{A}_{++}\boldsymbol{y}^+ + \hat{A}_{+-}\boldsymbol{y}^- = \sigma\boldsymbol{y}^+$. Thus, from eq. (4.9),

$$\begin{aligned}
\beta &= \frac{(\mathbf{1}^+)^\top L_{+.}\overline{\boldsymbol{y}}}{(\mathbf{1}^+)^\top L_{++}\mathbf{1}^+} = \frac{(\mathbf{1}^+)^\top \left((I - \hat{A}_{++})\boldsymbol{y}^+ + \hat{A}_{+-}\boldsymbol{y}^-\right)}{(\mathbf{1}^+)^\top (I - \hat{A}_{++})\mathbf{1}^+} \\
&= \frac{(\mathbf{1}^+)^\top (\boldsymbol{y}^+ - \hat{A}_{++}\boldsymbol{y}^+ + \sigma\boldsymbol{y}^+ - \hat{A}_{++}\boldsymbol{y}^+)}{n^+ - (\mathbf{1}^+)^\top \hat{A}_{++}\mathbf{1}^+} \\
&= \frac{(\sigma+1)(\mathbf{1}^+)^\top \boldsymbol{y}^+ - 2(\mathbf{1}^+)^\top \hat{A}_{++}\boldsymbol{y}^+}{n^+ - (\mathbf{1}^+)^\top \hat{A}_{++}\mathbf{1}^+} \\
&= \frac{(\sigma+1)(\mathbf{1}^+)^\top \boldsymbol{y}^+ - 2(Z_+^\top \mathbf{1}^+)^\top \Lambda^{-1}(Z_+^\top \boldsymbol{y}^+)}{n^+ - (Z_+^\top \mathbf{1}^+)^\top \Lambda^{-1}(Z_+^\top \mathbf{1}^+)},
\end{aligned} \tag{4.11}$$

where $Z_+ \in \mathbb{R}^{n^+ \times m}$ is the sub-matrix of $Z = \begin{bmatrix} Z_+ \\ Z_- \end{bmatrix}$ corresponding to $\boldsymbol{y}^+$. By putting eq. (4.9)-(4.11) together, we solve the target thresholds as

$$\begin{cases} b^+ = \dfrac{2(\mathbf{1}^+)^\top \boldsymbol{y}^+ + (n - n^+)\beta}{n} \\ b^- = \dfrac{-2(\mathbf{1}^+)^\top \boldsymbol{y}^+ + n^+\beta}{n}, \end{cases} \tag{4.12}$$

which requires $O(mn^+)$ time.

Now we give the two-layer hash functions for AGH to yield an r-bit code using the first $r/2$ graph Laplacian eigenvectors of the Anchor Graph. Conditioned on the outputs of the first-layer hash functions $\{h_k^{(1)}(\boldsymbol{x}) = \operatorname{sgn}\left(\boldsymbol{w}_k^\top \boldsymbol{z}(\boldsymbol{x})\right)\}_{k=1}^{r/2}$, the second-layer hash functions are generated dynamically as follows for $k = 1, \cdots, r/2$,

$$h_k^{(2)}(\boldsymbol{x}) = \begin{cases} \operatorname{sgn}\left(\boldsymbol{w}_k^\top \boldsymbol{z}(\boldsymbol{x}) - b_k^+\right) & \text{if} \quad h_k^{(1)}(\boldsymbol{x}) = 1 \\ \\ \operatorname{sgn}\left(-\boldsymbol{w}_k^\top \boldsymbol{z}(\boldsymbol{x}) + b_k^-\right) & \text{if} \quad h_k^{(1)}(\boldsymbol{x}) = -1 \end{cases} \tag{4.13}$$

in which (b_k^+, b_k^-) are calculated from each eigenvector $\boldsymbol{y}_k = Z\boldsymbol{w}_k$. Compared to r one-layer hash functions $\{h_k^{(1)}\}_{k=1}^r$, the proposed two-layer hash functions for r bits actually use the $r/2$ lower eigenvectors twice. Hence, they avoid using the higher eigenvectors which can potentially be of low quality for partitioning and hashing. The experiments conducted in Section 4.5 reveal that with the same number of bits, AGH using two-layer hash functions achieves comparable precision but much higher recall than using one-layer hash functions alone (see Figure 4.8(b)(c)). Of course, one can extend hierarchical hashing to more than

two layers. However, the accuracy of the resulting hash functions will depend on whether repeatedly partitioning the existing eigenvectors gives more informative bits than those from picking new eigenvectors.

4.4.6 Complexity Analysis

For the same budget of r hash bits, we analyze two hashing algorithms which are presented in subsections 4.4.4 and 4.4.5 and both based on Anchor Graphs with the same graph construction parameters m and s. For convenience, we name AGH with r one-layer hash functions $\{h_k^{(1)}\}_{k=1}^{r}$ *1-AGH*, and AGH with r two-layer hash functions $\{h_k^{(1)}, h_k^{(2)}\}_{k=1}^{r/2}$ *2-AGH*, respectively.

Below we give space and time complexities of 1-AGH and 2-AGH.

`Space Complexity:` $O((d+s+r)n)$ in the training phase and $O(rn)$ (binary bits) in the test phase for both of 1-AGH and 2-AGH.

`Time Complexity:` $O(dmnT + dmn + m^2n + (s+1)rn)$ for 1-AGH and $O(dmnT + dmn + m^2n + (s/2+m/2+1)rn)$ for 2-AGH in the training phase; $O(dm+sr)$ for both in the test phase.

To summarize, 1-AGH and 2-AGH both have linear training time and constant query time (m and s are fixed to constants). Under the online search scenario, the memory cost of either is quite efficient, capable of storing samples up to 3.58 billions in 10GB RAM if $r = 24$ bits are used to represent each sample.

4.5 Experiments

4.5.1 Methods and Evaluation Protocols

We evaluate the proposed graph-based unsupervised hashing, both single-layer AGH (1-AGH) and two-layer AGH (2-AGH), on two benchmark datasets: **MNIST** (70K) [102] and **NUS-WIDE** (270K) [31]. Their performance is compared against other popular unsupervised hashing methods including Locality-Sensitive Hashing (LSH) [4], PCA Hashing (PCAH) [186], Unsupervised Sequential Projection Learning for Hashing (USPLH) [186], Iterative Quantization (ITQ) [56], Spectral Hashing (SH) [191], Kernelized Locality-Sensitive

Hashing (KLSH) [94], and Shift-Invariant Kernel Hashing (SIKH) [142]. These methods cover both linear (LSH, PCAH, USPLH and ITQ) and nonlinear (SH, KLSH and SIKH) hashing paradigms. Our AGH methods are nonlinear. We used publicly available codes of USPLH, ITQ, SH and KLSH. To run KLSH, we sample 300 training points to form the empirical kernel mapping and use the same Gaussian kernel as for SIKH. The kernel width parameter is tuned to an appropriate value on each dataset. To run our methods 1-AGH and 2-AGH, we fix the graph construction parameters to $m = 300, s = 2$ on **MNIST** and $m = 300, s = 5$ on **NUS-WIDE**, respectively. We adopt the ℓ_2 distance for the distance function $\mathcal{D}()$ in defining the matrix Z. In addition, we run K-means clustering with $T = 5$ iterations to find anchors on each dataset. All our experiments are run on a workstation with a 2.53 GHz Intel Xeon CPU and 48GB RAM.

We follow two search procedures, i.e., hash lookup and Hamming ranking, for consistent evaluations across the two datasets. Note that evaluations based on the two distinct criteria concentrate on different characteristics of hashing techniques. Hash lookup emphasizes more on real-time search speed since it has constant query time. However, when using longer hash codes and a single hash table, hash lookup often fails because the Hamming code space becomes increasingly sparse and very few samples fall in the same hash bucket. Hence, similar to the experimental setting introduced in [191], we search within a Hamming radius 2 to retrieve potential neighbors for each query. Hamming ranking measures the search quality by ranking database points according to their Hamming distances to the query in the Hamming space. Even though the time complexity of Hamming ranking is linear, it is usually very fast in practice.

4.5.2 Datasets

The well-known **MNIST** dataset [102] consists of 784-dimensional 70,000 samples associated with digits from '0' to '9'. We split this dataset into two subsets: a training set containing $69,000$ samples and a query set of $1,000$ samples. Because this dataset is fully annotated, we define true neighbors as semantic neighbors based on the associated digit labels. The number of hash bits is varied from 8 to 64 in this group of experiments.

The second dataset **NUS-WIDE** [31] contains around $270,000$ web images associated

Figure 4.6: Example images from the **NUS-WIDE** dataset, every two of which respectively come from 12 most frequent tags: 'animal', 'buildings', 'clouds', 'grass', 'person', 'plants', 'sky', 'water', 'window', 'lake', 'ocean' and 'road'.

with 81 ground truth concept tags. Each image is represented by an ℓ_2 normalized 1024-dimensional sparse-coding feature vector [189] computed from dense SIFTs [101]. Unlike **MNIST**, each image in **NUS-WIDE** contains multiple semantic labels (tags). The true neighbors are defined based on whether two images share *at least one* common tag. For evaluation, we consider 21 most frequent tags, such as 'animal', 'buildings', 'person', etc., each of which has abundant relevant images ranging from 5,000 to 30,000. Some example images are displayed in Figure 4.6. We sample uniformly 100 images from each of the selected 21 tags to form a query set of 2,100 images with the rest serving as the training set. The number of hash bits also varies from 8 to 64.

4.5.3 Results

Table 4.4 shows the Hamming ranking performance measured by Mean Average Precision (MAP), training time, and test time for different hashing methods on **MNIST**. For each hashing method, training time refers to the hashing time for encoding all database samples

Table 4.4: Hamming ranking performance on **MNIST**. Mean Average Precision (MAP), training time, and test time are evaluated for each hashing method. r denotes the number of hash bits used in hashing algorithms, and also the number of eigenfunctions used in SE ℓ_2 linear scan. The K-means execution time is 20.1 seconds for training AGH on **MNIST**. All training and test time is recorded in second. At a fixed bit number, two highest MAP values achieved by hashing are displayed in boldface type.

Method	**MNIST (70K)**				
	MAP			**Train Time**	**Test Time**
	$r = 12$	$r = 24$	$r = 48$	$r = 48$	$r = 48$
ℓ_2 Scan	0.4125			–	
SE ℓ_2 Scan	0.6393	0.5269	0.3909	–	–
LSH	0.1488	0.1613	0.2196	1.8	2.1×10^{-5}
PCAH	0.2757	0.2596	0.2242	4.5	2.2×10^{-5}
USPLH	0.4062	0.4699	**0.4930**	163.2	2.3×10^{-5}
ITQ	0.3820	0.4351	0.4408	10.6	2.3×10^{-5}
SH	0.2743	0.2699	0.2453	4.9	4.9×10^{-5}
KLSH	0.2191	0.2555	0.3049	2.9	5.3×10^{-5}
SIKH	0.1632	0.1947	0.1972	0.4	1.3×10^{-5}
1-AGH	**0.5956**	**0.4997**	0.3971	22.9	5.3×10^{-5}
2-AGH	**0.5957**	**0.6738**	**0.6410**	23.2	6.5×10^{-5}

(images) into compact hash codes and test time is the time for hashing a query. We do not count in the time for building a hash table as well as the time for hash lookup since both can be ignored compared to the hashing time (i.e., data compression time) (see the last two paragraphs and Table 4.1 in Section 4.2 for elaborate illustration). We also report MAP for ℓ_2 linear scan in the original feature space and ℓ_2 linear scan in the spectral embedding (SE) space, namely **SE ℓ_2 linear scan** whose binary version is 1-AGH. From this table it is clear that SE ℓ_2 scan gives higher precision than ℓ_2 scan for $r \leq 24$. This shows that spectral embedding is capturing the semantic neighborhoods by learning the intrinsic manifold structure of the data. Increasing r leads to poorer MAP performance of SE ℓ_2

Table 4.5: Hamming ranking performance on **NUS-WIDE**. Mean Precision (MP) of top-5000 ranked neighbors, training time, and test time are evaluated for each hashing method. r denotes the number of hash bits used in hashing algorithms, and also the number of eigenfunctions used in SE ℓ_2 linear scan. The K-means execution time is 105.5 seconds for training AGH on **NUS-WIDE**. All training and test time is recorded in second. At a fixed bit number, two highest MP values achieved by hashing are displayed in boldface type.

Method	**NUS-WIDE (270K)**				
	MP/5000			**Train Time**	**Test Time**
	$r = 12$	$r = 24$	$r = 48$	$r = 48$	$r = 48$
ℓ_2 Scan	0.4523			–	
SE ℓ_2 Scan	0.4842	0.4866	0.4775	–	–
LSH	0.3062	0.3196	0.2844	8.5	1.0×10^{-5}
PCAH	0.3906	0.3643	0.3450	18.8	1.3×10^{-5}
USPLH	0.4212	0.4269	0.4322	834.7	1.3×10^{-5}
ITQ	**0.4615**	0.4669	0.4728	37.9	1.3×10^{-5}
SH	0.3846	0.3609	0.3420	25.1	4.1×10^{-5}
KLSH	0.3420	0.4232	0.4157	8.7	4.9×10^{-5}
SIKH	0.2914	0.3270	0.3094	2.0	1.1×10^{-5}
1-AGH	**0.4673**	**0.4762**	**0.4761**	115.2	4.4×10^{-5}
2-AGH	0.4525	**0.4699**	**0.4779**	118.1	5.3×10^{-5}

scan, indicating the intrinsic manifold dimension to be around 24. Further, the table shows that even 1-AGH performs better than all other hashing methods except USPLH which may be better than 1-AGH with more bits. 2-AGH performs significantly better than the other hashing methods and ℓ_2 scan, and even better than SE ℓ_2 scan when $r \geq 24$. Note that the results from both ℓ_2 and SE ℓ_2 linear scans are provided to show the advantage of taking the manifold view in AGH. Such linear scans are not fast NN search methods as they are very expensive to compute in comparison to any other hashing methods.

In terms of training time, while 1-AGH and 2-AGH need more time than the most hashing methods, they are faster than USPLH. Most of the training time in AGH is spent

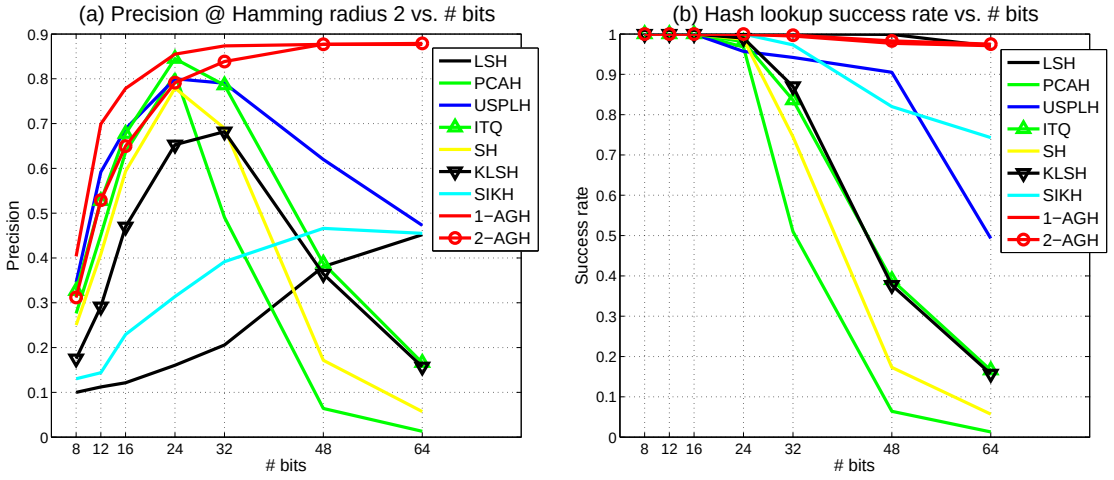


Figure 4.7: Hash lookup results on **MNIST**. (a) Mean precision of Hamming radius 2 hash lookup using the varying number of hash bits (r); (b) hash lookup success rate using the varying number of hash bits (r).

on the K-means step. By using a subsampled dataset, instead of the whole database, one can further speed up K-means significantly. The test time of AGH methods is comparable to the other nonlinear hashing methods SH and KLSH. Table 4.5 shows a similar trend on the **NUS-WIDE** dataset. As computing MAP is slow on this larger dataset, we show Mean Precision (MP) of top-5000 returned neighbors for all these compared hashing methods along with two exhaustive searches ℓ_2 scan and SE ℓ_2 scan. This time 1-AGH outperforms all of the other hashing methods in terms of MP when $r \leq 24$, and 2-AGH outperforms all

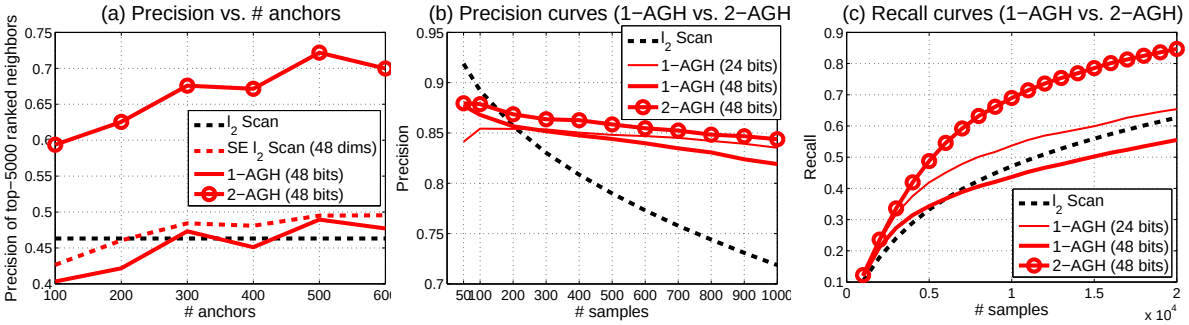


Figure 4.8: Hamming ranking results on **MNIST**. (a) Mean precision of top-5000 ranked neighbors using the varying number of anchors (m); (b) mean precision curves of Hamming ranking; (c) mean recall curves of Hamming ranking.

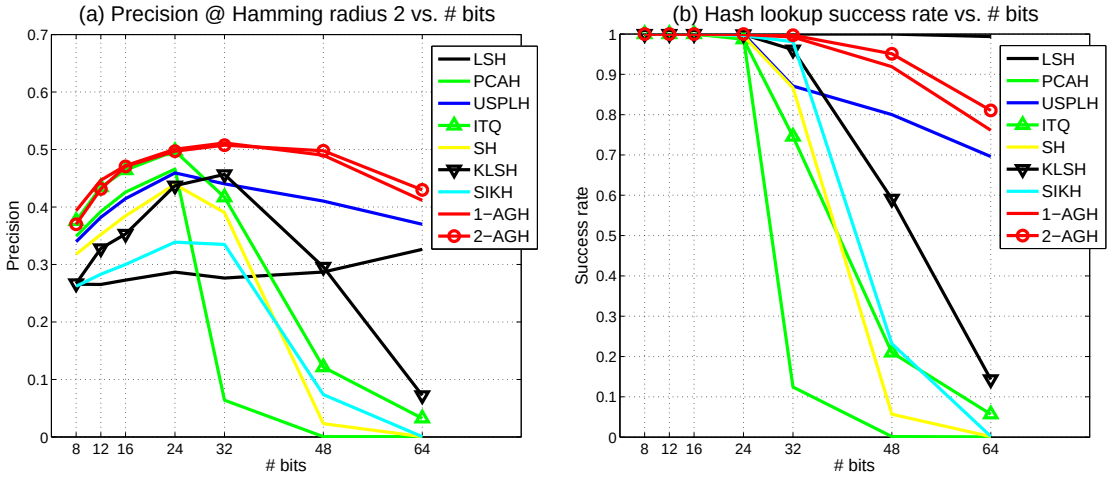


Figure 4.9: Hash lookup results on **NUS-WIDE**. (a) Mean precision of Hamming radius 2 hash lookup using the varying number of hash bits (r); (b) hash lookup success rate using the varying number of hash bits (r).

of the other hashing methods and even SE ℓ_2 scan when $r = 48$.

Figures 4.7(a) and 4.9(a) show the mean precision curves using hash lookup within Hamming radius 2. Note that we follow [186] to treat failing to find any hash bucket for a query as zero precision, different from [191] which ignored the failed queries in computing the mean hash lookup precision over all queries. Due to increased sparsity of the Hamming space with more bits, precision for the most hashing methods drops significantly when longer codes are used. However, both 1-AGH and 2-AGH do not suffer from this common drawback

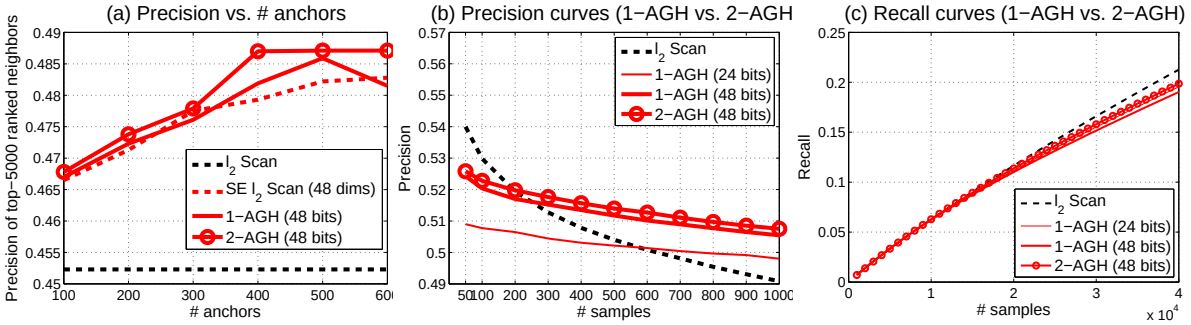


Figure 4.10: Hamming ranking results on **NUS-WIDE**. (a) Mean precision of top-5000 ranked neighbors using the varying number of anchors (m); (b) mean precision curves of Hamming ranking; (c) mean recall curves of Hamming ranking.

and provide stably higher precision when using more than 24 bits for both datasets. To deeply investigate the issue encountered in hash lookups using longer codes, we report the hash lookup success rate: the proportion of the queries resulting in successful hash lookups in all queries. This success rate quantifies the failing cases in hash lookups. Figures 4.7(b) and 4.9(b) clearly indicate that LSH, 2-AGH and 1-AGH enjoy the highest hash lookup success rates. When $r \geq 24$, the success rates of PCAH, ITQ, SH and KLSH drop sharply, while those of LSH, 1-AGH and 2-AGH keep high values which are always larger than 0.97 on **MNIST** and 0.76 on **NUS-WIDE**, respectively. The reported hash lookup success rates explain why some hashing methods suffer from very low Hamming radius 2 precision when using longer codes, and why our proposed hashing approaches 1-AGH and 2-AGH remain high precision. From another perspective, the high success rates that our approaches accomplish also reveal that our approaches maintain tight neighborhoods (e.g., Hamming distance 2), irrespective of the code length, in the produced code spaces.

We also plot the Hamming ranking precision of top-5000 returned neighbors with an increasing number of anchors ($100 \leq m \leq 600$) in Figures 4.8(a) and 4.10(a) (except these two, all of the results are reported under $m = 300$), from which one can observe that 2-AGH consistently provides superior precision performance compared to ℓ_2 linear scan, SE ℓ_2 linear scan, and 1-AGH. The gains are more significant on **MNIST**.

Finally, overall better performance of 2-AGH over 1-AGH implies that the higher (less smooth) eigenfunctions of Anchor Graph Laplacians are not as good as the lower (smoother) ones when used to create hash bits. 2-AGH reuses the lower eigenfunctions and gives higher search accuracy including both precision and recall, as shown in Figures 4.8(b)(c) and 4.10(b)(c). Note that though 1-AGH achieves comparable hash lookup and Hamming ranking precision as 2-AGH, it has lower Hamming ranking recall than 2-AGH with the same number of hash bits.

4.6 Summary and Discussion

In this chapter, we have developed a realizable graph-based unsupervised hashing approach which respects the underlying manifold structure of the input data to capture semantic

neighborhoods on manifolds and return meaningful nearest neighbors for a query. We further showed that Anchor Graphs can overcome the computationally prohibitive steps of building and manipulating large graph Laplacians by approximating graph adjacency matrices with low-rank matrices. As such, the hash functions can be efficiently obtained by thresholding the lower (smoother) eigenfunctions of the Anchor Graph Laplacian in a hierarchical fashion. Experimental comparison showed that our proposed approach Anchor Graph Hashing (AGH) enjoys significant performance gains over the state-of-the-art hashing methods in retrieving semantically similar neighbors. In the future, we would like to investigate if any theoretical guarantees could be provided on retrieval accuracy of AGH.

In addition, we make several important observations about the proposed AGH method below.

1. The idea of AGH is centered around an assumption that manifolds exist or nearly exist under the input data collection in a high-dimensional feature space. By making such an assumption, we are then able to leverage Anchor Graphs to discover the manifold structure of the data, and also conduct hashing to capture the manifold structure in the Hamming code space. Nevertheless, if manifolds do not exist due to diverse data distributions or poor feature extraction schemes, it remains an open question whether AGH can still be successful. In the next chapter, we will try to answer this question and propose a more general hashing approach which will not rely on any assumptions about the data but need the supervised information to explicitly capture the semantic relationships among the data into hashing.

2. Hierarchical hashing, that we have proposed in subsection 4.4.5, is a novel idea and has not been mentioned and explored in the literature. The devised hierarchical thresholding procedure perfectly works in conjunction with Anchor Graphs, providing data-adaptive hash functions (functions at the second layer are conditioned on the outputs of functions at the first layer). This hierarchical idea is not only suitable for Anchor Graph-based hashing, but also applicable to other hashing algorithms as an improvement. Possibly, one needs to make some proper modifications to our proposed hierarchical thresholding procedure. The recent work [89] follows the proposed hierarchical hashing idea to successfully extend several conventional one-layer hashing algorithms like LSH, SH and PCAH to two layers.

Chapter 5

Supervised Hashing

Recent years have witnessed the growing popularity of hashing, typically unsupervised hashing, in large-scale machine learning, computer vision, and information retrieval problems. Under the compact hashing framework that we presented in Chapter 4, there has been some recent research which shows that the hashing performance could be boosted by leveraging supervised information into the hash function learning process. However, the existing supervised hashing methods either lack adequate performance or often incur cumbersome model training.

In this chapter, we present a novel kernel-based supervised hashing model [116] which requires a limited amount of supervised information, i.e., similar and dissimilar data pairs, and a feasible training cost in achieving high quality hashing. The idea is to map the data to compact binary codes whose Hamming distances are minimized between points of similar pairs and simultaneously maximized between points of dissimilar pairs. Our approach is distinct from prior work in utilizing the equivalence between optimizing the code inner products and the Hamming distances. This enables us to sequentially and efficiently train the hash functions one bit at a time, yielding very short yet discriminative codes whose code inner products are optimized explicitly and at the same time whose Hamming distances are optimized in an implicit manner. We carry out extensive experiments on two image benchmarks of up to one million samples, demonstrating that our approach significantly outperforms the state-of-the-arts in searching both semantically similar neighbors and metric distance neighbors, with accuracy gains ranging from 13% to 46% over the alternative

methods.

In the rest of this chapter, we state the problem background of supervised hashing in Section 5.1, review the related work in Section 5.2, introduce the notations in Section 5.3, present our approach *Kernel-Based Supervised Hashing* (KSH) [116] in Section 5.4, show the experimental results in Section 5.5, and lastly give our summary and discussion in Section 5.6.

5.1 Problem Background

During the past few years, hashing has become a popular solution to tackle a large spectrum of large-scale computer vision problems including nonparametric (e.g., k-NN based) object recognition [174][175], fast image retrieval [95][186], fast image and video duplicate detection [32][77], fast pose estimation [152], fast image matching [90][163], compact local descriptor representation [129][77][79][78], etc. In these problems, hashing is exploited to expedite similarity computation and search. Since the encoding of high-dimensional image feature vectors to compact binary codes as proposed in [175], compact hashing has enabled significant efficiency gains in both storage and speed, as validated in Chapter 4. In many cases, hashing with only several tens of bits per image allows search into a collection of millions of images in a constant time [175][186].

In Chapter 4, we have mentioned that unsupervised hashing has some limitations such as low search accuracy (e.g., LSH, PCAH and SIKH) and dependence on somewhat strong assumptions about data (e.g., SH). Even our proposed AGH relies on the manifold assumption to derive smooth graph Laplacian eigenfunctions and then yield good hash codes. As we mentioned in Section 4.6, if manifolds do not exist under data due to diverse data distributions or poor feature sensing and extracting schemes, AGH cannot guarantee to achieve good performance. What's more, AGH is more suitable for retrieving semantically similar neighbors and was not designed for seeking metric neighbors. Therefore, a more general hashing approach that does not rely on any assumption about the data and can accommodate various neighbor definitions is needed.

In this chapter, we study novel hashing methods by taking advantage of supervised

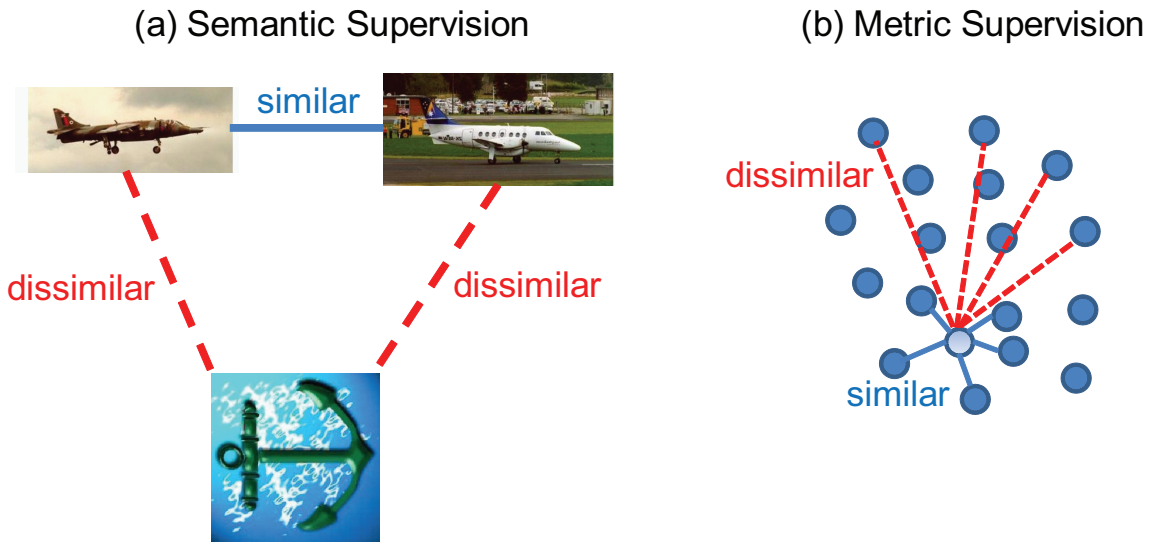


Figure 5.1: Two types of supervised information. A blue solid line denotes a similar relation, and a red dashed line denotes a dissimilar relation. (a) Semantic supervision: 'similar' represents that two images belong to the same object category, and 'dissimilar' represents that two images come from different categories. (b) Metric supervision: 'similar' represents that two samples hold a sufficiently short metric distance, and 'dissimilar' represents that two samples incur a sufficiently long metric distance.

information. In the literature of supervised hashing [145][93][137][186], the supervised information is often given in terms of pairwise labels: 1 labels specify *similar* (i.e., neighbor) pairs, and -1 labels designate *dissimilar* (i.e., nonneighbor) pairs. Such pairs may be acquired from neighborhood structures in a predefined metric (e.g., ℓ_2) space, or from semantic relevancy when semantic-level labels of some samples are available via meta-data or manual annotations. Hence, the supervised information covers definitions of semantic and metric neighborhoods, which is schematically displayed in Figure 5.1 in which the image examples are from **Caltech-101** [47]. The purpose of supervised hashing is to explicitly preserve the given supervised information during hash code generation. Figure 5.2 draws a desirable hash function $h(\boldsymbol{x})$ meeting such an objective. Such a function generates the same hash bit $(h(\boldsymbol{x}_i) = h(\boldsymbol{x}_j))$ for a similar data pair $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ whereas different hash bits $(h(\boldsymbol{x}_i) \neq h(\boldsymbol{x}_j))$ for a dissimilar data pair.

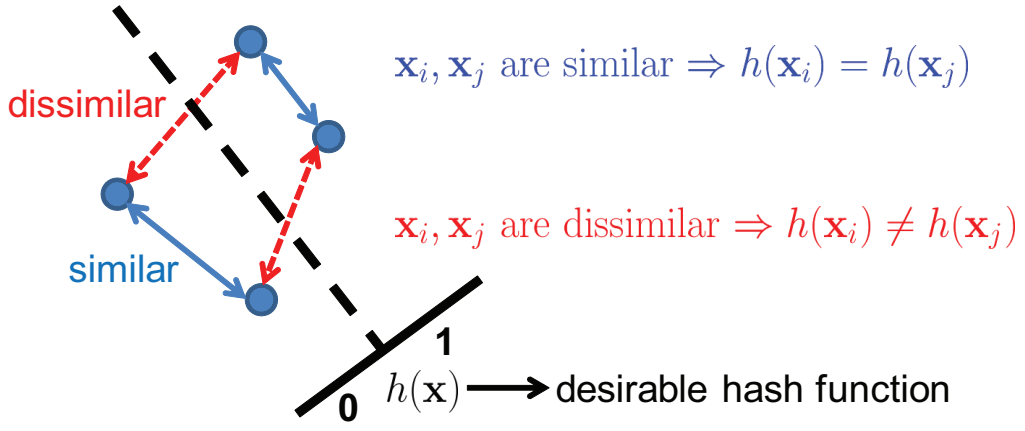


Figure 5.2: A desirable hash function subject to supervised information.

5.2 Related Work

The early exploration of hashing focuses on using random projections to construct randomized hash functions. One of the well-known representatives is Locality-Sensitive Hashing (LSH) [52][4] which has been continuously expanded to accommodate more distance/similarity metrics including ℓ_p distances for $p \in (0, 2]$ [39], the Mahalanobis distance [95], the cosine similarity [27], and the kernel similarity [94]. Because of theoretical guarantees that original metrics are asymptotically preserved in the Hamming (code) space with increasing code length, LSH-related methods usually require long codes to achieve good precision. Nonetheless, long codes result in low recall since the collision probability that two codes fall into the same hash bucket decreases exponentially as the code length increases. Thus, multiple hash tables are needed to maintain reasonable recall, leading to longer query time and tremendous increase in storage.

As discussed in Chapter 4, in contrast to the data-independent hash schemes employed in LSH-related methods, recent endeavors aim at *data-dependent hashing* which generates a compact set of hash bits via learning meaningful hash functions. Through encoding high-dimensional data points into compact codes, nearest neighbor search can be accomplished with a constant time complexity as long as the neighborhood of a point is well preserved in the code space. In addition, compact codes are particularly useful for saving storage in gigantic databases. Even for linear scans through entire databases compact codes still

allow fast search. As shown in our experiments presented in Section 5.5, searching hundreds of neighbors of a query in one million samples consumes about 0.005 second using 48 binary bits per sample. To design effective compact hashing, a number of methods such as projection learning for hashing [186][56][163], Spectral Hashing (SH) [191], Anchor Graph Hashing (AGH) [117] (as described in Chapter 4), Optimized Kernel Hashing [60], Semi-Supervised or Weakly-Supervised Hashing [186][128], Restricted Boltzmann Machines (RBMs) (or semantic hashing) [145][146], Binary Reconstruction Embeddings (BRE) [93], and Minimal Loss Hashing (MLH) [137] have been proposed. We summarize them into three main categories, *unsupervised*, *semi-supervised*, and *supervised* methods.

For unsupervised hashing, principled linear projections like PCA Hashing (PCAH) [186] and its rotational variant Iterative Quantization (ITQ) [56] were suggested for better quantization rather than random projections. Nevertheless, only a few orthogonal projections are good for quantization as the variances of data usually decay rapidly as pointed out by [186]. An alternative solution is to seek nonlinear data representation from the low-energy spectrums of data neighborhood graphs [191][117]. Exactly solving eigenvectors or eigenfunctions of large-scale graphs is computationally prohibitive. In response, [191][117] proposed several approximate solutions by adopting restrictive assumptions about the data distribution or the neighborhood structure.

While unsupervised hashing is promising to retrieve metric distance neighbors, e.g., ℓ_2 neighbors, a variety of practical applications including image search prefer semantically similar neighbors [175]. Therefore, the recent work incorporated supervised information to boost the hashing performance. Such supervised information is customarily expressed as pairwise labels of similar and dissimilar data pairs in availability. One representative work is Semi-Supervised Hashing (SSH) [186] which minimized the empirical error on the labeled data while maximizing entropy of the generated hash bits over the unlabeled data. Another work, namely Weakly-Supervised Hashing [128], handled higher-order supervised information.

Importantly, we argue that supervised hashing could attain higher search accuracy than unsupervised and semi-supervised hashing if the supervised information were thoroughly exploited. Though the simple supervised method Linear Discriminant Analysis Hashing

(LDAH) [163] can tackle supervision via easy optimization, it lacks adequate performance because of the use of orthogonal projections in hash functions. Optimized Kernel Hashing (OKH) [60] resembles the themes of SSH and LDAH in handling supervised information, but pursued the orthogonal projections in a kernel space. There exist more sophisticated supervised methods such as RBM [145], BRE [93] and MLH [137] that have shown higher search accuracy than unsupervised hashing approaches, but they all impose difficult optimization and slow training mechanisms. This inefficiency issue has greatly diminished the applicability of the existing supervised hashing methods to large-scale tasks. We discover that the expensive training costs are caused by the overcomplicated hashing objective functions used in the prior work. To this end, high-quality supervised hashing with a low training cost is fundamentally important, yet still unavailable to the best of our knowledge.

In this chapter, we show that the supervised information can be incorporated in a more effective and efficient manner. Specifically, we propose a novel and elegant objective function for learning the hash functions. The prior supervised methods [93][137] both tried to control the Hamming distances in the code space such that they correlate well with the given supervised information. Unfortunately, it is very difficult to directly optimize Hamming distances that are nonconvex and nonsmooth [93]. By utilizing the algebraic equivalence between a Hamming distance and a code inner product, the proposed objective function dexterously manipulates code inner products, leading to implicit yet more effective optimization on Hamming distances. We also exploit the separable property of code inner products to design an efficient greedy algorithm which sequentially solves the target hash functions bit by bit. To accommodate linearly inseparable data, we employ a kernel formulation for the target hash functions, so we name the presented approach *Kernel-Based Supervised Hashing* (KSH) which is able to deal with both semantic and metric supervision. We evaluate the performance of KSH in searching both semantically similar neighbors and metric distance neighbors on two large image benchmarks of up to one million samples, confirming its dramatically higher accuracy compared with the state-of-the-art unsupervised, semi-supervised, and supervised hashing methods.

There are other hashing methods [108][61] which stress the necessity of independence among hash bits based on the principle similar to that used in entropy-based hashing [141].

Specifically, they attempted to minimize the Hamming distances between neighbor pairs subject to the constraint that each bit of the learned hash functions should have maximum conditional entropy with respect to any of the other bits. Nevertheless, there is no convincing evidence that independent hash bits will result in higher NN search accuracy. Actually, in the supervised scenario, interdependent hash bits may be advantageous, which is corroborated by the recent work Minimal Loss Hashing (MLH) [137] where hash bits were treated as interdependent latent variables.

Following the practice in Chapter 4, in this chapter we will still focus on the compact hashing mode, in which only 8 to 48 bits will be used in all performance comparison in addition to the use of Hamming radius 2 hash table lookup. It is worthwhile to point out that hash lookup within a small Hamming radius is equivalent to multi-probe hashing [120][42] which constructs a few new queries around the original query.

Finally, to summarize various hashing methods we have discussed so far, we list the basic properties of the representative hash functions in Table 5.1.

5.3 Notations

In this section, we define the notations and symbols that we will use in the rest of this chapter. All notations as well as their definitions are listed in Tables 5.2 and 5.3.

5.4 Kernel-Based Supervised Hashing (KSH)

5.4.1 Hash Functions with Kernels

As stated in Chapter 4, the purpose of hashing is to look for a group of appropriate hash functions $\{h : \mathbb{R}^d \mapsto \{1, -1\}\}$, each of which accounts for the generation of a single hash bit, given a data set $\mathcal{X} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n\} \subset \mathbb{R}^d$. In the same way as Chapter 4, we treat '0' bit as '-1' bit for formulation and training, and use '0' bit back in the implementations of data coding and hashing. Different from AGH that we have proposed in Chapter 4, we use a kernel function $\kappa : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ to construct the hash functions in demand. This is because the kernel trick has been theoretically and empirically proven to be able to tackle

Table 5.1: The basic properties of various hashing methods. $\boldsymbol{x}$ denotes a data vector, $\boldsymbol{k}(\boldsymbol{x})$ denotes a kernel mapping, $\boldsymbol{z}(\boldsymbol{x})$ denotes the data-to-anchor mapping defined in Chapter 4, $\boldsymbol{w}$ denotes a projection vector, k denotes an integer, and b, t denote shift scalars.

Method	**Hash Function**	$\boldsymbol{w}$	**Learning Style**
LSH [4]	$\text{sgn}(\boldsymbol{w}^\top \boldsymbol{x} + b)$	randomized	no learning
PCAH [186]	$\text{sgn}(\boldsymbol{w}^\top \boldsymbol{x} + b)$	learned	unsupervised
ITQ [56]	$\text{sgn}(\boldsymbol{w}^\top \boldsymbol{x} + b)$	learned	unsupervised
SSH [186]	$\text{sgn}(\boldsymbol{w}^\top \boldsymbol{x} + b)$	learned	semi-supervised
LDAH [163]	$\text{sgn}(\boldsymbol{w}^\top \boldsymbol{x} + b)$	learned	supervised
KLSH [94]	$\text{sgn}\left(\boldsymbol{w}^\top \boldsymbol{k}(\boldsymbol{x})\right)$	randomized	no learning
OKH [60]	$\text{sgn}\left(\boldsymbol{w}^\top \boldsymbol{k}(\boldsymbol{x})\right)$	learned	unsupervised or supervised
SIKH [142]	$\text{sgn}\left(\cos(\boldsymbol{w}^\top \boldsymbol{x} + b) + t\right)$	randomized	no learning
SH [191]	$\text{sgn}\left(\sin(\frac{\pi}{2} + k\pi(\boldsymbol{w}^\top \boldsymbol{x} + b))\right)$	learned	unsupervised
AGH (Chapter 4)	$\text{sgn}\left(\boldsymbol{w}^\top \boldsymbol{z}(\boldsymbol{x})\right)$	learned	unsupervised
BRE [93]	$\text{sgn}\left(\boldsymbol{w}^\top \boldsymbol{k}(\boldsymbol{x})\right)$	learned	unsupervised or supervised
MLH [137]	$\text{sgn}(\boldsymbol{w}^\top \boldsymbol{x} + b)$	learned	supervised
KSH (this chapter)	$\text{sgn}\left(\boldsymbol{w}^\top \boldsymbol{k}(\boldsymbol{x})\right)$	learned	supervised

practical data that are mostly linearly inseparable [149].

Following the Kernelized Locality-Sensitive Hashing (KLSH) [94] algorithm, we first define a prediction function $f : \mathbb{R}^d \mapsto \mathbb{R}$ with the kernel κ plugged in as follows

$$f(\boldsymbol{x}) = \sum_{j=1}^{m} \kappa(\boldsymbol{x}_{(j)}, \boldsymbol{x}) w_j - b, \tag{5.1}$$

where $\boldsymbol{x}_{(1)}, \cdots, \boldsymbol{x}_{(m)}$ are m samples uniformly selected at random from $\mathcal{X}$, which behave as anchor points as what have been used in AGH. $w_j \in \mathbb{R}$ is the jth coefficient, and $b \in \mathbb{R}$ is the bias. Based on f, the hash function for a single hash bit is constructed by

$$h(\boldsymbol{x}) = \text{sgn}(f(\boldsymbol{x})), \tag{5.2}$$

in which the sign function $\text{sgn}(x)$ returns 1 if input variable $x > 0$ and returns -1 otherwise. Note that m is fixed to a constant much smaller than the data set size n in order to maintain fast hashing.

Table 5.2: Table of notations.

Notation	Definition
n	The number of database points
l	The number of labeled points
m	The number of anchor points
d	The dimension of database, anchor or query points
i, j	The indices of database or anchor points
$\boldsymbol{x}_i \in \mathbb{R}^d$	The ith database point
$\boldsymbol{x}_{(j)} \in \mathbb{R}^d$	The jth anchor point
$\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^n$	The data set
$\mathcal{X}_l = \{\boldsymbol{x}_i\}_{i=1}^l$	The labeled data set
$\boldsymbol{q} \in \mathbb{R}^d$	The query point
$h : \mathbb{R}^d \mapsto \{1, -1\}$	A hash function for a single bit
$\text{sgn}(x) \in \{1, -1\}$	The sign function that returns 1 for $x > 0$ and -1 otherwise
$\kappa : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$	A kernel function
$f : \mathbb{R}^d \mapsto \mathbb{R}$	A prediction function
$\bar{\boldsymbol{k}}(\boldsymbol{x}) \in \mathbb{R}^m$	The zero-centered kernel mapping
$\boldsymbol{w} \in \mathbb{R}^m$	A projection vector in a kernel feature space
$b \in \mathbb{R}$	The bias scalar
$\mu_j \in \mathbb{R}$	The jth mean value, $j \in [1 : m]$
r	The number of hash bits
k	The index of hash bits
$h_k : \mathbb{R}^d \mapsto \{1, -1\}$	The hash function for the kth bit
$\boldsymbol{w}_k \in \mathbb{R}^m$	The kth projection vector
$\mathcal{M} \subset \mathbb{R}^d \times \mathbb{R}^d$	The set of similar data pairs
$\mathcal{C} \subset \mathbb{R}^d \times \mathbb{R}^d$	The set of dissimilar data pairs
$S = (S_{ij})_{i,j} \in \mathbb{R}^{l \times l}$	The pairwise label matrix
$\mathcal{D}_{H_r} : \mathbb{R}^d \times \mathbb{R}^d \mapsto [0 : r]$	The Hamming distance function between r-bit binary codes
$H_r(\boldsymbol{x}) = [h_1(\boldsymbol{x}), \cdots, h_r(\boldsymbol{x})]$	The r-bit binary code of sample $\boldsymbol{x}$
$B_l \in \{1, -1\}^{l \times r}$	The code matrix of the labeled data $\mathcal{X}_l$
$\bar{K}_l \in \mathbb{R}^{l \times m}$	The kernel matrix between $\mathcal{X}_l$ and m anchor points

Table 5.3: Table of notations (continued).

$W = [\boldsymbol{w}_1, \cdots, \boldsymbol{w}_r] \in \mathbb{R}^{m \times r}$	The projection matrix in the kernel feature space
$\mathcal{Q} : \mathbb{R}^{m \times r} \mapsto \mathbb{R}$	The objective (cost) function for optimizing W
$R_{k-1} \in \mathbb{R}^{l \times l}$	The $(k-1)$th residue matrix, $k \in [1 : r]$
$g : \mathbb{R}^m \mapsto \mathbb{R}$	The objective (cost) function for optimizing $\boldsymbol{w}_k$
$\tilde{g} : \mathbb{R}^m \mapsto \mathbb{R}$	The smooth surrogate of g
$\varphi(x) \in \mathbb{R}$	The sigmoid-shaped function to approximate $\mathrm{sgn}(x)$

An important criterion guiding the design of hash functions is that the generated hash bit should take as much information as possible, which implies a balanced hash function that meets $\sum_{i=1}^{n} h(\boldsymbol{x}_i) = 0$ [191][186][56][117]. For our problem, this balancing criterion makes b be the median of $\left\{\sum_{j=1}^{m} \kappa(\boldsymbol{x}_{(j)}, \boldsymbol{x}_i) w_j\right\}_{i=1}^{n}$. As a fast alternative to the median, we adopt the mean

$$b = \sum_{i=1}^{n} \sum_{j=1}^{m} \kappa(\boldsymbol{x}_{(j)}, \boldsymbol{x}_i) w_j / n \tag{5.3}$$

like [186][56]. Through substituting b in eq. (5.1) with the mean value, we obtain

$$\begin{aligned} f(\boldsymbol{x}) &= \sum_{j=1}^{m} \left(\kappa(\boldsymbol{x}_{(j)}, \boldsymbol{x}) - \frac{1}{n} \sum_{i=1}^{n} \kappa(\boldsymbol{x}_{(j)}, \boldsymbol{x}_i) \right) w_j \\ &= \boldsymbol{w}^\top \bar{\boldsymbol{k}}(\boldsymbol{x}), \end{aligned} \tag{5.4}$$

where $\boldsymbol{w} = [w_1, \cdots, w_m]^\top$ and $\bar{\boldsymbol{k}} : \mathbb{R}^d \mapsto \mathbb{R}^m$ is a vectorial mapping defined by

$$\bar{\boldsymbol{k}}(\boldsymbol{x}) = [\kappa(\boldsymbol{x}_{(1)}, \boldsymbol{x}) - \mu_1, \cdots, \kappa(\boldsymbol{x}_{(m)}, \boldsymbol{x}) - \mu_m]^\top, \tag{5.5}$$

in which $\mu_j = \sum_{i=1}^{n} \kappa(\boldsymbol{x}_{(j)}, \boldsymbol{x}_i)/n$ can be precomputed. In KLSH, the coefficient vector $\boldsymbol{w}$ came as a random direction drawn from a Gaussian distribution. Since $\boldsymbol{w}$ completely determines a hash function $h(\boldsymbol{x})$, we seek to learn $\boldsymbol{w}$ by leveraging supervised information so that the resulted hash function is discriminative.

It is worthwhile to point out that $\boldsymbol{w}$'s role is a projection vector in the m-dimensional kernel feature space $\{\bar{\boldsymbol{k}}(\boldsymbol{x}) | \boldsymbol{x} \in \mathbb{R}^d\}$ induced by the m anchor points $\{\boldsymbol{x}_{(j)}\}_{j=1}^{m}$. We could obtain better anchor points than the randomly selected exemplars via running K-means clustering like Chapter 4, but we want to satisfy the most generic case that data are not

known to have vectorial representation but only known to have a kernel function defined between. Actually, in many vision problems, data such as images and videos do not have vectorial representation, and are possibly expressed in the form of sets of features on which K-means clustering is not trivial to implement.

5.4.2 Manipulating Code Inner Products

Suppose that r hash bits are needed. Accordingly, we have to find r projection vectors $\boldsymbol{w}_1, \cdots, \boldsymbol{w}_r$ to construct r hash functions $\left\{h_k(\boldsymbol{x}) = \mathrm{sgn}(\boldsymbol{w}_k^\top \bar{\boldsymbol{k}}(\boldsymbol{x}))\right\}_{k=1}^r$. In the customary setting for supervised hashing [145][93][137][186][116], the supervised information is given in terms of pairwise labels: 1 labels specify *similar* (or neighbor) pairs collected in set $\mathcal{M}$, and -1 labels designate *dissimilar* (or nonneighbor) pairs collected in set $\mathcal{C}$. Such pairs may be acquired from neighborhood structures in a predefined metric (e.g., ℓ_2) space, or from semantic relevancy when semantic-level labels of some samples are available. Without loss of generality, we assume that the first l ($m < l \ll n$) samples $\mathcal{X}_l = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_l\}$ are implicated in $\mathcal{M}$ and $\mathcal{C}$. To explicitly record the pairwise relationships among $\mathcal{X}_l$, we define a label matrix $S \in \mathbb{R}^{l \times l}$ as

$$S_{ij} = \begin{cases} 1, & (\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{M} \\ -1, & (\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{C} \\ 0, & \text{otherwise.} \end{cases} \tag{5.6}$$

Note that $S_{ii} \equiv 1$ since $(\boldsymbol{x}_i, \boldsymbol{x}_i) \in \mathcal{M}$. The intermediate label 0 implies that the similar/dissimilar relationship about some data pair is unknown or uncertain. The 0 labels mostly appear in the metric-based supervision (see Section 5.4.4).

Our purpose of supervised hashing is to generate discriminative hash codes such that similar pairs can be perfectly distinguished from dissimilar pairs by using Hamming distances in the code space. Specifically, we hope that the Hamming distances between the labeled pairs in $\mathcal{M} \cup \mathcal{C}$ correlate with the labels in S, that is, a pair with $S_{ij} = 1$ will have the minimal Hamming distance 0 while a pair with $S_{ij} = -1$ will take on the maximal Hamming distance, i.e., the number of hash bits r. Figure 5.3(b) illustrates our expectation for optimizing the Hamming distances.

However, directly optimizing the Hamming distances is nontrivial because of the complex

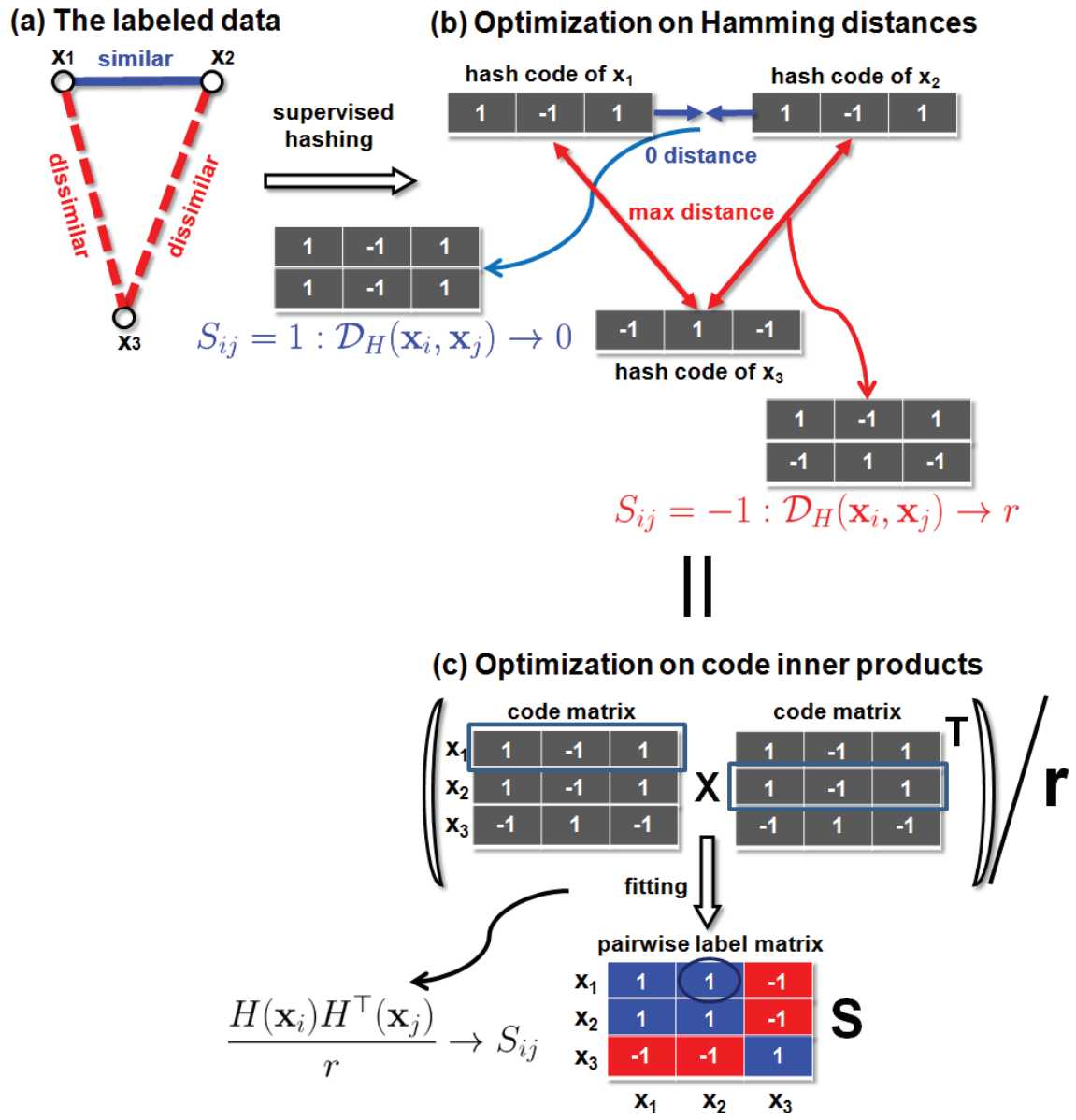


Figure 5.3: The core idea of our proposed supervised hashing. (a) Three data points with supervised pairwise labels: "1" (similar) and "-1" (dissimilar); (b) optimization on Hamming distances; (c) optimization on code inner products (r is the bit number). The latter two are equivalent due to the one-to-one correspondence between a Hamming distance and a code inner product.

mathematical formula $\mathcal{D}_{H_r}(\boldsymbol{x}_i, \boldsymbol{x}_j) = |\{k|h_k(\boldsymbol{x}_i) \neq h_k(\boldsymbol{x}_j), 1 \leq k \leq r\}|$. In this chapter, we advocate *code inner products* that are easier to manipulate and optimize, entailing a concise mathematical expression.

Code Inner Products vs. Hamming Distances

Let us write the r-bit hash code of sample $\boldsymbol{x}$ as $H_r(\boldsymbol{x}) = [h_1(\boldsymbol{x}), \cdots, h_r(\boldsymbol{x})] \in \{1, -1\}^{1\times r}$, and then deduce the code inner product as follows

$$\begin{aligned} & H_r(\boldsymbol{x}_i) \circ H_r(\boldsymbol{x}_j) \\ =& H_r(\boldsymbol{x}_i) H_r^\top(\boldsymbol{x}_j) \\ =& |\{k|h_k(\boldsymbol{x}_i) = h_k(\boldsymbol{x}_j), 1 \leq k \leq r\}| \\ & - |\{k|h_k(\boldsymbol{x}_i) \neq h_k(\boldsymbol{x}_j), 1 \leq k \leq r\}| \\ =& r - 2\,|\{k|h_k(\boldsymbol{x}_i) \neq h_k(\boldsymbol{x}_j), 1 \leq k \leq r\}| \\ =& r - 2\mathcal{D}_{H_r}(\boldsymbol{x}_i, \boldsymbol{x}_j), \end{aligned} \tag{5.7}$$

where the symbol $\circ$ stands for the code inner product. Critically, eq. (5.7) reveals that the Hamming distance and the code inner product is in one-to-one correspondence, hence enabling equivalent optimization on code inner products.

Given the observation of $H_r(\boldsymbol{x}_i) \circ H_r(\boldsymbol{x}_j) \in [-r, r]$ and $S_{ij} \in [-1, 1]$, we let $H_r(\boldsymbol{x}_i) \circ H_r(\boldsymbol{x}_j)/r$ fit S_{ij} as shown in Figure 5.3(c). This makes sense because $H_r(\boldsymbol{x}_i) \circ H_r(\boldsymbol{x}_j)/r = S_{ij} = 1$ leads to $\mathcal{D}_{H_r}(\boldsymbol{x}_i, \boldsymbol{x}_j) = 0$ and $H_r(\boldsymbol{x}_i) \circ H_r(\boldsymbol{x}_j)/r = S_{ij} = -1$ leads to $\mathcal{D}_{H_r}(\boldsymbol{x}_i, \boldsymbol{x}_j) = r$ by eq. (5.7). In a natural means, we propose a least-squares style objective function $\mathcal{Q}$ to learn the codes of the labeled data $\mathcal{X}_l$:

$$\min_{B_l \in \{1,-1\}^{l\times r}} \mathcal{Q} = \left\| \frac{1}{r} B_l B_l^\top - S \right\|_{\mathrm{F}}^2, \tag{5.8}$$

where $B_l = \begin{bmatrix} H_r(\boldsymbol{x}_1) \\ \cdots\cdots \\ H_r(\boldsymbol{x}_l) \end{bmatrix}$ denotes the code matrix of $\mathcal{X}_l$, and $\|.\|_{\mathrm{F}}$ represents the Frobenius norm.

We can generalize sgn() to take the elementwise sign operation for any vector or matrix

input, and then express the code matrix B_l as (given $h_k(\boldsymbol{x}) = \text{sgn}(\boldsymbol{w}_k^\top \bar{\boldsymbol{k}}(\boldsymbol{x})) = \text{sgn}(\bar{\boldsymbol{k}}^\top(\boldsymbol{x})\boldsymbol{w}_k)$)

$$B_l = \begin{bmatrix} h_1(\boldsymbol{x}_1), \cdots, h_r(\boldsymbol{x}_1) \\ \cdots\cdots \\ h_1(\boldsymbol{x}_l), \cdots, h_r(\boldsymbol{x}_l) \end{bmatrix} = \begin{bmatrix} \text{sgn}(\bar{\boldsymbol{k}}^\top(\boldsymbol{x}_1)\boldsymbol{w}_1), \cdots, \text{sgn}(\bar{\boldsymbol{k}}^\top(\boldsymbol{x}_1)\boldsymbol{w}_r) \\ \cdots\cdots \\ \text{sgn}(\bar{\boldsymbol{k}}^\top(\boldsymbol{x}_l)\boldsymbol{w}_1), \cdots, \text{sgn}(\bar{\boldsymbol{k}}^\top(\boldsymbol{x}_l)\boldsymbol{w}_r) \end{bmatrix} = \text{sgn}(\bar{K}_l W), \quad (5.9)$$

where $\bar{K}_l = [\bar{\boldsymbol{k}}(\boldsymbol{x}_1), \cdots, \bar{\boldsymbol{k}}(\boldsymbol{x}_l)]^\top \in \mathbb{R}^{l \times m}$ and $W = [\boldsymbol{w}_1, \cdots, \boldsymbol{w}_r] \in \mathbb{R}^{m \times r}$. After substituting B_l in eq. (5.8) with eq. (5.9), we obtain an analytical form of the objective function $\mathcal{Q}$ with respect to the projection matrix W:

$$\min_{W \in \mathbb{R}^{m \times r}} \mathcal{Q}(W) = \left\| \frac{1}{r} \text{sgn}(\bar{K}_l W)(\text{sgn}(\bar{K}_l W))^\top - S \right\|_{\text{F}}^2. \quad (5.10)$$

The novel objective function $\mathcal{Q}$ is simpler and more tractable than those of BRE [93] and MLH [137], because it offers a clearer connection and easier access to the model parameter W through manipulating code inner products.

In contrast, BRE and MLH optimize Hamming distances by pushing them close to raw metric distances or larger/smaller than appropriately chosen thresholds, either of which formulated a complicated objective function and incurred a tough optimization process, yet cannot guarantee the optimality of its solution. For direct comparison, we list the formulations of the objective functions used by BRE, MLH and KSH in Table 5.4. Importantly, our objective $\mathcal{Q}$ fulfills an intuitive notion that the high-level (semantic or metric) similarities revealed by the label matrix S are preserved into the low-level similarities, i.e., the (normalized) code inner products.

5.4.3 Greedy Optimization

The separable property of code inner products allows us to solve the hash functions in an incremental mode. With simple algebra, we rewrite $\mathcal{Q}$ in eq. (5.10) as

$$\min_{W} \quad \left\| \sum_{k=1}^{r} \text{sgn}(\bar{K}_l \boldsymbol{w}_k)(\text{sgn}(\bar{K}_l \boldsymbol{w}_k))^\top - rS \right\|_{\text{F}}^2, \quad (5.11)$$

where the r vectors $\boldsymbol{w}_k$'s, each of which determines a single hash function, are separated in the summation. This inspires a greedy idea for solving $\boldsymbol{w}_k$'s sequentially. At a time, it only involves solving one vector $\boldsymbol{w}_k$ provided with the previously solved vectors $\boldsymbol{w}_1^*, \cdots, \boldsymbol{w}_{k-1}^*$.

Table 5.4: The formulations of supervised hashing methods. $Loss_\rho(x, s) = \max(s(x-\rho)+1, 0)$ (ρ is a hyperparameter) is a hinge loss function used in MLH.

Method	Objective Function	Goal
BRE [93]	$\min_{H_r} \sum_{i,j=1}^{l} \left\| \frac{1}{r}\mathcal{D}_{H_r}(\boldsymbol{x}_i, \boldsymbol{x}_j) - \frac{1-S_{ij}}{2} \right\|^2$	$\mathcal{D}_{H_r}(\boldsymbol{x}_i, \boldsymbol{x}_j) \to 0$ for $S_{ij} = 1$ $\mathcal{D}_{H_r}(\boldsymbol{x}_i, \boldsymbol{x}_j) \to r$ for $S_{ij} = -1$
MLH [137]	$\min_{H_r} \sum_{i,j=1}^{l} Loss_\rho\left(\mathcal{D}_{H_r}(\boldsymbol{x}_i, \boldsymbol{x}_j), S_{ij}\right)$	$\mathcal{D}_{H_r}(\boldsymbol{x}_i, \boldsymbol{x}_j) \le \rho - 1$ for $S_{ij} = 1$ $\mathcal{D}_{H_r}(\boldsymbol{x}_i, \boldsymbol{x}_j) \ge \rho + 1$ for $S_{ij} = -1$
KSH (this chapter)	$\min_{H_r} \sum_{i,j=1}^{l} \left\| \frac{1}{r} H_r(\boldsymbol{x}_i) H_r^\top(\boldsymbol{x}_j) - S_{ij} \right\|^2$	$H_r(\boldsymbol{x}_i) H_r^\top(\boldsymbol{x}_j) \to r$ for $S_{ij} = 1$ $H_r(\boldsymbol{x}_i) H_r^\top(\boldsymbol{x}_j) \to -r$ for $S_{ij} = -1$

Let us define a residue matrix $R_{k-1} = rS - \sum_{t=1}^{k-1} \mathrm{sgn}(\bar{K}_l \boldsymbol{w}_t^*)(\mathrm{sgn}(\bar{K}_l \boldsymbol{w}_t^*))^\top$ ($R_0 = rS$). Then $\boldsymbol{w}_k$ can be pursued by minimizing the following cost

$$\begin{aligned}
&\left\| \mathrm{sgn}(\bar{K}_l \boldsymbol{w}_k)(\mathrm{sgn}(\bar{K}_l \boldsymbol{w}_k))^\top - R_{k-1} \right\|_{\mathrm{F}}^2 \\
=& \left((\mathrm{sgn}(\bar{K}_l \boldsymbol{w}_k))^\top \mathrm{sgn}(\bar{K}_l \boldsymbol{w}_k) \right)^2 \\
&- 2(\mathrm{sgn}(\bar{K}_l \boldsymbol{w}_k))^\top R_{k-1} \mathrm{sgn}(\bar{K}_l \boldsymbol{w}_k) + \mathrm{tr}(R_{k-1}^2) \\
=& -2(\mathrm{sgn}(\bar{K}_l \boldsymbol{w}_k))^\top R_{k-1} \mathrm{sgn}(\bar{K}_l \boldsymbol{w}_k) + l^2 + \mathrm{tr}(R_{k-1}^2) \\
=& -2(\mathrm{sgn}(\bar{K}_l \boldsymbol{w}_k))^\top R_{k-1} \mathrm{sgn}(\bar{K}_l \boldsymbol{w}_k) + const.
\end{aligned} \tag{5.12}$$

Discarding the constant term, we arrive at a cleaner cost

$$g(\boldsymbol{w}_k) = -(\mathrm{sgn}(\bar{K}_l \boldsymbol{w}_k))^\top R_{k-1} \mathrm{sgn}(\bar{K}_l \boldsymbol{w}_k). \tag{5.13}$$

A nice feature is that $g(\boldsymbol{w}_k)$ is lower-bounded as eq. (5.12) is always nonnegative. However, minimizing g is not easy to achieve because it is neither convex nor smooth. In what follows, we develop two optimization methods to approximately minimize g.

Spectral Relaxation. Motivated by the spectral methods for hashing [191][117], we apply the spectral relaxation trick to drop the sign functions involved in g, resulting in a constrained quadratic problem

$$\begin{aligned}
\max_{\boldsymbol{w}_k} \quad & (\bar{K}_l \boldsymbol{w}_k)^\top R_{k-1} (\bar{K}_l \boldsymbol{w}_k) \\
\text{s.t.} \quad & (\bar{K}_l \boldsymbol{w}_k)^\top (\bar{K}_l \boldsymbol{w}_k) = l
\end{aligned} \tag{5.14}$$

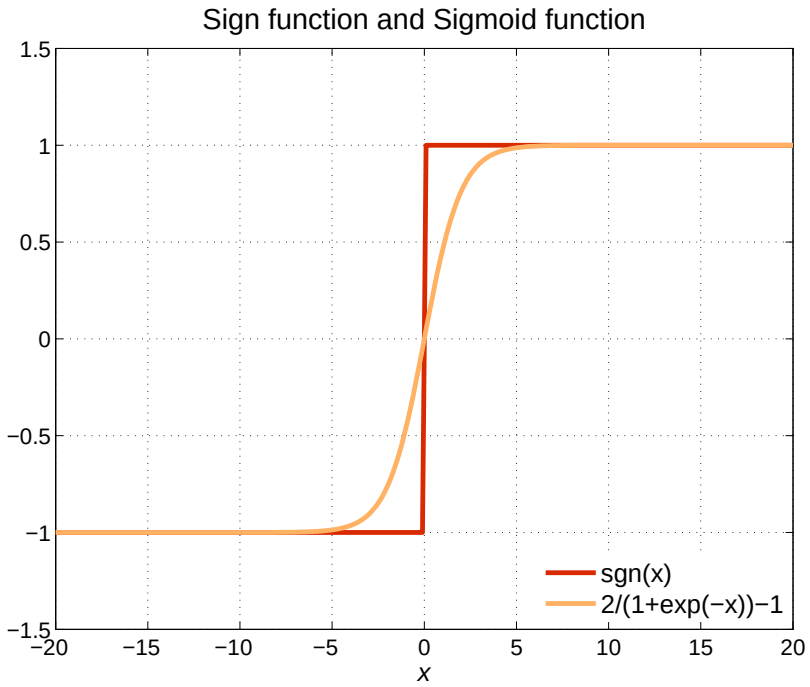


Figure 5.4: The sign function $\text{sgn}(x)$ and the sigmoid-shaped function $\varphi(x) = 2/(1 + \exp(-x)) - 1$.

where the constraint $(\bar{K}_l \boldsymbol{w}_k)^\top (\bar{K}_l \boldsymbol{w}_k) = l$ makes l elements in the vector $\bar{K}_l \boldsymbol{w}_k$ fall into the range of $[-1, 1]$ roughly, so that the solution of the relaxed problem eq. (5.14) is in the similar range to the original problem eq. (5.13). Eq. (5.14) is a standard generalized eigenvalue problem $\bar{K}_l^\top R_{k-1} \bar{K}_l \boldsymbol{w} = \lambda \bar{K}_l^\top \bar{K}_l \boldsymbol{w}$, and $\boldsymbol{w}_k$ is thus sought as the eigenvector associated with the largest eigenvalue. A proper scaling is conducted on the solved eigenvector, saved as $\boldsymbol{w}_k^0$, to satisfy the constraint in eq. (5.14).

Although spectral relaxation results in fast optimization, it might deviate far away from the optimal solution under larger l (e.g., $\geq$ 5,000) due to the amplified relaxation error (see Section 5.5.2). It is therefore used as the initialization to a more principled optimization scheme described below.

Sigmoid Smoothing. Since the hardness of minimizing g lies in the sign function, we replace sgn() in g with the sigmoid-shaped function $\varphi(x) = 2/(1 + \exp(-x)) - 1$ which is sufficiently smooth and well approximates $\text{sgn}(x)$ when $|x| > 6$, as shown in Figure 5.4.

Afterward, we propose to optimize the smooth surrogate $\tilde{g}$ of g:

$$\tilde{g}(\boldsymbol{w}_k) = -(\varphi(\bar{K}_l \boldsymbol{w}_k))^\top R_{k-1} \varphi(\bar{K}_l \boldsymbol{w}_k), \tag{5.15}$$

where $\varphi()$ operates elementwisely like sgn(). The gradient of $\tilde{g}$ with respect to $\boldsymbol{w}_k$ is derived as

$$\nabla \tilde{g} = -\bar{K}_l^\top \left((R_{k-1} \boldsymbol{b}_k) \odot (\mathbf{1} - \boldsymbol{b}_k \odot \boldsymbol{b}_k) \right), \tag{5.16}$$

where the symbol $\odot$ represents the Hadamard product (i.e., elementwise product), $\boldsymbol{b}_k = \varphi(\bar{K}_l \boldsymbol{w}_k) \in \mathbb{R}^l$, and $\mathbf{1}$ denotes a constant vector with l 1 entries.

Since the original cost g is lower-bounded, its smooth surrogate $\tilde{g}$ is lower-bounded as well. Consequently, we are capable of minimizing $\tilde{g}$ using the regular gradient descent technique [18]. Note that the smooth surrogate $\tilde{g}$ is still nonconvex, so it is unrealistic to look for a global minima of $\tilde{g}$. For fast convergence, we adopt the spectral relaxation solution $\boldsymbol{a}_k^0$ as a warm start and apply Nesterov's gradient method [134] to accelerate the gradient decent procedure. In most cases we can attain a locally optimal $\boldsymbol{w}_k^*$ at which $\tilde{g}(\boldsymbol{w}_k^*)$ is very close to its lower bound, which will be corroborated by the subsequent experiments in Section 5.5.

Finally, we describe the whole flowchart of the presented supervised hashing approach that we name *Kernel-Based Supervised Hashing* (KSH) in Algorithm 3. We also name another approach KSH^0 whose hash functions just use the initial spectral relaxation solutions $\{\boldsymbol{w}_k^0\}$. Empirically, we find that when l is not big the spectral relaxation solutions (i.e., initial solutions) are good enough. Thereby, within each loop of Algorithm 3, we check if the solutions found by gradient descent on the surrogate cost $\tilde{g}$ are surely better than the initial solutions in terms of minimizing the original cost g.

5.4.4 Analysis

Our approaches KSH^0 and KSH can both deal with semantic and metric supervision once the definitions about similar and dissimilar pairs are offered to learning. For example, a similar pair $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is confirmed if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ share at least one common semantic label or are nearest neighbors to each other under a predefined metric (e.g., ℓ_2); likewise, a dissimilar

Algorithm 3 Kernel-Based Supervised Hashing (KSH)

Input: a training sample set $\mathcal{X} = \{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^n$, a pairwise label matrix $S \in \mathbb{R}^{l \times l}$ defined on l samples $\mathcal{X}_l = \{\boldsymbol{x}_i\}_{i=1}^l$, a kernel function $\kappa : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$, the number of anchor points m ($< l$), and the number of hash bits r.

Preprocessing: uniformly randomly select m samples from $\mathcal{X}$, and compute zero-centered m-dim kernel vectors $\bar{\boldsymbol{k}}(\boldsymbol{x}_i)$ ($i = 1, \cdots, n$) using the kernel function κ according to eq. (5.5).

Greedy Optimization:

initialize $R_0 = rS$ and $T_{max} = 500$;

for $k = 1, \cdots, r$ **do**

solve the generalized eigenvalue problem

$\bar{K}_l^\top R_{k-1} \bar{K}_l \boldsymbol{w} = \lambda \bar{K}_l^\top \bar{K}_l \boldsymbol{w}$,

obtaining the largest eigenvector $\boldsymbol{w}_k^0$ such that $(\boldsymbol{w}_k^0)^\top \bar{K}_l^\top \bar{K}_l \boldsymbol{w}_k^0 = l$;

use the gradient descent method to optimize

$\min_{\boldsymbol{w}} -(\varphi(\bar{K}_l \boldsymbol{w}))^\top R_{k-1} \varphi(\bar{K}_l \boldsymbol{w})$ with the initial solution $\boldsymbol{w}_k^0$ and T_{max} budget iterations, achieving $\boldsymbol{w}_k^*$;

$\boldsymbol{h}^0 \longleftarrow \mathrm{sgn}(\bar{K}_l \boldsymbol{w}_k^0)$, $\boldsymbol{h}^* \longleftarrow \mathrm{sgn}(\bar{K}_l \boldsymbol{w}_k^*)$;

if $(\boldsymbol{h}^0)^\top R_{k-1} \boldsymbol{h}^0 > (\boldsymbol{h}^*)^\top R_{k-1} \boldsymbol{h}^*$ **then**

$\boldsymbol{w}_k^* \longleftarrow \boldsymbol{w}_k^0$, $\boldsymbol{h}^* \longleftarrow \boldsymbol{h}^0$;

end if

$R_k \longleftarrow R_{k-1} - \boldsymbol{h}^* (\boldsymbol{h}^*)^\top$;

end for

Coding: for $i = 1, \cdots, n$, do

$H_r(\boldsymbol{x}_i) \longleftarrow \left[\mathrm{sgn}(\bar{\boldsymbol{k}}^\top(\boldsymbol{x}_i) \boldsymbol{w}_1^*), \cdots, \mathrm{sgn}(\bar{\boldsymbol{k}}^\top(\boldsymbol{x}_i) \boldsymbol{w}_r^*)\right]$.

Output: r hash functions $\{h_k(\boldsymbol{x}) = \mathrm{sgn}(\bar{\boldsymbol{k}}^\top(\boldsymbol{x}) \boldsymbol{w}_k^*)\}_{k=1}^r$ as well as n hash codes $\{H_r(\boldsymbol{x}_i)\}_{i=1}^n$.

pair $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is determined if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ take different labels or are far away in the metric space.

In the semantic case, one can easily achieve the full entries (either 1 or -1) of the label matrix S since the implicated samples are all known to have semantic labels. But in the metric case, one needs to pre-compute two distance thresholds, one for similar pairs and the other for dissimilar pairs, to judge if two samples are metric neighbors or not by comparing their distance with the thresholds [191][93][186]. For the metric supervision, most entries in the label matrix S have 0 labels, which reveals that most distances fall into the middle ground between the two thresholds. To reduce the potential false alarms, our approaches implicitly push the Hamming distances of these 0-labeled pairs to $r/2$ as their code inner products have been pushed to zeros (see eq. (5.7)), which is reasonable since such pairs are not nearest neighbors in the metric space.

The time complexities for training KSH^0 and KSH are both bounded by $O((nm+l^2m+m^2l+m^3)r)$ which scales linearly with n given $n \gg l > m$. In practice, training KSH^0 is very fast and training KSH is about two times faster than two competing supervised hashing methods BRE and MLH. For each query, the hashing time of both KSH^0 and KSH is constant $O(dm + mr)$.

As far as the relationship among hash bits is concerned, all of BRE, MLH, and our proposed KSH^0 and KSH yield correlated hash bits. MLH treats the needed hash bits as interdependent latent variables. In BRE and our two approaches, the generated hash bits are also interdependent because any new bit was sought so as to best reduce the reconstruction residue caused by the previous bits. The other methods [108][61] stressing the necessity of independence of hash bits minimized the Hamming distances between neighbor pairs with the constraint that each bit of the learned hash functions should have maximum conditional entropy with respect to any of the other bits. However, there is no convincing evidence that independent hash bits will result in higher NN search accuracy. The empirical results conveyed by Section 5.5 corroborate that interdependent hash bits are advantageous for the supervised hashing task.

5.5 Experiments

We run large-scale image retrieval experiments on two image benchmarks: **CIFAR-10** [91] and one million subset of the 80 million tiny image collection [174]. **CIFAR-10** is a labeled subset of the 80M tiny images, consisting of a total of 60K 32×32 color images from ten object categories each of which contains 6K samples. Every image in this dataset is assigned to a mutually exclusive class label and represented by a 512-dimensional GIST feature vector [138]. The second dataset that we call **Tiny-1M** was acquired from [186], which does not include any semantic label but has been partially annotated to similar and dissimilar data pairs according to the ℓ_2 distance. Each image in **Tiny-1M** is represented by a 384-dimensional GIST vector.

We evaluate the proposed KSH^0 and KSH, and compare them against ten state-of-the-art methods including six unsupervised methods LSH [4], PCAH [186], SH [191], KLSH [94], 1-AGH, and 2-AGH proposed in the previous chapter, one semi-supervised method SSH (the nonorthogonal version) [186], and three supervised methods LDAH [163], BRE [93], and MLH [137]. These methods cover both linear (LSH, PCAH, SSH, LDAH and MLH) and nonlinear (SH, KLSH, 1-AGH, 2-AGH and BRE) hash schemes. We used the publicly available codes of SH, KLSH, SSH, BRE and MLH. All our experiments are run on a workstation with a 2.53 GHz Intel Xeon CPU and 48GB RAM.

Since KLSH, KSH^0 and KSH refer to kernels, we feed them the same Gaussian RBF kernel $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\|\boldsymbol{x}-\boldsymbol{y}\|/2\sigma^2)$ and the same $m = 300$ anchor samples on each dataset. The kernel parameter σ is tuned to an appropriate value on each dataset. It is noted that we did not assume a specific kernel, and that any kernel satisfying the Mercer's condition can be used in KLSH, KSH^0 and KSH. To run 1-AGH and 2-AGH, we set $m = 300,\ s = 3$ and assign 300 K-means clustering (5 iterations) centers to their used anchor points, so the anchors implicated in AGH algorithms are different from what we use for KSH^0 and KSH. As the experiments conducted in Chapter 4, we follow two search procedures *hash lookup* and *Hamming ranking* using 8 to 48 hash bits (this time we do not try 64 bits in order to highlight the compact hashing performance with shorter hash codes). Hash lookup consumes constant search time over a single hash table for each compared hashing method. We carry out hash lookup within a Hamming radius 2 and report the search precision.

Like Chapter 4, we also follow [186] to treat failing to find any hash bucket for a query as zero precision, different from [191][93][137] which ignored the failed queries in computing the mean hash lookup precision over all queries. As well, to quantify the failing cases, we report the hash lookup success rate: the proportion of the queries resulting in successful hash lookup in all queries. On the other side, Hamming ranking, that is fast enough with short hash codes in practice, measures the search quality through ranking the retrieved samples according to their Hamming distances to a specific query in the Hamming space.

5.5.1 CIFAR-10

This dataset is partitioned into two parts: a training set of 59,000 images and a test query set of 1,000 images evenly sampled from ten classes. We uniformly randomly sample 100 and 200 images from each class respectively, constituting 1,000 and 2,000 labeled subsets for training (semi-)supervised hashing methods SSH, LDAH, BRE, MLH, KSH^0 and KSH. The pairwise label matrices S are immediately acquired since the exact labels are available. To run BRE and MLH that admit 1,0 labels, we assign the labels of similar pairs to 1 and those of dissimilar pairs to 0. In terms of true semantic neighbors, we report the mean precision of Hamming radius 2 hash lookup, the success rate of hash lookup, the mean average precision (MAP) of Hamming ranking, and the mean precision-recall curves of Hamming ranking over 1,000 query images. All of the evaluation results are shown in Tables 5.5 and 5.6 and Figures 5.5, 5.6 and 5.7. For every compared method, we also report the training time for compressing all database images into compact codes as well as the test time for coding each query image.

As shown in Tables 5.5 and 5.6 and Figures 5.5, 5.6 and 5.7, KSH achieves the highest search accuracy (hash lookup precision with ≤ 32 bits, MAP, and PR-curve) and the second best is KSH^0. We find that 1-AGH and 2-AGH already work well without using any supervised information, and outperform the (semi-)supervised methods SSH and LDAH. They even achieve higher hash lookup precision than KSH^0 and KSH at 48 bits, as shown in Figure 5.5(b). The gain in MAP of KSH ranges from 27% to 46% over the best competitor except KSH^0. The prominent superiority of KSH corroborates that the proposed hashing objective $\mathcal{Q}$ and two optimization techniques including spectral relaxation and sig-

Table 5.5: Hamming ranking performance on **CIFAR-10** (60K). l denotes the number of labeled examples for training (semi-)supervised hashing methods. Six unsupervised methods LSH, PCAH, SH, KLSH, 1-AGH and 2-AGH do not use any labels. All training and test time is recorded in second. At a fixed bit number, two highest MAP values achieved by hashing are displayed in boldface type.

Method	$l = 1,000$				
	MAP			**Train Time**	**Test Time**
	12 bits	24 bits	48 bits	48 bits	48 bits
ℓ_2 Scan	0.1752			—	
LSH	0.1133	0.1245	0.1188	0.5	0.8×10^{-5}
PCAH	0.1368	0.1333	0.1271	1.5	0.9×10^{-5}
SH	0.1330	0.1317	0.1352	3.0	4.0×10^{-5}
KLSH	0.1212	0.1425	0.1602	1.6	4.3×10^{-5}
1-AGH	0.1705	0.1805	0.1685	11.0	2.6×10^{-5}
2-AGH	0.1776	0.1812	0.1842	11.4	3.2×10^{-5}
SSH	0.1514	0.1595	0.1755	2.1	0.9×10^{-5}
LDAH	0.1380	0.1334	0.1267	0.7	0.9×10^{-5}
BRE	0.1817	0.2024	0.2060	494.7	2.9×10^{-5}
MLH	0.1545	0.1932	0.2074	3666.3	1.8×10^{-5}
KSH^0	**0.1846**	**0.2047**	**0.2181**	7.0	3.3×10^{-5}
KSH	**0.2325**	**0.2588**	**0.2836**	156.1	4.3×10^{-5}

Table 5.6: Hamming ranking performance on **CIFAR-10** (60K). l denotes the number of labeled examples for training (semi-)supervised hashing methods. All training and test time is recorded in second. At a fixed bit number, two highest MAP values achieved by hashing are displayed in boldface type.

Method	$l = 2,000$				
	MAP			**Train Time**	**Test Time**
	12 bits	24 bits	48 bits	48 bits	48 bits
ℓ_2 Scan	0.1752			—	
SSH	0.1609	0.1758	0.1841	2.2	0.9×10^{-5}
LDAH	0.1379	0.1316	0.1257	1.1	0.9×10^{-5}
BRE	0.1847	0.2024	0.2074	1392.3	3.0×10^{-5}
MLH	0.1695	0.1953	0.2288	3694.4	2.0×10^{-5}
KSH^0	**0.2271**	**0.2461**	**0.2545**	9.4	3.5×10^{-5}
KSH	**0.2700**	**0.2895**	**0.3153**	564.1	4.5×10^{-5}

Table 5.7: Comparison with SVM hashing on **CIFAR-10**. MAP of Hamming ranking is reported for SVM hashing, KSH^0 and KSH using 10 (the number of classes) hash bits.

MAP	$l = 1,000$	$l = 2,000$
SVM hashing	0.1772	0.2202
KSH^0	0.1832	0.2210
KSH	**0.2290**	**0.2517**

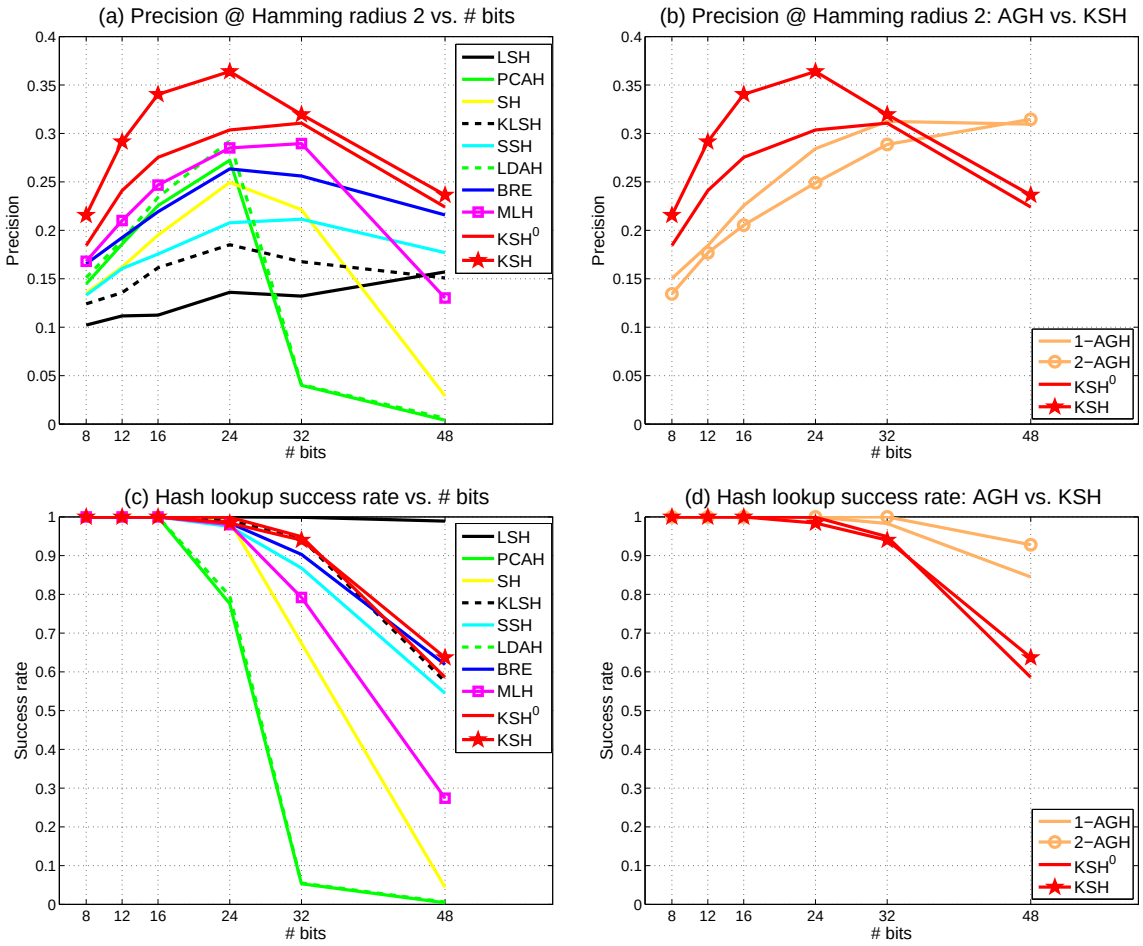


Figure 5.5: The hash lookup results on **CIFAR-10**. Six (semi-)supervised hashing methods use 1K labeled examples. (a)(b) Mean precision of Hamming radius 2 hash lookup; (c)(d) hash lookup success rate.

moid smoothing are so successful that the semantic supervision information is maximally utilized. For the hash lookup success rate, KSH is lower than LSH, 1-AGH and 2-AGH but still superior to the others, as shown in Figures 5.5(c)(d). More notably, KSH with only 48 binary bits and a limited amount of supervised information (1.7% and 3.4% labeled samples) significantly outperforms ℓ_2 linear scan (0.1752 MAP) in the GIST feature space, accomplishing up to 1.8 times higher MAP. Compared to BRE and MLH, KSH^0 (several seconds) and KSH (several minutes) are much faster in supervised training. The test time of KSH^0 and KSH is acceptably fast, comparable to that of the nonlinear hashing methods

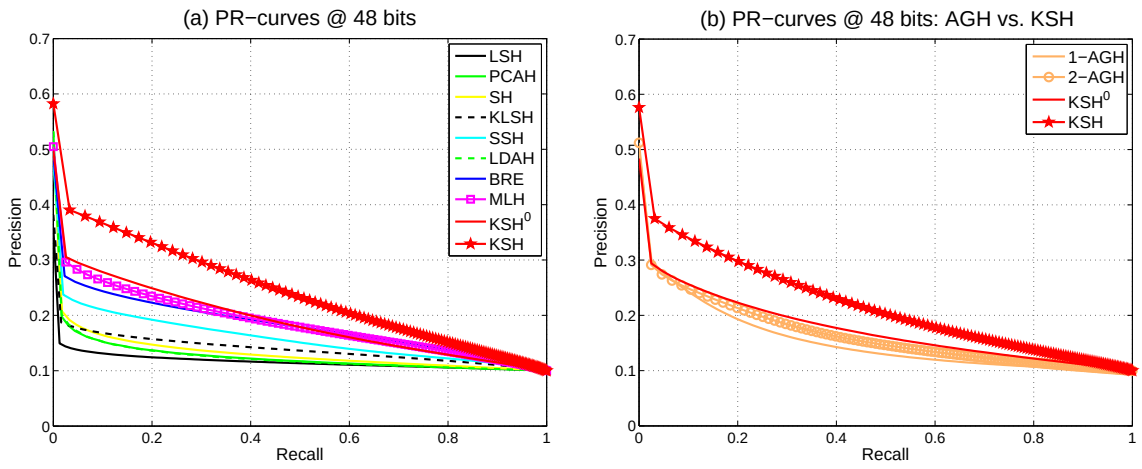


Figure 5.6: Mean precision-recall curves of Hamming ranking with 48 bits on **CIFAR-10**.

SH, KLSH, 1-AGH, 2-AGH and BRE.

Incorporating twice (2,000) labeled examples for training (semi-)supervised hashing, we further plot the mean precision of Hamming radius 2 hash lookup, the success rate of hash lookup, and the mean precision-recall curves of Hamming ranking for six (semi-)supervised hashing methods in terms of finding true semantic neighbors. All of the corresponding results are shown in Figure 5.7. Again, we observe that KSH achieves the highest search accuracy (hash lookup precision, Hamming ranking precision and recall) and the second best is KSH^0. For the hash lookup success rate, KSH is lower than SSH only at 48 bits.

Lastly, we do two additional groups of evaluations to further validate the success of KSH. We first explicitly contrast AGH and KSH. The reported results in Table 5.5 indicate that the GIST feature space did not present evident manifolds, so 2-AGH is only slightly better than ℓ_2 linear scan in MAP. Through taking full advantage of the given supervised information, KSH breaks through the limited MAP, accomplishing 0.2836 up to 0.3153 MAP with 48 bits and exceeding 2-AGH (0.1842 MAP) by 54% up to 71%. Second, we compare against *SVM hashing* which takes the signs of SVM [149] classifiers as the hash functions. Since the supervised information discussed in this subsection supplies sample-level labels, we can immediately use the labeled samples to train 10 (the number of classes) standard kernel SVMs in the one-versus-all fashion. Hence, we fairly compare KSH^0 and KSH using 10 hash bits to SVM hashing. The MAP results are listed in Table 5.7 which clearly shows

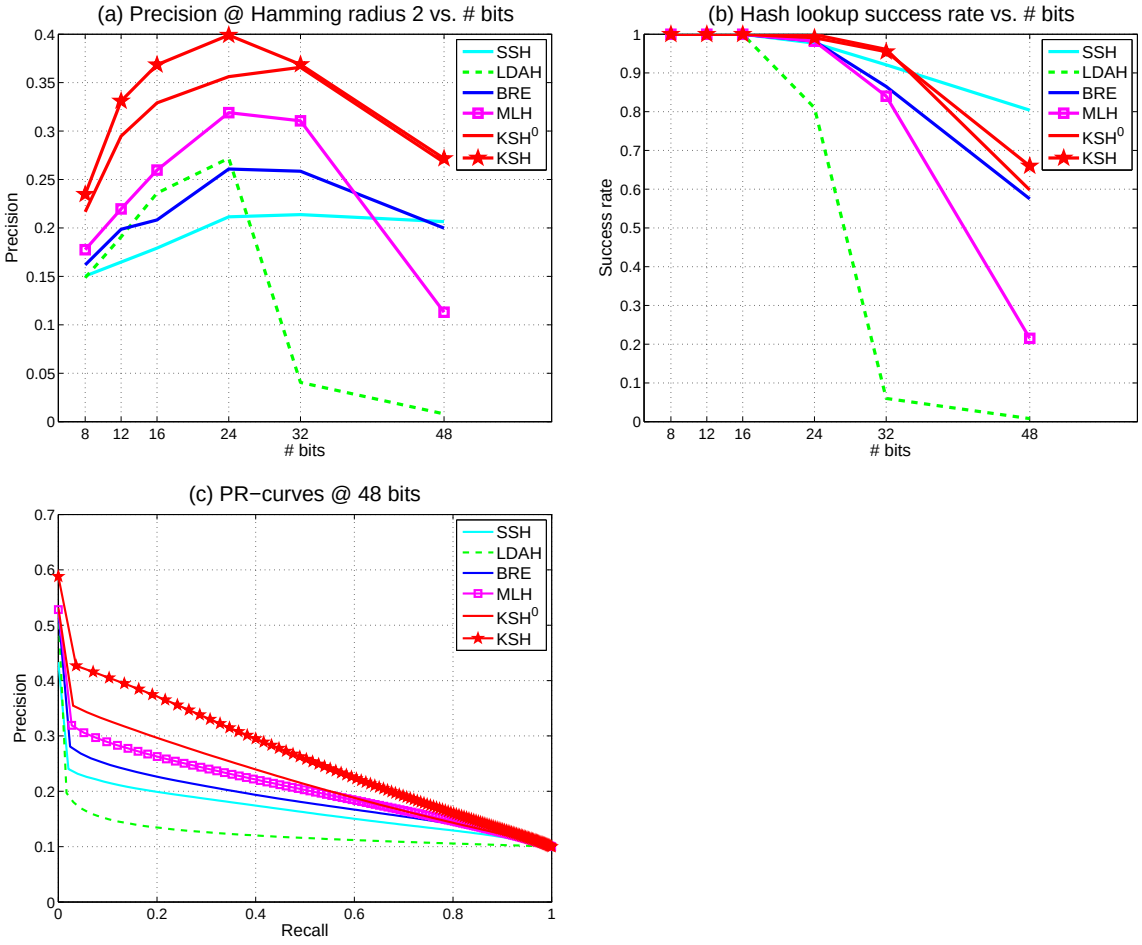


Figure 5.7: The results on **CIFAR-10**. Six (semi-)supervised hashing methods use 2K labeled examples. (a) Mean precision of Hamming radius 2 hash lookup; (b) hash lookup success rate; (c) mean precision-recall curves of Hamming ranking with 48 bits.

Figure 5.8: 100 query images from the **Tiny-1M** dataset.

that both KSH^0 and KSH are superior to SVM hashing. The reason is also very apparent. KSH^0 and KSH yield more discriminative hash codes and result in larger margin in the Hamming space than SVM hashing, as KSH^0 and KSH push differently labeled samples to the Hamming distance $r = 10$ while SVM hashing only to the Hamming distance 1.

5.5.2 Tiny-1M

The one million subset of the 80 million tiny image benchmark [174], which has been frequently utilized to evaluate hashing, was sampled to construct the training set and a separate subset of 2,000 images was used as the test (query) set [186], the union of which is called the **Tiny-1M** dataset. One hundred query images are displayed in Figure 5.8. The 10,000×10,000 pairwise pseudo label matrix S was constructed according to the ℓ_2 distance. Concretely, [186] randomly selected 10,000 images from the training set and computed their

Table 5.8: Hamming ranking performance on **Tiny-1M**. l denotes the number of pseudo-labeled examples for training (semi-)supervised hashing methods. Six unsupervised methods LSH, PCAH, SH, KLSH, 1-AGH and 2-AGH do not use any labels. All training and test time is recorded in second. At a fixed bit number, two highest MP values achieved by hashing are displayed in boldface type.

Method	$l = 5,000$				
	MP/50K			**Train Time**	**Test Time**
	12 bits	24 bits	48 bits	48 bits	48 bits
LSH	0.1107	0.1421	0.1856	3.0	0.3×10^{-5}
PCAH	0.2371	0.2159	0.1954	7.1	0.4×10^{-5}
SH	0.2404	0.2466	0.2414	47.1	3.3×10^{-5}
KLSH	0.1834	0.2490	0.3008	9.9	2.6×10^{-5}
1-AGH	**0.3152**	**0.3405**	0.3429	191.3	2.7×10^{-5}
2-AGH	0.2921	0.3253	0.3426	201.3	3.2×10^{-5}
SSH	0.1985	0.2923	0.3785	14.8	0.6×10^{-5}
LDAH	0.2365	0.2208	0.2077	5.8	0.6×10^{-5}
BRE	0.2782	0.3400	**0.3961**	18443.0	3.3×10^{-5}
MLH	0.2071	0.2592	0.3723	4289.2	1.4×10^{-5}
KSH^0	0.1889	0.2295	0.2346	56.0	3.1×10^{-5}
KSH	**0.3164**	**0.3896**	**0.4579**	2210.3	3.2×10^{-5}

Euclidean distance matrix D from which S was obtained by using the rule: $S_{ij} = 1$ if D_{ij} is within 5% of the whole one million distances and $S_{ij} = -1$ if D_{ij} is more than 95%. The top 5% distances from a query were also used as the groundtruths of nearest metric neighbors. As most entries in S are zeros, to follow [93] each 0 label is replaced by $1 - \hat{D}_{ij}/2$ in which $0 \leq \hat{D}_{ij} \leq 2$ is the normalized ℓ_2 distance. Like above experiments, we treat 1,-1 labels in S as 1,0 labels for running BRE and MLH.

In terms of ℓ_2 metric neighbors (each query has 50,000 groundtruth neighbors), we evaluate the mean precision of Hamming radius 2 hash lookup, the success rate of hash

Table 5.9: Hamming ranking performance on **Tiny-1M**. l denotes the number of pseudo-labeled examples for training (semi-)supervised hashing methods. All training and test time is recorded in second. At a fixed bit number, two highest MP values achieved by hashing are displayed in boldface type.

Method	$l = 10,000$				
	MP/50K			**Train Time**	**Test Time**
	12 bits	24 bits	48 bits	48 bits	48 bits
SSH	0.1718	0.2767	0.3524	16.9	0.6×10^{-5}
LDAH	0.2373	0.2238	0.2072	13.3	0.6×10^{-5}
BRE	**0.2762**	**0.3403**	**0.3889**	27580.0	3.3×10^{-5}
MLH	0.1875	0.2873	0.3489	4820.8	1.8×10^{-5}
KSH^0	0.1886	0.1985	0.2341	84.5	3.2×10^{-5}
KSH	**0.3216**	**0.3929**	**0.4645**	2963.3	3.3×10^{-5}

lookup, the mean top-50K precision (MP/50K) of Hamming ranking[1], and the mean precision/recall curves of Hamming ranking. The results are shown in Tables 5.8 and 5.9 and Figures 5.9, 5.10 and 5.11. To illustrate the overfitting phenomenon in (semi-)supervised hashing methods, we inspect the half supervision, i.e., the 5,000 pseudo-labeled images, and the full 10,000 labeled images, respectively. KSH refrains from overfitting, showing higher MP when absorbing more supervision. On the contrary, LDAH remains MP but SSH, BRE, MLH and KSH^0 all suffer from overfitting to different extents – their MP drops faced with increased supervision.

Consistent with the finding from the results on **CIFAR-10**, we can see that KSH consistently attains the highest search accuracy (hash lookup precision with ≤ 32 bits, MP, precision-curve, and recall-curve) and the same highest hash lookup success rate as LSH. The gain in MP of KSH ranges from 13% to 19% over the best competitor. Referring to Section 5.4.3, the spectral relaxation solutions employed by KSH^0 might become poor when

[1]As computing MAP at the million scale is very slow, we instead compute MP over the scope of groundtruth neighbors.

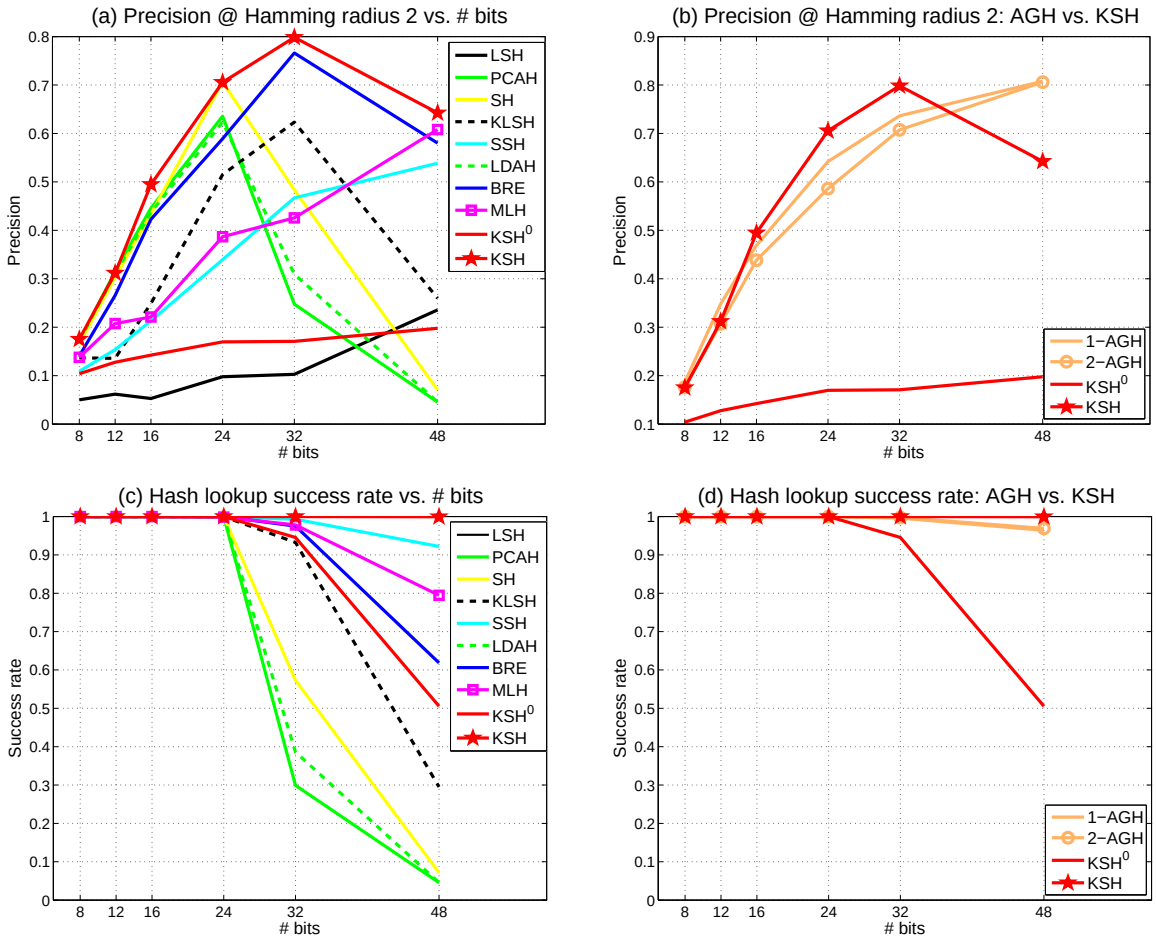


Figure 5.9: The hash lookup results on **Tiny-1M**. Six (semi-)supervised hashing methods use 5,000 pseudo-labeled examples. (a)(b) Mean precision of Hamming radius 2 hash lookup; (c)(d) hash lookup success rate.

l is larger, which is verified in these experiments where KSH^0 performs as poorly as LDAH. It is noticeable that KSH with only 48 binary bits and a very limited amount of supervised information (0.5% and 1% pseudo-labeled samples) can retrieve about 46% groundtruth ℓ_2 neighbors (see Tables 5.8 and 5.9) and reach higher precision by using longer bits. Therefore, we can say that KSH well preserves the ℓ_2 metric similarities in the Hamming code space by taking full advantage of the neighborhood structure captured into the supervised label matrix S. From Table 5.8, we find that 1-AGH obtains MP close to KSH when using ≤ 24 bits but falls far behind KSH when using 48 bits.

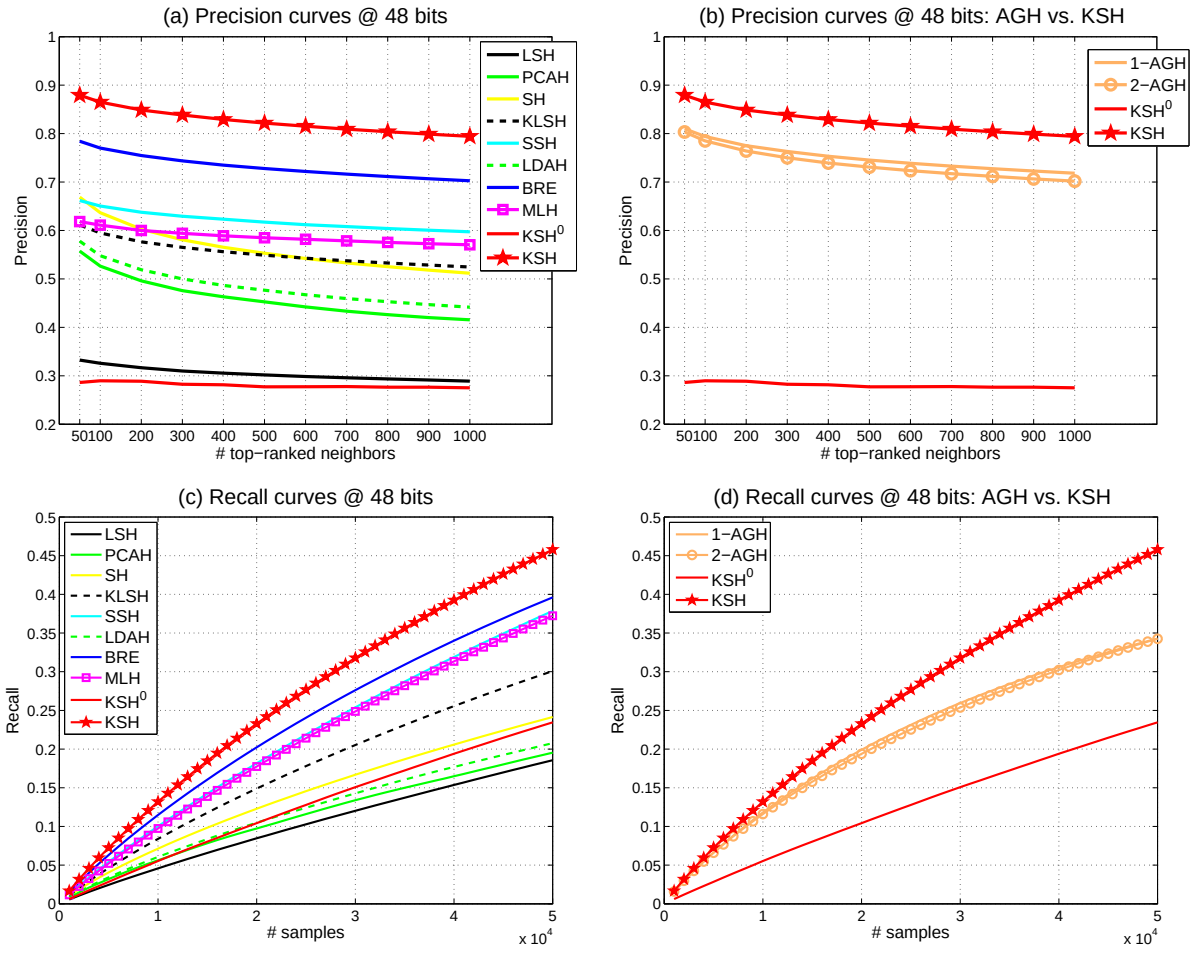


Figure 5.10: The Hamming ranking results on **Tiny-1M**. Six (semi-)supervised hashing methods use 5,000 pseudo-labeled examples. (a)(b) Mean precision curves of Hamming ranking with 48 bits; (c)(d) mean recall curves of Hamming ranking with 48 bits.

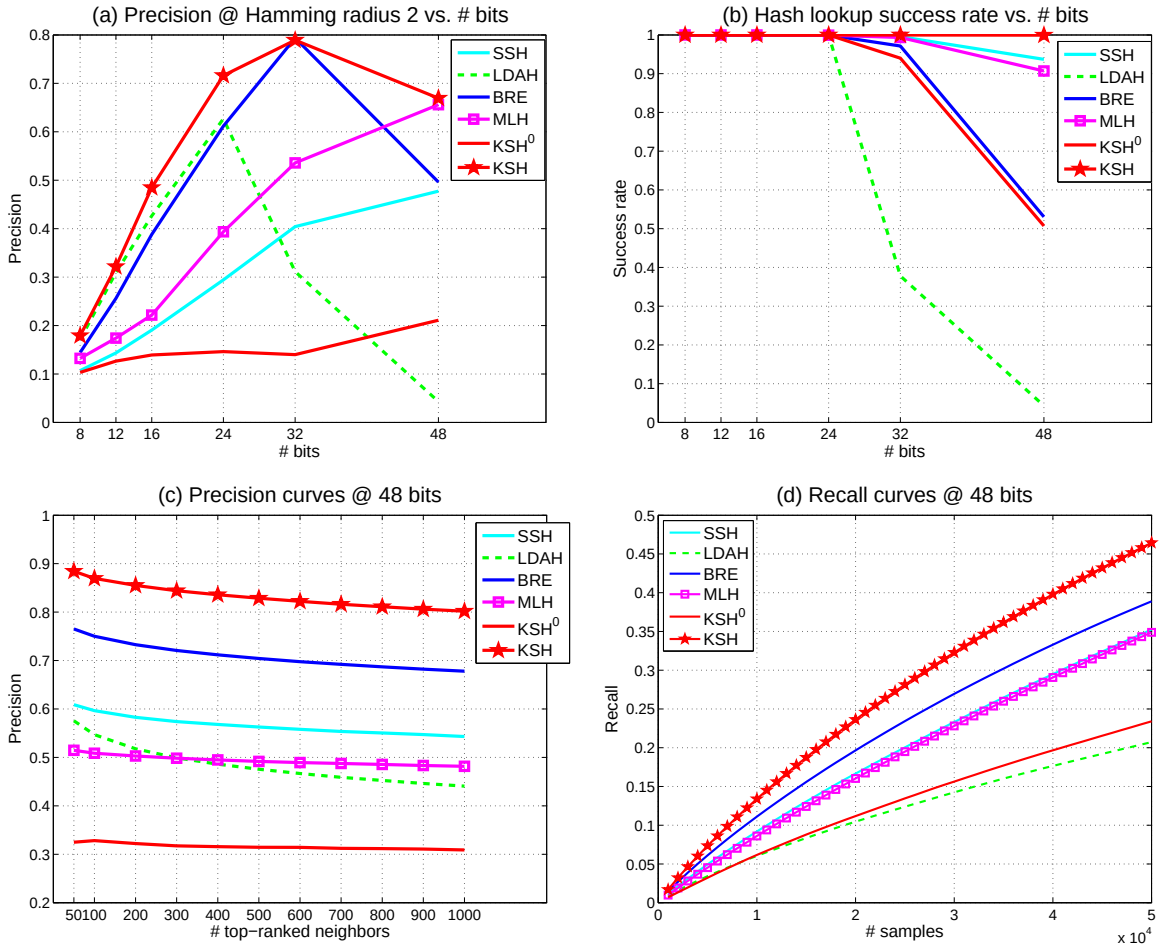


Figure 5.11: The results on **Tiny-1M**. Six (semi-)supervised hashing methods use 10,000 pseudo-labeled examples. (a) Mean precision of Hamming radius 2 hash lookup; (b) hash lookup success rate; (c) mean precision curves of Hamming ranking with 48 bits; (d) mean recall curves of Hamming ranking with 48 bits.

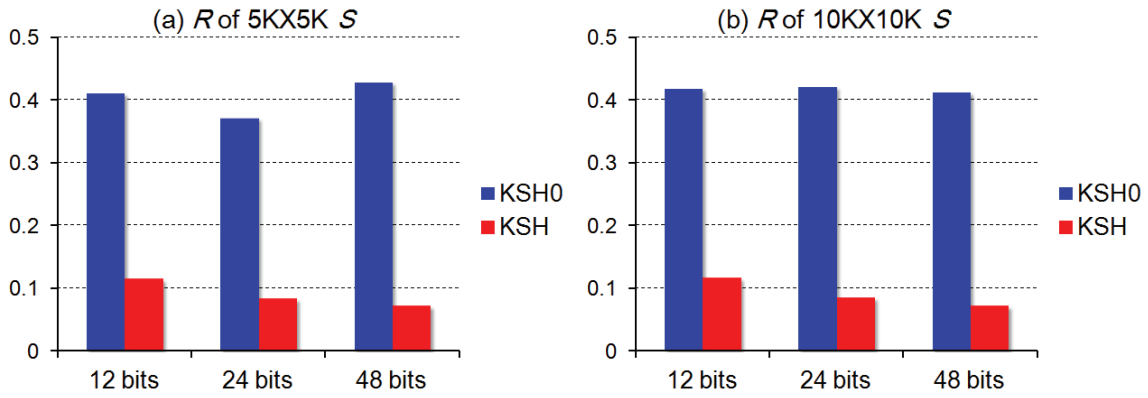


Figure 5.12: The average quadratic residues $R = \mathcal{Q}/l^2 = \|B_l B_l^\top/r - S\|/l^2$ of reconstructing the pairwise label matrix S with the r-bit codes B_l learned by KSH^0 and KSH, respectively. (a) R of reconstructing 5,000×5,000 S with 5,000 binary codes; (b) R of reconstructing 10,000×10,000 S with 10,000 binary codes.

To inspect the performance with more supervised information, we plot the mean precision of Hamming radius 2 hash lookup, the success rate of hash lookup, the mean precision curves of Hamming ranking, and the mean recall curves of Hamming ranking for six (semi-)supervised hashing methods taking 10,000 pseudo-labeled examples. All of the results are evaluated based on ℓ_2 metric neighbors, and shown in Figure 5.11. Figure 5.11 displays that KSH still achieves the highest search accuracy (hash lookup precision, Hamming ranking precision and recall). For the hash lookup success rate, KSH is also the highest and almost approaches the 100% success rate, as disclosed by Figure 5.11(b).

It is noticeable that the search performance of KSH^0 drops on this dataset. We have interpreted the reason in Section 5.4.3. Here we further investigate the solutions that the spectral relaxation optimization scheme (used by KSH^0) and the sigmoid smoothing optimization scheme (used by KSH) result in. Respectively with respect to 5,000 and 10,000 pseudo-labeled data $\mathcal{X}_l$, we save the final objective function values $\mathcal{Q}$ (that is formulated in eq. (5.8)(5.10) in Section 5.4) with the optimal compact codes $\mathcal{B}_l$ of $\mathcal{X}_l$ solved by KSH^0 and KSH. Specifically, Figure 5.12 visualizes the normalized $\mathcal{Q}$, that is, $R = \mathcal{Q}/l^2 = \|B_l B_l^\top/r - S\|/l^2$. Actually, R is the average quadratic residue of reconstructing the $l \times l$ label matrix S with the r-bit codes B_l. From Figure 5.12, we can clearly see that the reconstruction residues

Figure 5.13: Top six neighbors of four query images, returned by SSH, BRE, MLH and KSH using 5,000 pseudo-labeled training images and 48 binary bits on the **Tiny-1M** dataset.

caused by KSH are significantly smaller than those of KSH^0, indicating that the sigmoid smoothing optimization is more stable and finer than the spectral relaxation optimization though the former takes the latter as the initialization.

Besides the quantitative evaluations, Figure 5.13 showcases some exemplar query images and their retrieved neighbors with 48 bits, where KSH still exhibits the best search quality in visual relevance. To obtain the exact ranks of the retrieved neighbors, we perform ℓ_2 linear scan in a short list of top 0.01% Hamming ranked neighbors like [93]. Such search time is close to real time, costing only 0.005 second per query.

5.6 Summary and Discussion

We accredit the success of the presented KSH approach to three main aspects: (1) kernel-based hash functions were exploited to handle linearly inseparable data; (2) an elegant objective function designed for supervised hashing was skillfully formulated based on code inner products instead of Hamming distances; and (3) a greedy optimization algorithm was deployed to solve the hash functions efficiently. Extensive image retrieval results shown on large image corpora up to one million have demonstrated that KSH surpasses the state-of-the-art hashing methods by a large margin in searching both semantic neighbors and metric neighbors. We thus believe that hashing via optimizing code inner products is a promising direction, generating hash codes superior to the prior methods relying on optimizing Hamming distances. Furthermore, KSH does not need any special assumptions about the data other than a predefined kernel function. For the datasets where the underlying manifolds may not exist or are not evident, KSH works better than AGH which depends on the manifold assumption.

While in this chapter we select image retrieval as the testbed for the proposed hashing approach, we want to emphasize that it is a general method and can be applied to a large spectrum of computer vision and information retrieval problems such as duplicate image, video and document detection, image matching, image classification, and so on. Because of the high search accuracy reported in the experiments, we believe that KSH will become a new criterion for compact hashing (with less than 100 hash bits), and that supervised

hashing can achieve much higher accuracy than unsupervised and semi-supervised hashing with moderate supervision. We thus suggest providing proper supervised information to hashing, which crucially results in better performance than conventional hashing.

Till now, we have discussed two key paradigms for learning to hash: unsupervised hashing in Chapter 4 and supervised hashing in this chapter. All of the proposed hashing approaches exhibit appealing NN search performance on benchmark databases which vary in size from 60,000 to one million. Training our approaches including both AGH and KSH allows space and time complexities linear in the size of databases, so they can scale up to larger databases conceivably reaching billions of samples due to their economical training expenses.

There exist some other promising directions for hashing, such as (1) strategically constructing multiple hash tables [168][197], (2) designing particular hashing algorithms for sparse data vectors such as the increasingly popular *minwise hashing* technology [19][20][106], (3) devising rank-preserving hash functions [45][198], and (4) learning task-specific hash functions [150][81][127] by incorporating query-specific supervised information. For direction (2), how to design a supervised version of the originally unsupervised minwise hashing algorithms is very interesting and challenging. It will lead to widespread applications in text and document retrieval. Directions (3) and (4) share a similar characteristic: pursuing more informative search results where the exact ranks of returned neighbors are preferred. How to adapt the proposed supervised hashing technique to incorporate broad types of supervised information is also very important. For example, we can make supervised hashing amenable to the relative rank orders over triplets $\{(\boldsymbol{q}_t, \boldsymbol{x}_i, \boldsymbol{x}_j)\}$ in which $\boldsymbol{q}_t$ is a query and sample $\boldsymbol{x}_i$ is more relevant to $\boldsymbol{q}_t$ than sample $\boldsymbol{x}_j$.

Chapter 6

Hyperplane Hashing

Distinct from the conventional hashing techniques and our proposed AGH and KSH all of which address the problem of fast point-to-point nearest neighbor search, this chapter studies a new scenario "point-to-hyperplane" hashing, where the query is a hyperplane instead of a data point. Such a new scenario requires hashing the hyperplane query to near database points, which is difficult to accomplish because point-to-hyperplane distances are quite different from routine point-to-point distances in terms of the computation mechanism. Despite the bulk of research on hashing, this special hashing paradigm is rarely touched.

However, hyperplane hashing, aiming at rapidly searching the database points near a given hyperplane, is actually quite important for many applications such as large-scale active learning with SVMs. In SVM-based active learning, the well proven sample selection strategy is to search in the unlabeled sample pool to identify the sample closest to the current hyperplane decision boundary, thus providing the most useful information for improving the learning model - SVM. When making such active learning scalable to gigantic databases, exhaustive search for the point nearest to the hyperplane is not efficient for the online sample selection requirement, so novel hashing methods that can principally handle hyperplane queries have been called for.

Unfortunately, the existing hyperplane hashing methods are randomized in nature and need long hash codes to achieve reasonable search accuracy, thus suffering from reduced search speed and large memory overhead. To this end, this chapter presents a novel hyperplane hashing technique, described in our recent work [118], which is able to yield high-

quality compact hash codes. The core idea underlying our technique is the bilinear form adopted in the proposed hash functions, which leads to a higher collision probability than all of the existing hyperplane hash functions when using random projections. To further increase the performance, we develop a learning based framework in which the bilinear functions are directly learned from the input data. This results in short yet discriminative codes, and significantly boosts the search performance compared to the prior random projection based solutions. Large-scale active learning experiments carried out on two datasets of up to one million samples demonstrate the overall superiority of the developed approach.

The hyperplane hashing technique presented in this chapter can also be considered as an excellent bridge connecting the classification research explored in Part I of this thesis and the hashing research investigated in Part II. We establish such a bridge through employing the hashing technique to significantly speed up the search process of choosing the optimal sample needed by active learning that incrementally refines the SVM classification model. To that end, this chapter serves as a consummate one for ending the thesis.

In the remainder of the chapter, we introduce the problem background of hyperplane hashing in Section 6.1, review the related work in Section 6.2, illustrate the notations that we will use in Section 6.3, formulate the point-to-hyperplane search problem in Section 6.4, present our two approaches including the randomized version in Section 6.5 and the learning-based version in Section 6.6, show the experimental results in Section 6.7, and finally give our summary and discussion in Section 6.8.

6.1 Problem Background

Most of the existing hashing methods discussed in Chapter 4 and 5 try to solve the problem of "point-to-point" nearest neighbor search. Namely, both queries and database items are represented as individual points in some feature space. Considering complex structures of real-world data, other forms of hashing paradigms beyond point-to-point search have also been explored in the literature, e.g., "subspace-to-subspace" nearest neighbor search [8]. In this chapter, we address a more challenging "*point-to-hyperplane*" search problem, where queries come as hyperplanes in $\mathbb{R}^d$, i.e., $(d-1)$-dimensional subspaces, and database items

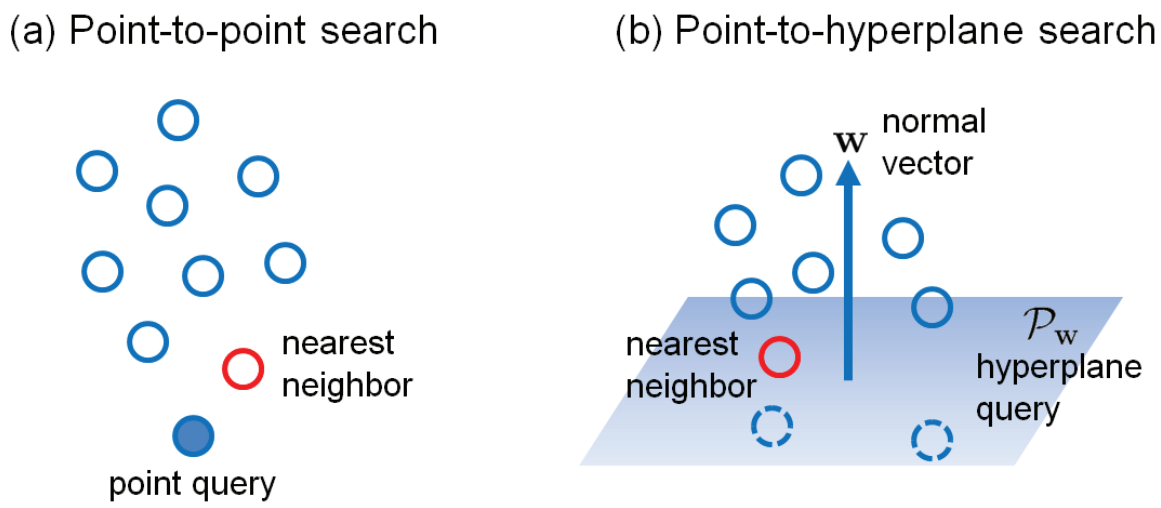


Figure 6.1: Two distinct nearest neighbor search problems. (a) Point-to-point search, the blue solid circle represents a point query and the red circle represents the found nearest neighbor point. (b) Point-to-hyperplane search, the blue plane denotes a hyperplane query $\mathcal{P}_{\boldsymbol{w}}$ with $\boldsymbol{w}$ being its normal vector, and the red circle denotes the found nearest neighbor point.

are conventional points. Then the search problem is formally defined as follows:

Given a hyperplane query and a database of points, return the database point which has minimal distance to the hyperplane.

We plot Figure 6.1 to visually describe the two NN search problems discussed above. In the literature, not much work has been done on the point-to-hyperplane search problem except [75] which demonstrated the vital importance of such a problem in making SVM-based active learning viable for massive data sets.

Let us include a brief review of active learning in order to elicit the hyperplane hashing problem more naturally. *Active learning* (AL), also known as pool-based active learning, circumvents the high cost of blind labeling by judiciously selecting a few samples to label. In each iteration, a typical AL learner seeks the most *informative* sample from an unlabeled sample pool, so that maximal information gain is achieved after labeling the selected sample. Subsequently, the learning model is re-trained on the incrementally labeled sample set. The

classical AL algorithms [173][148][22] used SVMs as learning models. Based on the theory of "version spaces" [173], it was provably shown that the best sample to select is simply the one closest to the current decision hyperplane if the assumption of symmetric version spaces holds. Unfortunately, the active selection method faces serious computational challenges when applied to gigantic databases. An exhaustive search to find the best sample is usually computationally prohibitive. Hence, quickening point-to-hyperplane search methods are strongly desired to scale up active learning on large real-world data sets.

In the early exploration aiming at expediting the active sample selection process, that is the core of all active learning algorithms, either clustering or randomly subsampling of the data pool [17][140] were used, both of which lack theoretical foundations. Recently, two principled hashing schemes were proposed in [75] to specifically cope with hyperplane queries. Compared with the brute-force scan through all database points, the schemes proposed in [75] are significantly more efficient, while providing theoretical guarantees of sub-linear query time and tolerable accuracy losses for retrieved approximate nearest neighbors. Consequently, when applying hyperplane hashing to the sample selection task needed by SVM-based active learning, one just needs to scan orders of magnitude fewer database points than the linear scan to deliver the next active label request, thereby making active learning scalable.

6.2 Related Work

Fast approximate nearest neighbor search arises commonly in a variety of domains and applications due to massive growth in data that one is confronted with. An attractive solution to overcome the speed bottleneck that an exhaustive linear scan incurs is the use of algorithms from the *Locality-Sensitive Hashing* (LSH) family [52][27][39][4][94] which use random projections to convert input data into binary hash codes. Although enjoying theoretical guarantees on sub-linear hashing/search time and the accuracy of the returned neighbors, LSH-related methods typically need long codes and a large number of hash tables to achieve good search accuracy. This may lead to considerable storage overhead and reduced search speed. Hence, in the literature, directly learning data-dependent hash

functions to generate compact codes has become popular. Such hashing typically needs a small number of bits per data item and can be designed to work well with a single hash table and constant hashing time. The state-of-the-arts include unsupervised hashing [191][56][117] (see more details in Chapter 4), semi-supervised hashing [186][128], and supervised hashing [93][137][163][116] (see more details in Chapter 5).

Recently, two hyperplane hashing schemes were first proposed in [75] to tackle rapid point-to-hyperplane search. Compared with the brute-force scan through all of the database points, the two schemes are significantly more efficient, offering theoretical guarantees of sub-linear query time and tolerable losses of accuracy for retrieved approximate nearest neighbors. Using the simpler one of the two methods in [75], [180] successfully applied hyperplane hashing to accelerate SVM active learning and showed promising SVM classification performance in the practical computer vision application, large-scale live object detection.

More concretely, two families of randomized hash functions in [75] were proved locality-sensitive to the angle between a database point and a hyperplane query; however, long hash bits and plentiful hash tables are required to cater for the theoretical guarantees. Actually, 300 bits and 500 tables were adopted in [75] to achieve reasonable performance, which incurs a heavy burden on both computation and storage. To mitigate the above mentioned issues, this chapter develops a compact hyperplane hashing scheme which exploits only a single hash table with several tens of hash bits to tackle point-to-hyperplane search. The thrust of our hashing scheme is to design and learn *bilinear* hash functions such that nearly parallel input vectors are hashed to the same bits whereas nearly perpendicular input vectors are hashed to different bits. In fact, we first show that even without any learning, the randomized version of the proposed bilinear hashing gives a higher near-neighbor collision probability than the existing methods.

Next, we cast the bilinear projections in a learning framework and show that one can do even better by using learned hash functions. Given a hyperplane query, its normal vector is used as the input and the corresponding hash code is obtained by concatenating the output bits from the learned hash functions. Then, the database points whose codes have the farthest Hamming distances to the query's code are retrieved. Critically, the retrieved

points, called *near-to-hyperplane neighbors*, maintain small angles to the hyperplane following our learning principle. Experiments in Section 6.7 conducted on two large datasets up to one million corroborate that our approach enables scalable active learning with good performance. Finally, while in this chapter we select SVM active learning as the testbed for hyperplane hashing, we want to highlight that the developed compact hyperplane hashing is a general method and thus applicable to a large spectrum of machine learning problems such as minimal tangent distance pursuit [195] and cutting-plane based maximum margin clustering [207].

6.3 Notations

We first define the notations and symbols that will be needed in this chapter. All notations as well as their definitions are listed in Tables 6.1 and 6.2.

6.4 Point-to-Hyperplane Search

First of all, let us revisit the well-known margin-based AL strategy proposed by [173]. For the convenience of expression, we append each data vector with a 1 and use a linear kernel. Then, the SVM classifier [149] becomes $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$ where vector $\boldsymbol{x} \in \mathbb{R}^d$ represents a data point and vector $\boldsymbol{w} \in \mathbb{R}^d$ determines a hyperplane $\mathcal{P}_{\boldsymbol{w}}$ passing through the origin. Figure 6.2(a) displays the geometric relationship between $\boldsymbol{w}$ and $\mathcal{P}_{\boldsymbol{w}}$, where $\boldsymbol{w}$ is the vector normal to the hyperplane $\mathcal{P}_{\boldsymbol{w}}$. Given a hyperplane query $\mathcal{P}_{\boldsymbol{w}}$ and a database of points $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^n$, the active selection criterion prefers the most *informative* database point

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x} \in \mathcal{X}} \; D(\boldsymbol{x}, \mathcal{P}_{\boldsymbol{w}}), \tag{6.1}$$

which has the minimum margin to the SVM's decision boundary $\mathcal{P}_{\boldsymbol{w}}$. Note that $D(\boldsymbol{x}, \mathcal{P}_{\boldsymbol{w}}) = |\boldsymbol{w}^\top \boldsymbol{x}|/\|\boldsymbol{w}\|$ is the *point-to-hyperplane distance*. To derive provable hyperplane hashing like [75], this chapter focuses on a slightly modified "distance" $\dfrac{|\boldsymbol{w}^\top \boldsymbol{x}|}{\|\boldsymbol{w}\|\|\boldsymbol{x}\|}$ which is the sine of the *point-to-hyperplane angle*

$$\alpha_{\boldsymbol{x},\boldsymbol{w}} = \left|\theta_{\boldsymbol{x},\boldsymbol{w}} - \frac{\pi}{2}\right| = \sin^{-1} \frac{|\boldsymbol{w}^\top \boldsymbol{x}|}{\|\boldsymbol{w}\|\|\boldsymbol{x}\|}, \tag{6.2}$$

Table 6.1: Table of notations.

Notation	Definition
n	The number of database points
l	The number of sampled points for training hashing
d	The dimension of database or query points
$i, i^{'}$	The indices of database points
$\boldsymbol{x}_i \in \mathbb{R}^d$	The ith database point
$\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^n$	The data set
$\mathcal{X}_l = \{\boldsymbol{x}_i\}_{i=1}^l$	The sampled data set
$\boldsymbol{w} \in \mathbb{R}^d$	The query normal vector
$\mathcal{P}_{\boldsymbol{w}}$	The query hyperplane
$h : \mathbb{R}^d \mapsto \{1, -1\}$	A hash function for a single bit
$\text{sgn}(x) \in \{1, -1\}$	The sign function that returns 1 for $x > 0$ and -1 otherwise
$f : \mathbb{R}^d \mapsto \mathbb{R}$	A classification function
$D(\boldsymbol{x}, \mathcal{P}_{\boldsymbol{w}})$	The point-to-hyperplane distance from $\boldsymbol{x}$ to $\mathcal{P}_{\boldsymbol{w}}$
$\alpha_{\boldsymbol{x},\boldsymbol{w}} \in [0, \pi/2]$	The point-to-hyperplane angle from $\boldsymbol{x}$ to $\mathcal{P}_{\boldsymbol{w}}$
$\theta_{\boldsymbol{x},\boldsymbol{w}} \in [0, \pi]$	The angle between point $\boldsymbol{x}$ and the hyperplane normal $\boldsymbol{w}$
$\boldsymbol{x}^* \in \mathbb{R}^d$	The most informative sample chosen for active learning
$\boldsymbol{u}, \boldsymbol{v} \sim \mathcal{N}(0, I_{d\times d})$	Two random projection vectors
$\mathbf{U} \sim \mathcal{N}(0, I_{d^2\times d^2})$	A random projection vector in $\mathbb{R}^{d^2}$
$\mathbf{V}(A)$	The vectorial concatenation of input matrix A
$\boldsymbol{z} \in \mathbb{R}^d$	An input vector for hash functions
$h^{\mathcal{A}} : \mathbb{R}^d \mapsto \{1, -1\}$	A hash function from the Angle-Hyperplane Hash family
$h^{\mathcal{E}} : \mathbb{R}^d \mapsto \{1, -1\}$	A hash function from the Embedding-Hyperplane Hash family
$h^{\mathcal{B}} : \mathbb{R}^d \mapsto \{1, -1\}$	A hash function from the Bilinear-Hyperplane Hash family
$\mathbf{Pr}[A] \in [0, 1]$	The probability of an event A
$\mathbf{E}[x] \in \mathbb{R}$	The mathematical expectation of a random variable x
$\mathcal{D}(\ ,\ )$	A distance function used in a LSH algorithm
$r, \epsilon > 0$	Two parameters in a LSH algorithm
$p_1 > p_2 > 0$	Two probabilities in a LSH algorithm
$\rho \in [0, 1]$	The exponent of query time n^ρ achieved by a LSH algorithm

Table 6.2: Table of notations (continued).

Notation	Definition
$c \in \mathbb{N}$	The natural number parameter in a LSH algorithm
e	The base of the natural logarithm
k	The number of hash bits
j	The index of hash bits
$h_j : \mathbb{R}^d \mapsto \{1, -1\}$	The hash function for the jth bit
$(\boldsymbol{u}_j, \boldsymbol{v}_j)$	The jth projection pair
$t_1, t_2 > 0$	Two similarity thresholds
$S = (S_{ii'})_{i,i'} \in \mathbb{R}^{l \times l}$	The pairwise pseudo label matrix
$H(\boldsymbol{x}) = [h_1(\boldsymbol{x}), \cdots, h_k(\boldsymbol{x})]$	The k-bit binary code of sample $\boldsymbol{x}$
$B_l \in \{1, -1\}^{l \times k}$	The code matrix of the sampled data $\mathcal{X}_l$
$\mathcal{Q}$	The objective (cost) function for optimizing $(\boldsymbol{u}_j, \boldsymbol{v}_j)_{j=1}^k$
$R_{j-1} \in \mathbb{R}^{l \times l}$	The $(j-1)$th residue matrix, $j \in [1 : k]$
$g : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$	The objective (cost) function for optimizing $(\boldsymbol{u}_j, \boldsymbol{v}_j)$
$\tilde{g} : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$	The smooth surrogate of g
$\varphi(x) \in \mathbb{R}$	The sigmoid-shaped function to approximate $\text{sgn}(x)$
$\boldsymbol{b}_j \in \{1, -1\}^l$	The jth column vector in the code matrix B_l
$\tilde{\boldsymbol{b}}_j \in \mathbb{R}^l$	The smooth surrogate of $\boldsymbol{b}_j$

where $\theta_{\boldsymbol{x},\boldsymbol{w}} \in [0, \pi]$ is the angle between $\boldsymbol{x}$ and the hyperplane normal $\boldsymbol{w}$. The angle measure $\alpha_{\boldsymbol{x},\boldsymbol{w}} \in [0, \pi/2]$ between a database point and a hyperplane query can readily be reflected into hashing.

As shown in Figure 6.2(b), the goal of hyperplane hashing is to hash a hyperplane query $\mathcal{P}_{\boldsymbol{w}}$ and the informative samples (e.g., $\boldsymbol{x}_1, \boldsymbol{x}_2$) with narrow $\alpha_{\boldsymbol{x},\boldsymbol{w}}$ into the same or nearby hash buckets, meanwhile avoiding to return the uninformative samples (e.g., $\boldsymbol{x}_3, \boldsymbol{x}_4$) with wide $\alpha_{\boldsymbol{x},\boldsymbol{w}}$. Because $\alpha_{\boldsymbol{x},\boldsymbol{w}} = \left|\theta_{\boldsymbol{x},\boldsymbol{w}} - \frac{\pi}{2}\right|$, the point-to-hyperplane search problem can be equivalently transformed to a specific point-to-point search problem where the query is the hyperplane normal $\boldsymbol{w}$ and the desirable nearest neighbor to the raw query $\mathcal{P}_{\boldsymbol{w}}$ is the one whose angle $\theta_{\boldsymbol{x},\boldsymbol{w}}$ from $\boldsymbol{w}$ is closest to $\pi/2$, i.e., most closely perpendicular to $\boldsymbol{w}$ (we write "perpendicular to $\boldsymbol{w}$" as $\perp \boldsymbol{w}$ for convenience). This is very different from traditional point-to-point nearest neighbor search which returns the most similar point to the query point.

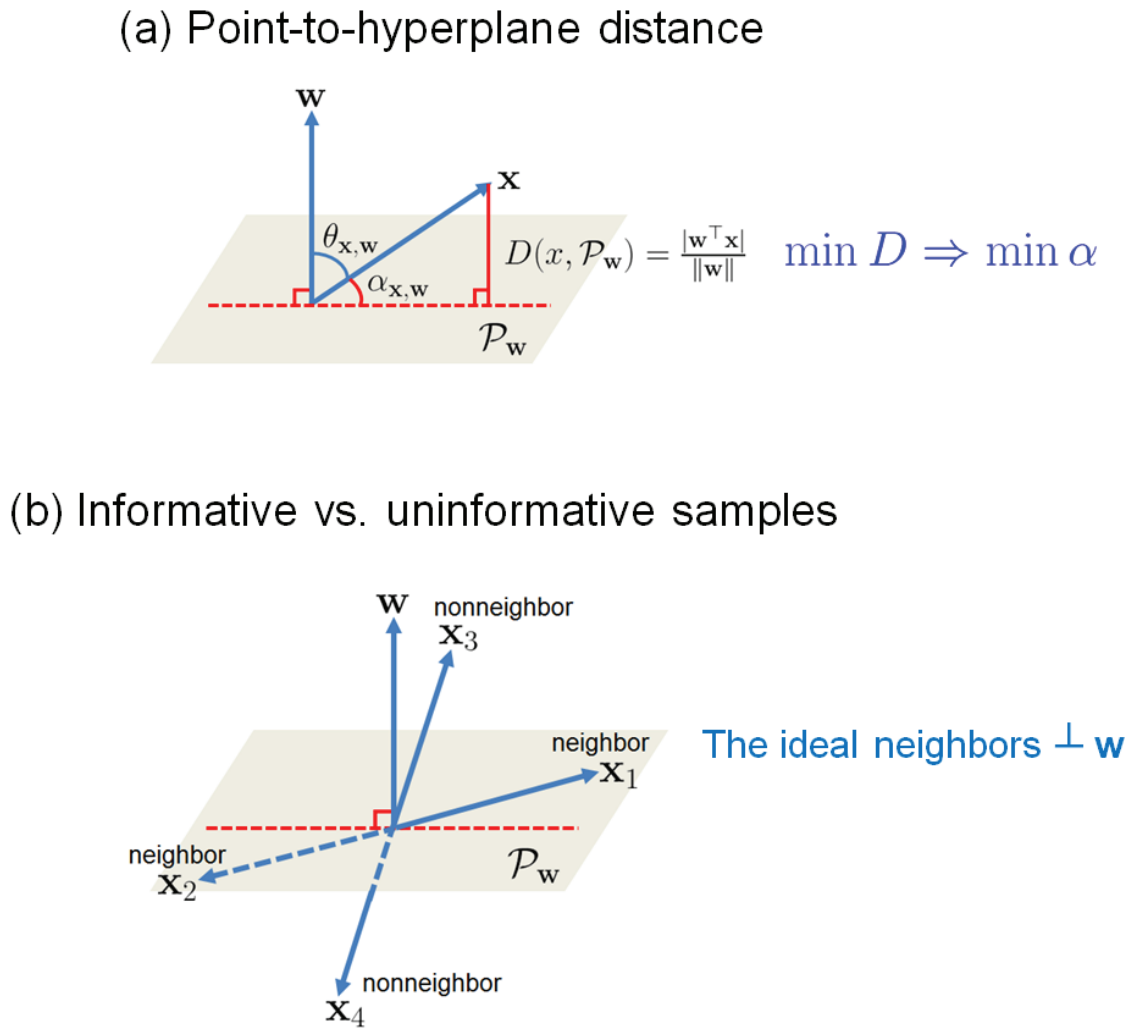


Figure 6.2: The point-to-hyperplane search problem encountered in SVM active learning. $\mathcal{P}_{\boldsymbol{w}}$ is the SVM's hyperplane decision boundary, $\boldsymbol{w}$ is the normal vector to $\mathcal{P}_{\boldsymbol{w}}$, and $\boldsymbol{x}$ is a data vector. (a) Point-to-hyperplane distance $D(\boldsymbol{x}, \mathcal{P}_{\boldsymbol{w}})$ and point-to-hyperplane angle $\alpha_{\boldsymbol{x},\boldsymbol{w}}$. (b) Informative $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ and uninformative $(\boldsymbol{x}_3, \boldsymbol{x}_4)$ samples for the margin-based active sample selection criterion. The ideal neighbors of $\mathcal{P}_{\boldsymbol{w}}$ are the points $\perp \boldsymbol{w}$.

If we regard $|\cos(\theta_{\boldsymbol{x},\boldsymbol{w}})| = \dfrac{|\boldsymbol{w}^\top \boldsymbol{x}|}{\|\boldsymbol{w}\|\|\boldsymbol{x}\|}$ as a similarity measure between $\boldsymbol{x}$ and $\boldsymbol{w}$, hyperplane hashing actually seeks for the most dissimilar point to the query point $\boldsymbol{w}$, which is because

$$\begin{aligned}
\boldsymbol{x}^* &= \arg\min_{\boldsymbol{x}\in\mathcal{X}} \ \alpha_{\boldsymbol{x},\boldsymbol{w}} \\
&= \arg\min_{\boldsymbol{x}\in\mathcal{X}} \ \sin(\alpha_{\boldsymbol{x},\boldsymbol{w}}) \\
&= \arg\min_{\boldsymbol{x}\in\mathcal{X}} \ \sin(\Big|\theta_{\boldsymbol{x},\boldsymbol{w}} - \frac{\pi}{2}\Big|) \\
&= \arg\min_{\boldsymbol{x}\in\mathcal{X}} \ |\cos(\theta_{\boldsymbol{x},\boldsymbol{w}})|. \qquad (6.3)
\end{aligned}$$

On the contrary, the most similar points to $\boldsymbol{w}$, such as $\boldsymbol{w}$, $2\boldsymbol{w}$ and $-\boldsymbol{w}$, are certainly uninformative for the active selection criterion, so they must be excluded.

6.5 Randomized Hyperplane Hashing

In this section, we first briefly review the existing linear function based randomized hyperplane hashing methods, then present our bilinearly formed randomized hashing approach, and finally provide theoretic analysis for the proposed bilinear hash function which is novel in the hashing field and exclusively attacks point-to-hyperplane search.

6.5.1 Background – Linear Hash Functions

Jain *et al.* [75] devised two distinct families of randomized hash functions to attack the hyperplane hashing problem.

The first one is *Angle-Hyperplane Hash* (AH-Hash) $\mathcal{A}$, of which one example is

$$h^{\mathcal{A}}(\boldsymbol{z}) = \begin{cases} [\text{sgn}(\boldsymbol{u}^\top \boldsymbol{z}), \text{sgn}(\boldsymbol{v}^\top \boldsymbol{z})], \boldsymbol{z} \text{ is a database point} \\ [\text{sgn}(\boldsymbol{u}^\top \boldsymbol{z}), \text{sgn}(-\boldsymbol{v}^\top \boldsymbol{z})], \quad \boldsymbol{z} \text{ is a hyperplane normal} \end{cases} \qquad (6.4)$$

where $\boldsymbol{z} \in \mathbb{R}^d$ represents an input vector, and $\boldsymbol{u}$ and $\boldsymbol{v}$ are both drawn independently from a standard d-variate Gaussian, i.e., $\boldsymbol{u}, \boldsymbol{v} \sim \mathcal{N}(0, I_{d\times d})$. Throughout this chapter, the sign function $\text{sgn}(x)$ returns 1 if $x > 0$ and -1 otherwise. In the same treatment as Chapter 4 and 5, we treat '0' bit as '-1' bit for formulation and training, and use '0' bit back in the implementations of data coding and hashing.

Note that $h^{\mathcal{A}}$ is a two-bit hash function which leads to the probability of collision for a hyperplane normal $\boldsymbol{w}$ and a database point $\boldsymbol{x}$:

$$\mathbf{Pr}\left[h^{\mathcal{A}}(\boldsymbol{w}) = h^{\mathcal{A}}(\boldsymbol{x})\right] = \frac{1}{4} - \frac{\alpha_{\boldsymbol{x},\boldsymbol{w}}^2}{\pi^2}. \tag{6.5}$$

The probability monotonically decreases as the point-to-hyperplane angle $\alpha_{\boldsymbol{x},\boldsymbol{w}}$ increases, ensuring angle-sensitive hashing.

The second is *Embedding-Hyperplane Hash* (EH-Hash) function family $\mathcal{E}$ of which one example is

$$h^{\mathcal{E}}(\boldsymbol{z}) = \begin{cases} \operatorname{sgn}\left(\mathbf{U}^\top \mathbf{V}(\boldsymbol{z}\boldsymbol{z}^\top)\right), \boldsymbol{z} \text{ is a database point} \\ \operatorname{sgn}\left(-\mathbf{U}^\top \mathbf{V}(\boldsymbol{z}\boldsymbol{z}^\top)\right), \quad \boldsymbol{z} \text{ is a hyperplane normal} \end{cases} \tag{6.6}$$

where $\mathbf{V}(A)$ returns the vectorial concatenation of matrix A, and $\mathbf{U} \sim \mathcal{N}(0, I_{d^2 \times d^2})$.

In particular, the EH hash function $h^{\mathcal{E}}$ yields hash bits on an embedded space $\mathbb{R}^{d^2}$ resulting from vectorizing rank-one matrices $\boldsymbol{z}\boldsymbol{z}^\top$ and $-\boldsymbol{z}\boldsymbol{z}^\top$. Compared with $h^{\mathcal{A}}$, $h^{\mathcal{E}}$ gives a higher probability of collision for a hyperplane normal $\boldsymbol{w}$ and a database point $\boldsymbol{x}$:

$$\mathbf{Pr}\left[h^{\mathcal{E}}(\boldsymbol{w}) = h^{\mathcal{E}}(\boldsymbol{x})\right] = \frac{\cos^{-1}\sin^2(\alpha_{\boldsymbol{x},\boldsymbol{w}})}{\pi}, \tag{6.7}$$

which also bears the angle-sensitive hashing property. However, it is much more expensive to compute than AH-Hash.

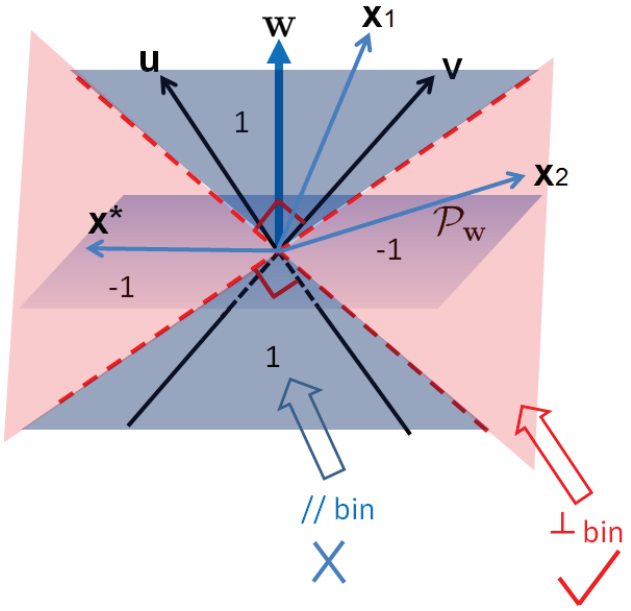
It is important to note that both AH-Hash and EH-Hash are essentially linear hashing techniques. On the contrary, in this work we introduce bilinear hash functions which allow nonlinear hashing.

6.5.2 Bilinear Hash Functions

As a critically novel means of addressing hyperplane hashing, we propose a *bilinear* hash function as follows

$$h^{\mathcal{B}}(\boldsymbol{z}) = \operatorname{sgn}(\boldsymbol{u}^\top \boldsymbol{z}\boldsymbol{z}^\top \boldsymbol{v}), \tag{6.8}$$

where $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d$ are two projection vectors. Our motivation for devising such a bilinear form comes from the following two requirements: 1) $h^{\mathcal{B}}$ should be invariant to the scale of $\boldsymbol{z}$, which is motivated by the fact that $\boldsymbol{z}$ and $\beta\boldsymbol{z}$ ($\beta \neq 0$) hold the same point-to-hyperplane



$$h^{\mathcal{B}}(\mathbf{w}) = \mathrm{sgn}(\mathbf{u}^\top \mathbf{w}\mathbf{w}^\top \mathbf{v}) = \mathrm{sgn}(\mathbf{u}^\top \mathbf{w})\mathrm{sgn}(\mathbf{v}^\top \mathbf{w}) = 1 \cdot 1 = 1$$

$$h^{\mathcal{B}}(\mathbf{x}_1) = \mathrm{sgn}(\mathbf{u}^\top \mathbf{x}_1)\mathrm{sgn}(\mathbf{v}^\top \mathbf{x}_1) = 1 \cdot 1 = 1$$

$$h^{\mathcal{B}}(\mathbf{x}_2) = \mathrm{sgn}(\mathbf{u}^\top \mathbf{x}_2)\mathrm{sgn}(\mathbf{v}^\top \mathbf{x}_2) = -1 \cdot 1 = -1$$

$$h^{\mathcal{B}}(\mathbf{x}^*) = \mathrm{sgn}(\mathbf{u}^\top \mathbf{x}^*)\mathrm{sgn}(\mathbf{v}^\top \mathbf{x}^*) = 1 \cdot -1 = -1$$

Figure 6.3: The single-bit outputs of the proposed bilinear hash function $h^{\mathcal{B}}$. Data vector $\boldsymbol{x}_1$ is nearly parallel to the normal vector $\boldsymbol{w}$, data vector $\boldsymbol{x}_2$ is closely $\perp \boldsymbol{w}$, and data vector $\boldsymbol{x}^* \perp \boldsymbol{w}$ is the ideal neighbor for the goal of angle-based hyperplane hashing. The blue "// bin" is useless but the red "$\perp$" bin is imperative.

angle; and 2) $h^{\mathcal{B}}$ should yield different hash bits for two perpendicular input vectors. The former definitely holds due to the bilinear formulation. We show in Lemma 9 that the latter holds with a constant probability when $\boldsymbol{u}, \boldsymbol{v}$ are drawn independently from the standard normal distribution.

In terms of formulation, the proposed bilinear hash function $h^{\mathcal{B}}$ is correlated with yet different from $h^{\mathcal{A}}$ and $h^{\mathcal{E}}$. (1) $h^{\mathcal{B}}$ produces a single hash bit which is the product of the two hash bits produced by $h^{\mathcal{A}}$. (2) $h^{\mathcal{B}}$ may be a rank-one special case of $h^{\mathcal{E}}$ in algebra if we write $\boldsymbol{u}^\top \boldsymbol{z}\boldsymbol{z}^\top \boldsymbol{v} = \mathrm{tr}(\boldsymbol{z}\boldsymbol{z}^\top \boldsymbol{v}\boldsymbol{u}^\top)$ and $\mathbf{U}^\top \mathbf{V}(\boldsymbol{z}\boldsymbol{z}^\top) = \mathrm{tr}(\boldsymbol{z}\boldsymbol{z}^\top \mathbf{U})$. (3) $h^{\mathcal{B}}$ appears in a universal form, while both $h^{\mathcal{A}}$ and $h^{\mathcal{E}}$ treat a query and a database item in a distinct manner. The computation time of $h^{\mathcal{B}}$ is $\Theta(2d)$ which is the same as that of $h^{\mathcal{A}}$ and one order of magnitude faster than $\Theta(2d^2)$ of $h^{\mathcal{E}}$. Even though [75] developed a random subsampling trick to speed up expensive inner product operations in $\mathbb{R}^{d^2}$ required by $h^{\mathcal{E}}$, a high expense of $\Theta(d^2)+O(d)$ is still needed.

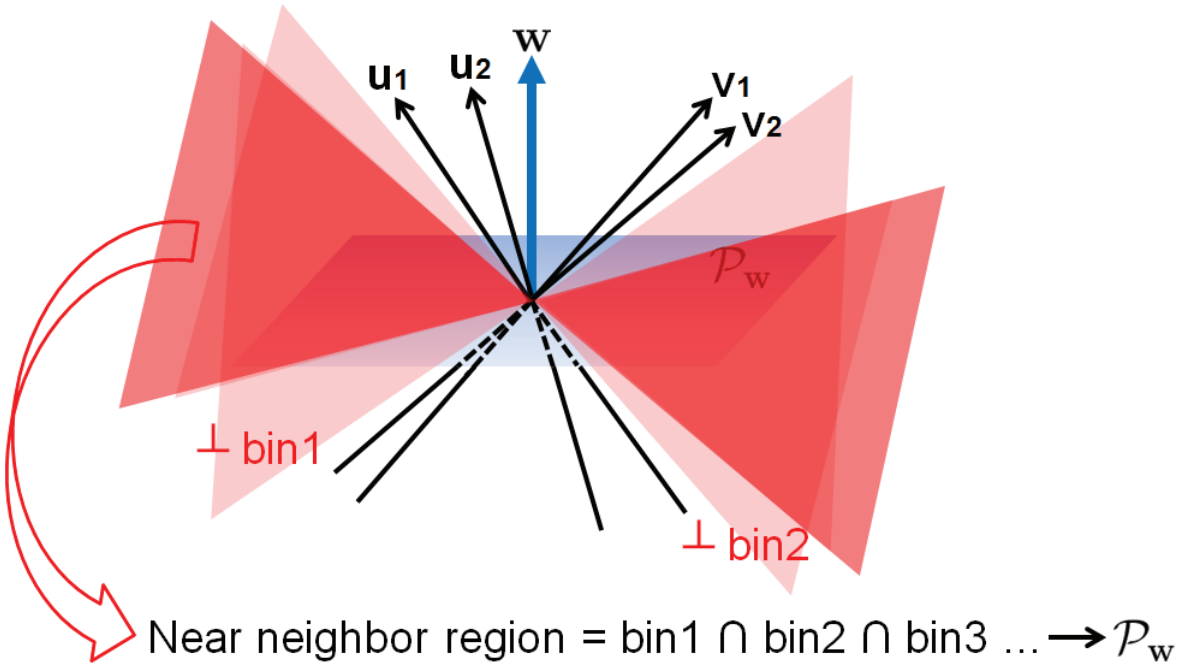


Figure 6.4: Enforcing multiple hash bits to refine the near neighbor region for point-to-hyperplane search. $(\boldsymbol{u}_1, \boldsymbol{v}_1)$ and $(\boldsymbol{u}_2, \boldsymbol{v}_2)$ are projection pairs used by two bilinear hash functions $h_1^{\mathcal{B}}$ and $h_2^{\mathcal{B}}$, respectively.

Figure 6.3 shows the single-bit outputs of the proposed bilinear hash function $h^{\mathcal{B}}$. By employing valid projection directions $\boldsymbol{u}, \boldsymbol{v}$ to design a good bilinear hash function, $h^{\mathcal{B}}$ essentially partitions the whole space $\mathbb{R}^d$ into two cone bins, as displayed in Figure 6.3. The // bin that contains uninformative data points nearly parallel to the query normal $\boldsymbol{w}$ must be excluded under nearest neighbor consideration, while the $\perp$ bin that contains informative data points nearly perpendicular to the normal $\boldsymbol{w}$ is exactly what we need. As far as the produced hash bits are concerned, $h^{\mathcal{B}}$ outputs the same bit for the query normal $\boldsymbol{w}$ and the database points in the // bin (e.g., $h^{\mathcal{B}}(\boldsymbol{w}) = h^{\mathcal{B}}(\boldsymbol{x}_1) = 1$), but different bits for $\boldsymbol{w}$ and the points in the $\perp$ bin (e.g., $h^{\mathcal{B}}(\boldsymbol{w}) = -h^{\mathcal{B}}(\boldsymbol{x}_2) = -h^{\mathcal{B}}(\boldsymbol{x}^*)$).

If a series of hash functions were used to generate more bits, we would obtain a much more narrow $\perp$ bin which is the intersection of multiple $\perp$ bins each of which is determined by a single hash function. Consequently, the search region for candidate near neighbors, that are supposed to hold small point-to-hyperplane angles to the query hyperplane $\mathcal{P}_{\boldsymbol{w}}$, becomes smaller, more accurate, and will converge to $\mathcal{P}_{\boldsymbol{w}}$ itself when more and more hash bits are exploited. Figure 6.4 reveals the effect of enforcing multiple hash bits.

To summarize, for the purpose of angle-based hyperplane hashing described in Section 6.4, the pivotal role of bilinear hash functions is to map the query point $\boldsymbol{w}$ (the hyperplane normal) and the desirable most informative point (with $\theta_{\boldsymbol{x},\boldsymbol{w}} = \pi/2$) to bitwise different hash codes, whereas map $\boldsymbol{w}$ and the undesirable most uninformative point (with $\theta_{\boldsymbol{x},\boldsymbol{w}} = 0$ or π) to identical hash codes. Therefore, hyperplane hashing using our proposed bilinear hash functions is implemented as finding the database points in $\mathcal{X}$ whose hash codes have the farthest Hamming distances to the hash code of the query $\boldsymbol{w}$. This is very different from traditional hashing methods as well as the existing hyperplane hashing methods AH-Hash and BH-Hash, all of which always seek the closest codes to the query's code in the Hamming space.

6.5.3 Theoretic Analysis

Based on the bilinear formulation in eq. (6.8), we define a novel randomized function family *Bilinear-Hyperplane Hash* (BH-Hash) as:

$$\mathcal{B} = \left\{ h^{\mathcal{B}}(\boldsymbol{z}) = \operatorname{sgn}(\boldsymbol{u}^\top \boldsymbol{z}\boldsymbol{z}^\top \boldsymbol{v}),\ \text{i.i.d. } \boldsymbol{u}, \boldsymbol{v} \sim \mathcal{N}(0, I_{d\times d}) \right\}. \tag{6.9}$$

Here we prove several key characteristics of $\mathcal{B}$. Specially, we define $h^{\mathcal{B}}(\mathcal{P}_{\boldsymbol{w}}) = -h^{\mathcal{B}}(\boldsymbol{w})$ for an easy derivation.

Lemma 9. *Given a hyperplane query $\mathcal{P}_{\boldsymbol{w}}$ with the normal vector $\boldsymbol{w} \in \mathbb{R}^d$ and a database point $\boldsymbol{x} \in \mathbb{R}^d$, the probability of collision for $\mathcal{P}_{\boldsymbol{w}}$ and $\boldsymbol{x}$ under $h^{\mathcal{B}}$ is*

$$\mathbf{Pr}\left[h^{\mathcal{B}}(\mathcal{P}_{\boldsymbol{w}}) = h^{\mathcal{B}}(\boldsymbol{x})\right] = \frac{1}{2} - \frac{2\alpha^2_{\boldsymbol{x},\boldsymbol{w}}}{\pi^2}. \tag{6.10}$$

Proof. This probability is equal to the probability of $h^{\mathcal{B}}(\boldsymbol{w}) \neq h^{\mathcal{B}}(\boldsymbol{x})$. Because the two random projections $\boldsymbol{u}$ and $\boldsymbol{v}$ are independent,

$$\begin{aligned}\mathbf{Pr}\left[h^{\mathcal{B}}(\boldsymbol{w}) \neq h^{\mathcal{B}}(\boldsymbol{x})\right] = &\mathbf{Pr}\left[\operatorname{sgn}(\boldsymbol{u}^\top \boldsymbol{w}) = \operatorname{sgn}(\boldsymbol{u}^\top \boldsymbol{x})\right] \mathbf{Pr}\left[\operatorname{sgn}(\boldsymbol{v}^\top \boldsymbol{w}) \neq \operatorname{sgn}(\boldsymbol{v}^\top \boldsymbol{x})\right] \\ &+ \mathbf{Pr}\left[\operatorname{sgn}(\boldsymbol{u}^\top \boldsymbol{w}) \neq \operatorname{sgn}(\boldsymbol{u}^\top \boldsymbol{x})\right] \mathbf{Pr}\left[\operatorname{sgn}(\boldsymbol{v}^\top \boldsymbol{w}) = \operatorname{sgn}(\boldsymbol{v}^\top \boldsymbol{x})\right].\end{aligned}$$

By exploiting the fact $\mathbf{Pr}\left[\operatorname{sgn}(\boldsymbol{u}^\top \boldsymbol{z}) = \operatorname{sgn}(\boldsymbol{u}^\top \boldsymbol{z}')\right] = 1 - \theta_{\boldsymbol{z},\boldsymbol{z}'}/\pi$ from [53],

$$\begin{aligned}&\mathbf{Pr}\left[h^{\mathcal{B}}(\boldsymbol{w}) \neq h^{\mathcal{B}}(\boldsymbol{x})\right] \\ =&\left(1 - \frac{\theta_{\boldsymbol{x},\boldsymbol{w}}}{\pi}\right)\frac{\theta_{\boldsymbol{x},\boldsymbol{w}}}{\pi} + \frac{\theta_{\boldsymbol{x},\boldsymbol{w}}}{\pi}\left(1 - \frac{\theta_{\boldsymbol{x},\boldsymbol{w}}}{\pi}\right) \\ =&2\left(\frac{1}{2} - \frac{\theta_{\boldsymbol{x},\boldsymbol{w}} - \frac{\pi}{2}}{\pi}\right)\left(\frac{1}{2} + \frac{\theta_{\boldsymbol{x},\boldsymbol{w}} - \frac{\pi}{2}}{\pi}\right) \\ =&\frac{1}{2} - \frac{2(\theta_{\boldsymbol{x},\boldsymbol{w}} - \frac{\pi}{2})^2}{\pi^2} \\ =&\frac{1}{2} - \frac{2\alpha^2_{\boldsymbol{x},\boldsymbol{w}}}{\pi^2},\end{aligned} \tag{6.11}$$

which completes the proof. □

Lemma 9 shows that the probability of $h^{\mathcal{B}}(\boldsymbol{w}) \neq h^{\mathcal{B}}(\boldsymbol{x})$ is 1/2 for perpendicular $\boldsymbol{w}$ and $\boldsymbol{x}$ that hold $\theta_{\boldsymbol{x},\boldsymbol{w}} = \pi/2$ (accordingly $\alpha_{\boldsymbol{x},\boldsymbol{w}} = 0$). It is important to realize that this collision probability is twice of that from the linear AH hash function $h^{\mathcal{A}}$ described in Section 6.5.1.

Theorem 10. *The BH-Hash function family $\mathcal{B}$ is $\left(r, r(1+\epsilon), \frac{1}{2} - \frac{2r}{\pi^2}, \frac{1}{2} - \frac{2r(1+\epsilon)}{\pi^2}\right)$-sensitive to the distance measure $\mathcal{D}(\boldsymbol{x}, \mathcal{P}_{\boldsymbol{w}}) = \alpha^2_{\boldsymbol{x},\boldsymbol{w}}$, where $r, \epsilon > 0$.*

Proof. By utilizing Lemma 9, for any $h^{\mathcal{B}} \in \mathcal{B}$, when $\mathcal{D}(\boldsymbol{x}, \mathcal{P}_{\boldsymbol{w}}) \leq r$ we have

$$\begin{aligned} \mathbf{Pr}\left[h^{\mathcal{B}}(\mathcal{P}_{\boldsymbol{w}}) = h^{\mathcal{B}}(\boldsymbol{x})\right] &= \frac{1}{2} - \frac{2\mathcal{D}(\boldsymbol{x}, \mathcal{P}_{\boldsymbol{w}})}{\pi^2} \\ &\geq \frac{1}{2} - \frac{2r}{\pi^2} = p_1. \end{aligned} \tag{6.12}$$

Likewise, when $\mathcal{D}(\boldsymbol{x}, \mathcal{P}_{\boldsymbol{w}}) > r(1+\epsilon)$ we have

$$\begin{aligned} \mathbf{Pr}\left[h^{\mathcal{B}}(\mathcal{P}_{\boldsymbol{w}}) = h^{\mathcal{B}}(\boldsymbol{x})\right] &= \frac{1}{2} - \frac{2\mathcal{D}(\boldsymbol{x}, \mathcal{P}_{\boldsymbol{w}})}{\pi^2} \\ &< \frac{1}{2} - \frac{2r(1+\epsilon)}{\pi^2} = p_2. \end{aligned} \tag{6.13}$$

This completes the proof. □

Note that p_1, p_2 ($p_1 > p_2$) depend on $0 \leq r \leq \pi^2/4$ and $\epsilon > 0$. We present the following theorem by adapting Theorem 1 in [52] and Theorem 0.1 in the supplementary material of [75].

Theorem 11. *Suppose we have a database $\mathcal{X}$ of n points. Denote the parameters $k = \log_{1/p_2} n$, $\rho = \frac{\ln p_1}{\ln p_2}$, and $c \geq 2$. Given a hyperplane query $\mathcal{P}_{\boldsymbol{w}}$, if there exists a database point $\boldsymbol{x}^*$ such that $\mathcal{D}(\boldsymbol{x}^*, \mathcal{P}_{\boldsymbol{w}}) \leq r$, then a BH-Hash algorithm is able to return a database point $\hat{\boldsymbol{x}}$ such that $\mathcal{D}(\hat{\boldsymbol{x}}, \mathcal{P}_{\boldsymbol{w}}) \leq r(1+\epsilon)$ with a probability at least $1 - \frac{1}{c} - \frac{1}{e}$ by using n^ρ hash tables of k hash bits each. The query time is dominated by $O(n^\rho \log_{1/p_2} n)$ evaluations of the hash functions from $\mathcal{B}$ and cn^ρ computations of the pairwise distances $\mathcal{D}$ between $\mathcal{P}_w$ and the points hashed into the same buckets.*

Proof. This theorem introduces the asymptotic computational complexity of our proposed randomized bilinear hashing scheme BH-Hash. For self-contained consideration, here we

provide the proof, which is basically following the technique previously used in the proof of Theorem 1 in [52]. Recall that in Theorem 10, we have shown that BH-Hash is $(r, r(1+\epsilon), p_1, p_2)$-sensitive. Particularly, we show that $p_1 = \frac{1}{2} - \frac{2r}{\pi^2}$ and $p_2 = \frac{1}{2} - \frac{2r(1+\epsilon)}{\pi^2}$ for BH-Hash.

Denote the number of hash tables by L. For the l-th hash table, a BH-Hash algorithm randomly samples k hash functions $h^{\mathcal{B}}_{l,1}, \cdots, h^{\mathcal{B}}_{l,k}$ with replacement from $\mathcal{B}$, which will generate a k-bit hash key for each input data vector $\boldsymbol{x}$. We denote $\boldsymbol{x}$'s hash code by $H^{\mathcal{B}}_l(\boldsymbol{x}) = [h^{\mathcal{B}}_{l,1}(\boldsymbol{x}), \cdots, h^{\mathcal{B}}_{l,k}(\boldsymbol{x})]$. The main observation is that using $L = n^\rho$ independent hash tables, a $(1+\epsilon)$-approximate nearest neighbor is achieved with a non-trivial constant probability. Moreover, the query (search) time complexity is proved to be sub-linear with respect to the entire data size n.

To complete the proof, we define the following two events $\mathbf{F}_1$ and $\mathbf{F}_2$. It suffices to prove the theorem by showing that both $\mathbf{F}_1$ and $\mathbf{F}_2$ hold with a probability larger than 0.5. The two events are defined as below:

$\mathbf{F}_1$: If there exists a database point $\boldsymbol{x}^*$ such that $\mathcal{D}(\boldsymbol{x}^*, \mathcal{P}_{\boldsymbol{w}}) \leq r$, then $H^{\mathcal{B}}_l(\boldsymbol{x}^*) = H^{\mathcal{B}}_l(\mathcal{P}_{\boldsymbol{w}})$ for some $1 \leq l \leq L$.

$\mathbf{F}_2$: Provided with a false alarm set

$$\mathcal{S} = \left\{ \check{\boldsymbol{x}} \;\middle|\; \check{\boldsymbol{x}} \in \mathcal{X} \text{ such that } \mathcal{D}(\check{\boldsymbol{x}}, \mathcal{P}_{\boldsymbol{w}}) > r(1+\epsilon) \text{ and } \exists l \in [1:L],\ H^{\mathcal{B}}_l(\check{\boldsymbol{x}}) = H^{\mathcal{B}}_l(\mathcal{P}_{\boldsymbol{w}}) \right\}$$

in which $\epsilon > 0$ is the given small constant, then the set cardinality $|\mathcal{S}| < cL$.

First, we prove that $\mathbf{F}_1$ **holds with a probability at least** $1 - \frac{1}{e}$.

Let us consider the converse case that $H^{\mathcal{B}}_l(\boldsymbol{x}^*) \neq H^{\mathcal{B}}_l(\mathcal{P}_{\boldsymbol{w}})$ for $\forall l \in [1:L]$ whose probability is

$$\begin{aligned}
&\mathbf{Pr}\left[H^{\mathcal{B}}_l(\boldsymbol{x}^*) \neq H^{\mathcal{B}}_l(\mathcal{P}_{\boldsymbol{w}}), \forall l \in [1:L]\right] \\
=&\left(\mathbf{Pr}\left[H^{\mathcal{B}}_l(\boldsymbol{x}^*) \neq H^{\mathcal{B}}_l(\mathcal{P}_{\boldsymbol{w}})\right]\right)^L \\
=&\left(1 - \mathbf{Pr}\left[H^{\mathcal{B}}_l(\boldsymbol{x}^*) = H^{\mathcal{B}}_l(\mathcal{P}_{\boldsymbol{w}})\right]\right)^L \\
\leq&\left(1 - p_1^k\right)^L = \left(1 - p_1^{\log_{\frac{1}{p_2}} n}\right)^{n^\rho} \\
=&\left(1 - n^{-\rho}\right)^{n^\rho} = \left(\left(1 - n^{-\rho}\right)^{-n^\rho}\right)^{-1} \\
\leq&\frac{1}{e},
\end{aligned} \tag{6.14}$$

where Inequality (6.14) follows from the inequality $(1-n^{-\rho})^{-n^{\rho}} \geq e$. Herewith we derive

$$\begin{aligned}
&\mathbf{Pr}\left[H_l^{\mathcal{B}}(\boldsymbol{x}^*) = H_l^{\mathcal{B}}(\mathcal{P}_{\boldsymbol{w}}), \exists l \in [1:L]\right] \\
=&1 - \mathbf{Pr}\left[H_l^{\mathcal{B}}(\boldsymbol{x}^*) \neq H_l^{\mathcal{B}}(\mathcal{P}_{\boldsymbol{w}}), \forall l \in [1:L]\right] \\
\geq&1 - \frac{1}{e}.
\end{aligned}$$

Second, we prove that $\mathbf{F}_2$ **holds with a probability at least** $1 - \frac{1}{c}$.

For every false alarm point $\check{\boldsymbol{x}}$ conforming to $\mathcal{D}(\check{\boldsymbol{x}}, \mathcal{P}_{\boldsymbol{w}}) > r(1+\epsilon)$, in any hash table $l \in [1:L]$ we have

$$\begin{aligned}
&\mathbf{Pr}\left[H_l^{\mathcal{B}}(\check{\boldsymbol{x}}) = H_l^{\mathcal{B}}(\mathcal{P}_{\boldsymbol{w}})\right] \\
<&(p_2)^k = (p_2)^{\log_{\frac{1}{p_2}} n} \\
=&\frac{1}{n}.
\end{aligned}$$

Therefore the expected number of false alarm points, which fall into the same hash bucket with the query $\mathcal{P}_{\boldsymbol{w}}$ in hash table l, is smaller than $n \times 1/n = 1$. Immediately, we conclude $\mathbf{E}\left[|\mathcal{S}|\right] < L$. Subsequently, we further apply Markov's inequality to derive the following result:

$$\mathbf{Pr}\left[|\mathcal{S}| \geq cL\right] \leq \frac{\mathbf{E}\left[|\mathcal{S}|\right]}{cL} < \frac{L}{cL} = \frac{1}{c},$$

which leads to

$$\mathbf{Pr}\left[|\mathcal{S}| < cL\right] = 1 - \mathbf{Pr}\left[|\mathcal{S}| \geq cL\right] > 1 - \frac{1}{c}.$$

Third, we prove that $\mathbf{F}_1$ **and** $\mathbf{F}_2$ **simultaneously hold with a probability at least** $1 - \frac{1}{c} - \frac{1}{e}$.

Let us deduce the conditional probability $\mathbf{Pr}\left[\overline{\mathbf{F}}_2 | \mathbf{F}_1\right]$ as follows

$$\begin{aligned}
\mathbf{Pr}\left[\overline{\mathbf{F}}_2 | \mathbf{F}_1\right] &= \frac{\mathbf{Pr}\left[\overline{\mathbf{F}}_2 \bigcap \mathbf{F}_1\right]}{\mathbf{Pr}\left[\mathbf{F}_1\right]} \\
&\leq \frac{\mathbf{Pr}\left[\overline{\mathbf{F}}_2\right]}{\mathbf{Pr}\left[\mathbf{F}_1\right]} = \frac{1 - \mathbf{Pr}\left[\mathbf{F}_2\right]}{\mathbf{Pr}\left[\mathbf{F}_1\right]} \\
&< \frac{1 - (1 - \frac{1}{c})}{\mathbf{Pr}\left[\mathbf{F}_1\right]} \\
&= \frac{1}{c\mathbf{Pr}\left[\mathbf{F}_1\right]}.
\end{aligned}$$

Then, we derive

$$\begin{aligned}\mathbf{Pr}\left[\mathbf{F}_1 \bigcap \mathbf{F}_2\right] &= \mathbf{Pr}\left[\mathbf{F}_2|\mathbf{F}_1\right]\mathbf{Pr}\left[\mathbf{F}_1\right] \\ &= (1 - \mathbf{Pr}\left[\overline{\mathbf{F}}_2|\mathbf{F}_1\right])\mathbf{Pr}\left[\mathbf{F}_1\right] \\ &> (1 - \frac{1}{c\mathbf{Pr}\left[\mathbf{F}_1\right]})\mathbf{Pr}\left[\mathbf{F}_1\right] \\ &= \mathbf{Pr}\left[\mathbf{F}_1\right] - \frac{1}{c} \\ &\geq 1 - \frac{1}{c} - \frac{1}{e}.\end{aligned}$$

To sum up, the inequality $\mathbf{Pr}\left[\mathbf{F}_1 \bigcap \mathbf{F}_2\right] > 1 - \frac{1}{c} - \frac{1}{e}$ uncovers that with a constant probability larger than $1 - \frac{1}{c} - \frac{1}{e}$, the BH-Hash algorithm is guaranteed to return at least a database point $\hat{\boldsymbol{x}}$ conforming to $\mathcal{D}(\hat{\boldsymbol{x}}, \mathcal{P}_{\boldsymbol{w}}) \leq r(1+\epsilon)$ by checking the first cL database points that collide to the query $\mathcal{P}_{\boldsymbol{w}}$ in either of the L hash tables. The algorithm terminates when cL database points have been scanned. Since at most L hash tables are visited, at most $L \cdot k = n^{\rho} \log_{1/p_2} n$ hash functions are evaluated. Accordingly, at most $cL = cn^{\rho}$ computations of the distance function $\mathcal{D}$ are needed to confirm the $(1+\epsilon)$-approximate nearest neighbors. Complete the proof. □

Theorem 11 indicates that the query time of a BH-Hash algorithm is essentially bounded by $O(n^{\rho})$, in which the exponent $0 < \rho < 1$ relies on both r and ϵ. In fact, it can be simplified under additional assumptions. We present the following Corollary.

Corollary 12. *If $r \geq \gamma\pi^2/4$ with $0 < \gamma < 1$, then the query time of a BH-Hash algorithm is bounded by $O\left(n^{\frac{\ln(2/(1-\gamma))}{\ln 2 + \gamma(1+\epsilon)}}\right)$.*

Proof. Given a fixed $\epsilon > 0$,

$$\rho = \frac{\ln p_1}{\ln p_2} = \frac{\ln\left(\frac{1}{2} - \frac{2r}{\pi^2}\right)}{\ln\left(\frac{1}{2} - \frac{2r(1+\epsilon)}{\pi^2}\right)}, \tag{6.15}$$

which can be proved to be a monotonically decreasing function with respect to the variable r. Then, if $r \geq \gamma\pi^2/4$,

$$\rho \leq \frac{\ln\left(\frac{1}{2} - \frac{\gamma}{2}\right)}{\ln\left(\frac{1}{2} - \frac{\gamma(1+\epsilon)}{2}\right)} = \frac{\ln\left(\frac{1-\gamma}{2}\right)}{\ln(1 - \gamma(1+\epsilon)) - \ln 2} \leq \frac{\ln\left(\frac{1-\gamma}{2}\right)}{-\gamma(1+\epsilon) - \ln 2} = \frac{\ln\left(\frac{2}{1-\gamma}\right)}{\ln 2 + \gamma(1+\epsilon)},$$

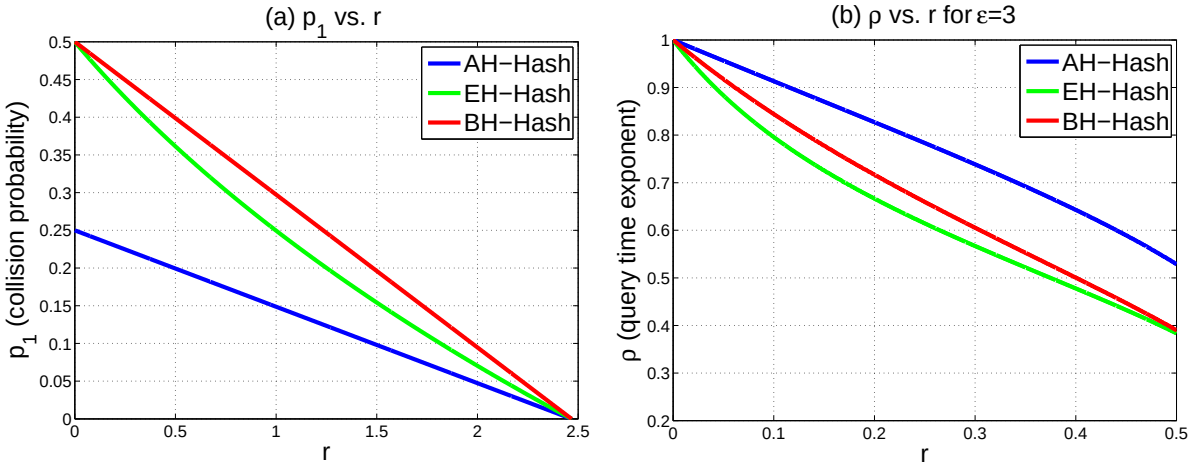


Figure 6.5: Theoretical comparison of three randomized hash schemes. (a) p_1 (probability of collision) vs. r (squared point-to-hyperplane angle); (b) ρ (query time exponent) vs. r for $\epsilon = 3$.

where we use the inequality $\ln(1-\gamma(1+\epsilon)) \leq -\gamma(1+\epsilon)$.

Thus the query time bound $O(n^\rho)$ does not exceed $O\left(n^{\frac{\ln(2/(1-\gamma))}{\ln 2+\gamma(1+\epsilon)}}\right)$. It completes the proof. □

Theorem 11 implies that if $c \geq 8$ the probability of a "successful" sub-linear time scan, that guarantees to find a $(1+\epsilon)$-approximate nearest neighbor, is above 0.5. All of the query time bounds of AH-Hash, EH-Hash and BH-Hash are $O(n^\rho)$, yet with different parameters ρ. To compare them deeply, for each of the three hash schemes we plot the collision probability p_1 and the query time exponent ρ under $\epsilon = 3$ with varying r in Figures 6.5(a) and (b), respectively.

Figure 6.5(a) shows that at any fixed r BH-Hash accomplishes the highest probability of collision, which is twice p_1 of AH-Hash. Though BH-Hash has slightly bigger ρ than EH-Hash displayed in Figure 6.5(b), it has much faster hash function computation $\Theta(2dk)$ than $\Theta(d^2(k+1))$ of EH-Hash per hash table for each query or database point. Note that the accelerated version of EH-Hash can reduce to $\Theta(d^2) + O(dk)$ time but inevitably loses theoretic guarantees. Even the reduced time complexity of EH-Hash is expensive for high-dimensional data since it is still in quadratic dependence on the dimension d. In contrast, both of AH-Hash and our proposed BH-Hash are very efficient, only linearly dependent on

Table 6.3: Several key properties of three randomized hyperplane hashing methods AH-Hash, EH-Hash and BH-Hash. r denotes the squared point-to-hyperplane angle $\alpha^2_{\boldsymbol{x},\boldsymbol{w}}$, k denotes the number of hash bits used in a single hash table, and d denotes the data dimension. "Hashing Time" refers to the time of evaluating k hash functions to hash an input (database point or query) into a single hash table.

Method	**Collision Probability** p_1	**Hashing Time**	**Search Strategy**
AH-Hash [75]	$\frac{1}{4} - \frac{r}{\pi^2}$	$\Theta(2dk)$	nearest codes
EH-Hash [75]	$\frac{\cos^{-1}\sin^2(\sqrt{r})}{\pi}$	$\Theta(d^2(k+1))$	nearest codes
BH-Hash (this chapter)	$\frac{1}{2} - \frac{2r}{\pi^2}$	$\Theta(2dk)$	farthest codes

the data dimension. We list several key characteristics of the three hash schemes in Table 6.3 for direct comparison.

It is interesting to see that AH-Hash and our proposed BH-Hash have a tight connection in the style of hashing database points. BH-Hash actually performs the XNOR operation over the two bits that AH-Hash outputs, returning a composite single bit. As a relevant reference, the idea of applying the XOR operation over binary bits in constructing hash functions has ever been used in [106]. However, this is only suitable for the limited data type, discrete sets (e.g., bag-of-words representation), and still falls into the point-to-point search scenario.

6.6 Compact Hyperplane Hashing

6.6.1 Motivations

Despite the higher collision probability of the proposed BH-Hash than AH-Hash and EH-Hash, it is still a randomized approach. The use of random projections in $h^{\mathcal{B}}$ has three potential issues. (i) The probability of colliding for parallel $\mathcal{P}_{\boldsymbol{w}}$ and $\boldsymbol{x}$ with $\alpha_{\boldsymbol{x},\boldsymbol{w}} = 0$ is not too high and only 1/2 according to Lemma 9. (ii) The hashing time is sub-linear $O(n^{\rho}\log_{1/p_2} n)$ in order to bound the approximation error of the retrieved neighbors, as presented in Theorem 11. (iii) When the target nearest neighbor has very tiny point-to-hyperplane angle (i.e., very small r), the sub-linear query time $O(n^{\rho})$ will degenerate to the

brute-force linear scan $O(n)$ because $\lim_{r\to 0}\rho \to 1$, as disclosed by eq. (6.15) and Figure 6.5(b).

AH-Hash and EH-Hash also suffer from the three issues above. Even though these randomized hyperplane hashing methods maintain bounded approximation errors on retrieved neighbors, they require long hash codes and plenty (even hundreds) of hash tables to cater for the accuracy guarantees. Hence, these solutions have tremendous computational and storage costs which limit practical performance of hyperplane hashing on massive databases. In addition, the time spent in generating hash codes also grows linearly with the numbers of hash bits and hash tables.

To this end, we develop a *Compact Hyperplane Hashing* approach to further enhance the power of bilinear hash functions such that, instead of being random, the bilinear projections used in hash functions are learned from input data. Such learning yields compact yet discriminative codes which index the data into a single hash table, leading to substantially reduced computational and storage needs.

6.6.2 Learning Bilinear Hash Functions

Following the notion of *learning to hash* having executed in Chapter 4 and 5, we aim at learning a group of bilinear hash functions $\{h_j\}$ to yield compact binary codes. Note that h_j is different from the randomized bilinear hash function $h_j^{\mathcal{B}}$, and that we consistently define $h_j(\mathcal{P}_{\boldsymbol{w}}) = -h_j(\boldsymbol{w})$. We would like to learn h_j such that smaller $\alpha_{\boldsymbol{x},\boldsymbol{w}}$ results in larger $h_j(\mathcal{P}_{\boldsymbol{w}})h_j(\boldsymbol{x})$. Hence, we make $h_j(\mathcal{P}_{\boldsymbol{w}})h_j(\boldsymbol{x})$ to monotonically decrease as $\alpha_{\boldsymbol{x},\boldsymbol{w}}$ increases. This is equivalent to the requirement that $h_j(\boldsymbol{w})h_j(\boldsymbol{x})$ monotonically increases with increasing $\sin(\alpha_{\boldsymbol{x},\boldsymbol{w}}) = |\cos(\theta_{\boldsymbol{x},\boldsymbol{w}})|$.

Suppose that k hash functions are learned to produce k-bit codes. We develop a hash function learning approach with the goal that $\sum_{j=1}^{k} h_j(\boldsymbol{w})h_j(\boldsymbol{x})/k \propto |\cos(\theta_{\boldsymbol{x},\boldsymbol{w}})|$. Further, since $\sum_{j=1}^{k} h_j(\boldsymbol{w})h_j(\boldsymbol{x})/k \in [-1, 1]$ and $|\cos(\theta_{\boldsymbol{x},\boldsymbol{w}})| \in [0, 1]$, we specify the learning goal as

$$\frac{1}{k}\sum_{j=1}^{k} h_j(\boldsymbol{w})h_j(\boldsymbol{x}) = 2|\cos(\theta_{\boldsymbol{x},\boldsymbol{w}})| - 1, \tag{6.16}$$

which makes sense since $\theta_{\boldsymbol{x},\boldsymbol{w}} = \pi/2$ ($\alpha_{\boldsymbol{x},\boldsymbol{w}} = 0$) causes $h_j(\boldsymbol{w}) \neq h_j(\boldsymbol{x})$ or $h_j(\mathcal{P}_{\boldsymbol{w}}) = h_j(\boldsymbol{x})$ for any $j \in [1 : k]$. As such, the developed learning method achieves explicit collision for

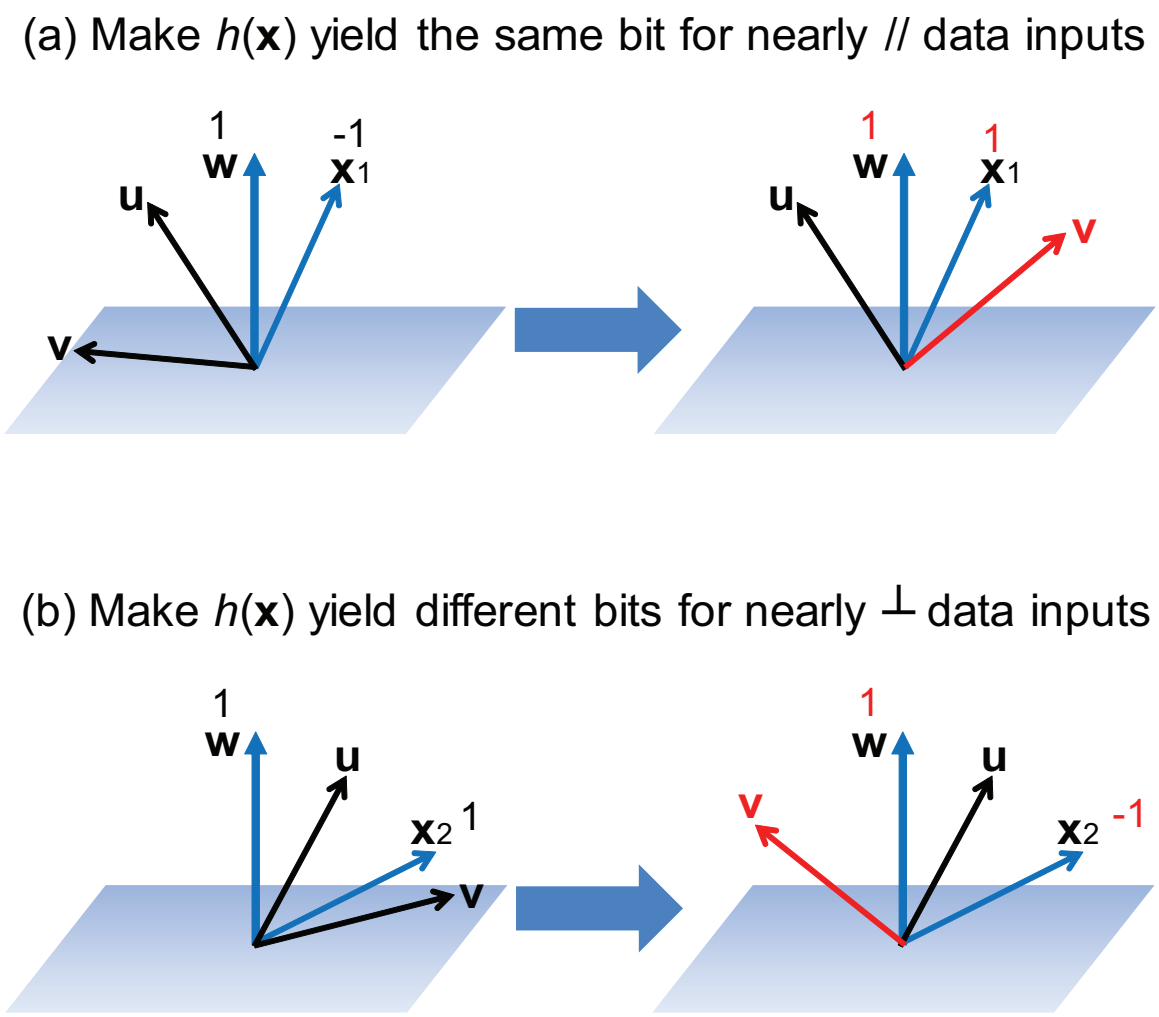


Figure 6.6: Learning the bilinear projections $(\boldsymbol{u}, \boldsymbol{v})$ such that the resulting hash function $h(\boldsymbol{x}) = \text{sgn}(\boldsymbol{u}^\top \boldsymbol{x} \boldsymbol{x}^\top \boldsymbol{v})$ yields the same bit for nearly // inputs whereas different bits for nearly $\perp$ inputs.

parallel $\mathcal{P}_{\boldsymbol{w}}$ and $\boldsymbol{x}$.

Enforcing eq. (6.16) tends to make $\{h_j\}_{j=1}^k$ to yield identical hash codes for nearly parallel inputs whereas bitwise different hash codes for nearly perpendicular inputs. Figure 6.6 shows the spirit of learning one bilinear hash function $h(\boldsymbol{x})$ to satisfy the discriminative requirement imposed on output bits.

At the query time, given a hyperplane query $\mathcal{P}_{\boldsymbol{w}}$, we first extract its k-bit hash code using the k learned hash functions applied to the hyperplane normal vector $\boldsymbol{w}$. Then, the database points whose codes have the largest Hamming distances to the query's code are returned. Thus, the returned points, called *near-to-hyperplane neighbors*, maintain small angles to the hyperplane because such points and the hyperplane normal are nearly perpendicular. In our learning setting, k is typically very short, no more than 30, so we can retrieve the desirable near-to-hyperplane neighbors via constant time hashing over a single hash table.

Now we describe how we learn k pairs of projections $(\boldsymbol{u}_j, \boldsymbol{v}_j)_{j=1}^k$ so as to construct k bilinear hash functions $\{h_j(\boldsymbol{z}) = \operatorname{sgn}(\boldsymbol{u}_j^\top \boldsymbol{z}\boldsymbol{z}^\top \boldsymbol{v}_j)\}_{j=1}^k$. Since the hyperplane normal vectors come up only during the query time, we cannot access $\boldsymbol{w}$ during the training stage. Instead, we sample a few database points for learning projections. Without the loss of generality, we assume that the first l ($k < l \ll n$) samples saved in the matrix $X_l = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_l]$ are used for learning. To capture the pairwise relationships among them, we define a matrix $S \in \mathbb{R}^{l \times l}$ as

$$S_{ii'} = \begin{cases} 1, & |\cos(\theta_{\boldsymbol{x}_i, \boldsymbol{x}_{i'}})| \geq t_1 \\ -1, & |\cos(\theta_{\boldsymbol{x}_i, \boldsymbol{x}_{i'}})| \leq t_2 \\ 2|\cos(\theta_{\boldsymbol{x}_i, \boldsymbol{x}_{i'}})| - 1, & \text{otherwise} \end{cases} \tag{6.17}$$

where $0 < t_2 < t_1 < 1$ are two thresholds. For any sample $\boldsymbol{x}$, its k-bit hash code is written as $H(\boldsymbol{x}) = [h_1(\boldsymbol{x}), \cdots, h_k(\boldsymbol{x})]$, and $\sum_{j=1}^k h_j(\boldsymbol{x}_i) h_j(\boldsymbol{x}_{i'}) = H(\boldsymbol{x}_i) H^\top(\boldsymbol{x}_{i'})$. By taking advantage of the learning goal given in eq. (6.16), we formulate a least-squares style objective function $\mathcal{Q}$ to learn X_l's binary codes as

$$\min_{B_l \in \{1,-1\}^{l \times k}} \mathcal{Q} = \left\| \frac{1}{k} B_l B_l^\top - S \right\|_{\mathrm{F}}^2, \tag{6.18}$$

where $B_l = \begin{bmatrix} H(\boldsymbol{x}_1) \\ \cdots\cdots \\ H(\boldsymbol{x}_l) \end{bmatrix}$ represents the code matrix of X_l and $\|.\|_{\mathrm{F}}$ denotes the Frobenius norm.

The defined matrix S is akin to the pseudo pairwise label matrix S that we have defined in Chapter 5 and was computed from metric-based supervision. The difference is that this time we introduce "soft" labels instead of 0 labels previously used. As well, the objective function $\mathcal{Q}$ proposed in eq. (6.18) for learning the code matrix B_l is exactly the same as the objective function eq. (5.8) we have proposed in Chapter 5. This confirms our expectation, made in Section 5.6 of Chapter 5, that optimizing code inner products will have significant impact on learning to hash.

The thresholds t_1, t_2 used in eq. (6.17) have an important role. When two inputs are prone to being parallel so that $|\cos(\theta_{\boldsymbol{x}_i,\boldsymbol{x}_{i'}})|$ is large enough ($\geq t_1$), minimizing $\mathcal{Q}$ drives each bit of their codes to collide, i.e., $H(\boldsymbol{x}_i)H^\top(\boldsymbol{x}_{i'})/k = 1$; when two inputs tend to be perpendicular so that $|\cos(\theta_{\boldsymbol{x}_i,\boldsymbol{x}_{i'}})|$ is small enough ($\leq t_2$), minimizing $\mathcal{Q}$ tries to make their codes bit-by-bit different, i.e., $H(\boldsymbol{x}_i)H^\top(\boldsymbol{x}_{i'})/k = -1$.

Let us follow the optimization flowchart presented in Section 5.4.3 of Chapter 5 to solve eq. (6.18). With simple algebra, one can rewrite $\mathcal{Q}$ as

$$\begin{aligned} \min_{(\boldsymbol{u}_j,\boldsymbol{v}_j)_{j=1}^k} & \left\| \sum_{j=1}^{k} \boldsymbol{b}_j \boldsymbol{b}_j^\top - kS \right\|_{\mathrm{F}}^2 \\ \text{s.t.} \quad & \boldsymbol{b}_j = \begin{bmatrix} \operatorname{sgn}\left(\boldsymbol{u}_j^\top \boldsymbol{x}_1 \boldsymbol{x}_1^\top \boldsymbol{v}_j\right) \\ \cdots\cdots \\ \operatorname{sgn}\left(\boldsymbol{u}_j^\top \boldsymbol{x}_l \boldsymbol{x}_l^\top \boldsymbol{v}_j\right) \end{bmatrix}. \end{aligned} \tag{6.19}$$

Every bit vector $\boldsymbol{b}_j \in \{1, -1\}^l$ in $B_l = [\boldsymbol{b}_1, \cdots, \boldsymbol{b}_k]$ determines one hash function h_j parameterized by one projection pair $(\boldsymbol{u}_j, \boldsymbol{v}_j)$. Note that $\boldsymbol{b}_j$'s are separable in the summation, which inspires a greedy idea for solving $\boldsymbol{b}_j$'s sequentially. At a time, it only involves solving one bit vector $\boldsymbol{b}_j(\boldsymbol{u}_j, \boldsymbol{v}_j)$ given the previously solved vectors $\boldsymbol{b}_1^*, \cdots, \boldsymbol{b}_{j-1}^*$. Let us define a residue matrix $R_{j-1} = kS - \sum_{j'=1}^{j-1} \boldsymbol{b}_{j'} \boldsymbol{b}_{j'}^\top$ with $R_0 = kS$. Then, $\boldsymbol{b}_j$ can be pursued by

minimizing the following cost

$$\begin{aligned}\left\|\boldsymbol{b}_j\boldsymbol{b}_j^\top - R_{j-1}\right\|_{\mathrm{F}}^2 &= \left(\boldsymbol{b}_j^\top \boldsymbol{b}_j\right)^2 - 2\boldsymbol{b}_j^\top R_{j-1}\boldsymbol{b}_j + \mathrm{tr}(R_{j-1}^2) \\ &= -2\boldsymbol{b}_j^\top R_{j-1}\boldsymbol{b}_j + l^2 + \mathrm{tr}(R_{j-1}^2) \\ &= -2\boldsymbol{b}_j^\top R_{j-1}\boldsymbol{b}_j + const. \end{aligned} \tag{6.20}$$

Discarding the constant term, the final cost is given as

$$g(\boldsymbol{u}_j, \boldsymbol{v}_j) = -\boldsymbol{b}_j^\top R_{j-1}\boldsymbol{b}_j. \tag{6.21}$$

Note that $g(\boldsymbol{u}_j, \boldsymbol{v}_j)$ is lower-bounded as eq. (6.20) is always nonnegative. However, minimizing g is not easy because it is neither convex nor smooth. Below we propose an approximate optimization algorithm like what we have done in Chapter 5.

Since the hardness of minimizing g lies in the sign function, we replace sgn() in $\boldsymbol{b}_j$ with the sigmoid-shaped function $\varphi(x) = 2/(1+\exp(-x)) - 1$ which is sufficiently smooth and well approximates $\mathrm{sgn}(x)$ when $|x| > 6$. Subsequently, we propose to optimize a smooth surrogate $\tilde{g}$ of g defined by

$$\tilde{g}(\boldsymbol{u}_j, \boldsymbol{v}_j) = -\tilde{\boldsymbol{b}}_j^\top R_{j-1}\tilde{\boldsymbol{b}}_j, \tag{6.22}$$

where the vector

$$\tilde{\boldsymbol{b}}_j = \begin{bmatrix} \varphi\left(\boldsymbol{u}_j^\top \boldsymbol{x}_1\boldsymbol{x}_1^\top \boldsymbol{v}_j\right) \\ \cdots\cdots \\ \varphi\left(\boldsymbol{u}_j^\top \boldsymbol{x}_l\boldsymbol{x}_l^\top \boldsymbol{v}_j\right) \end{bmatrix}. \tag{6.23}$$

We derive the gradient of $\tilde{g}$ with respect to $[\boldsymbol{u}_j^\top, \boldsymbol{v}_j^\top]^\top$ as follows

$$\nabla\tilde{g} = -\begin{bmatrix} X_l\Sigma_j X_l^\top \boldsymbol{v}_j \\ X_l\Sigma_j X_l^\top \boldsymbol{u}_j \end{bmatrix}, \tag{6.24}$$

where $\Sigma_j \in \mathbb{R}^{l\times l}$ is a diagonal matrix whose diagonal elements come from the l-dimensional vector $(R_{j-1}\tilde{\boldsymbol{b}}_j) \odot (\mathbf{1} - \tilde{\boldsymbol{b}}_j \odot \tilde{\boldsymbol{b}}_j)$. Here the symbol $\odot$ represents the Hadamard product (i.e., elementwise product), and $\mathbf{1}$ denotes a constant vector with l 1 entries. Since the original cost g in eq. (6.21) is lower-bounded, its smooth surrogate $\tilde{g}$ in eq. (6.22) is lower-bounded as well. We are thus able to minimize $\tilde{g}$ using the regular gradient descent technique. Note

that the smooth surrogate $\tilde{g}$ is still nonconvex, so it is unrealistic to look for a global minima of $\tilde{g}$. For fast convergence, we adopt a pair of random projections $(\boldsymbol{u}_j^0, \boldsymbol{v}_j^0)$, which were used in $h_j^{\mathcal{B}}$, as a warm start and apply Nesterov's gradient method [134] to accelerate the gradient decent procedure. In most cases we attain a locally optimal $(\boldsymbol{u}_j^*, \boldsymbol{v}_j^*)$ at which $\tilde{g}(\boldsymbol{u}_j^*, \boldsymbol{v}_j^*)$ is very close to its lower bound.

The final optimized bilinear hash functions are given as $\left\{h_j(\boldsymbol{z}) = \operatorname{sgn}\left((\boldsymbol{u}_j^*)^\top \boldsymbol{z}\boldsymbol{z}^\top \boldsymbol{v}_j^*\right)\right\}_{j=1}^k$. Although, unlike the randomized hashing, it is not easy to prove their theoretical properties such as the collision probability, they result in a more accurate point-to-hyperplane search than the randomized functions $\left\{h_j^{\mathcal{B}}\right\}$, as demonstrated by the subsequent experiments in Section 6.7.

Now we can present a compact hyperplane hashing algorithm using the learned bilinear hash functions $H = [h_1, \cdots, h_k]$. With H in hand, we implement the proposed hyperplane hashing by simply treating '-1' bit as '0' bit. In the preprocessing stage, each database point $\boldsymbol{x}$ is converted into a k-bit hash code $H(\boldsymbol{x})$ and stored in a single hash table with k-bit hash keys as addresses. To perform search at the query time, given a hyperplane normal $\boldsymbol{w}$, the hashing algorithm

1) extracts its hash key $H(\boldsymbol{w})$ and executes the bitwise NOT operation to obtain the key $\overline{H(\boldsymbol{w})}$;

2) looks up hash buckets by addressing $\overline{H(\boldsymbol{w})}$ up to a small Hamming radius into the hash table, obtaining a short list $\mathcal{L}$ whose points are retrieved from the found hash buckets;

3) scans the short list $\mathcal{L}$ and then returns the database point $x^* = \arg\min_{\boldsymbol{x}\in\mathcal{L}} |\boldsymbol{w}^\top \boldsymbol{x}|/\|\boldsymbol{w}\|$.

In fact, searching within a small Hamming ball centered at the flipped code $\overline{H(\boldsymbol{w})}$ is equivalent to searching the codes that have largest possible Hamming distances to the code $H(\boldsymbol{w})$ in the Hamming space.

6.7 Experiments

6.7.1 Datasets

We conduct experiments on two publicly available datasets including the **20 Newsgroups** textual corpus [126] and the 1.06 million subset, called **Tiny-1M**, of the 80 million tiny

image collection [174]. The first dataset is the version 2 [21] of **20 Newsgroups**. It is comprised of 18,846 documents from 20 newsgroup categories. Each document is represented by a 26,214-dimensional *tf-idf* feature vector that is ℓ_2 normalized. The **Tiny-1M** dataset is a union of **CIFAR-10** [91] and one million tiny images sampled from the entire 80 million tiny image set. **CIFAR-10** is a labeled subset of the 80 million tiny image set, consisting of a total of 60,000 color images from ten object categories each of which has 6,000 samples. The other one million images do not have annotated labels. In our experiments, we treat them as the "other" class besides the ten classes appearing in **CIFAR-10**, since they were sampled as the farthest one million images to the mean image of **CIFAR-10**. Each image in **Tiny-1M** is represented by a 384-dimensional GIST [138] feature vector. Note that this **Tiny-1M** data set is new and different from what we have used in the experiments conducted in Chapter 5, because its one million unlabeled images are not necessarily the same as the one million unlabeled images included in the old **Tiny-1M** dataset.

For each dataset, we train a linear SVM in the one-versus-all setting with an initially labeled set which contains randomly selected labeled samples from all classes, and then run active sample selection for 300 iterations. The initially labeled set for **20 Newsgroups** includes 5 samples per class, while for **Tiny-1M** includes 50 samples per class. For both datasets, we try 5 random initializations. After each sample selection is made, we add it to the labeled set and re-train the SVM. We use LIBLINEAR [46] for running linear SVMs. All our experiments are run on a workstation with a 2.53 GHz Intel Xeon CPU and 48GB RAM.

6.7.2 Evaluations and Results

We carry out SVM active learning [173] using the minimum-margin based sample selection criterion for which we apply hyperplane hashing techniques to expedite the selection procedure. To validate the actual performance of the discussed hyperplane hashing methods, we compare them with two baselines: *random selection* where the next label request is randomly made, and *exhaustive selection* where the margin criterion is evaluated for all currently unlabeled samples. We compare four hashing methods including two randomized linear hash schemes AH-Hash and EH-Hash [75], the proposed randomized bilinear hash

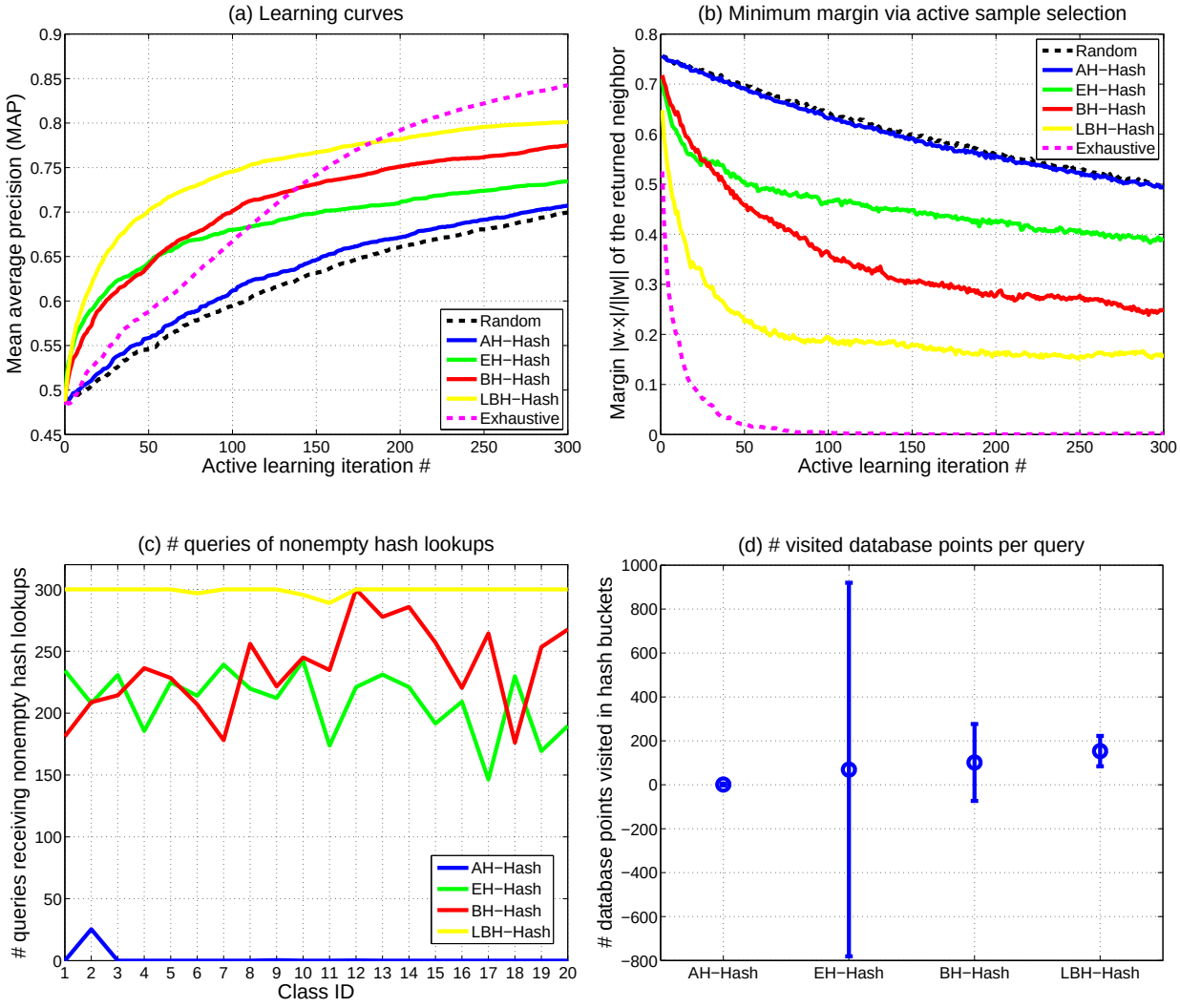


Figure 6.7: Results on **20 Newsgroups** using 16 hash bits. (a) Learning curves of MAP averaged over 20 classes and 5 runs; (b) minimum-margin curves of active sample selection averaged over 20 classes and 5 runs; (c) the number of queries (≤ 300) receiving nonempty hash lookups across 20 classes averaged over 5 runs; and (d) the number of database points visited in the found hash buckets per query averaged over 5 runs.

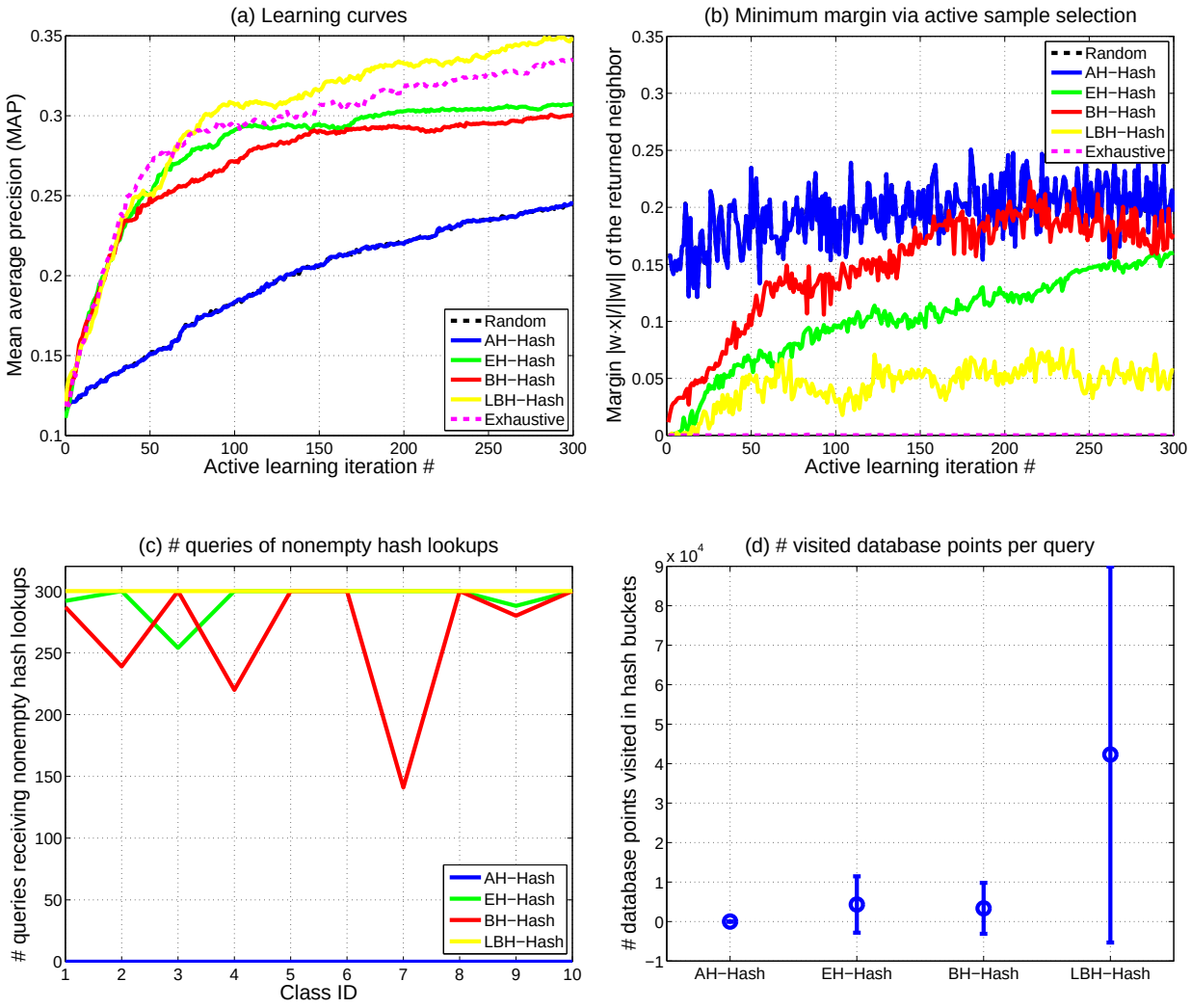


Figure 6.8: Results on **Tiny-1M** using 20 hash bits. (a) Learning curves of MAP averaged over 10 classes and 5 runs; (b) minimum-margin curves of active sample selection averaged over 10 classes and 5 runs; (c) the number of queries ($\leq$ 300) receiving nonempty hash lookups across 10 classes averaged over 5 runs; and (d) the number of database points visited in the found hash buckets per query averaged over 5 runs.

scheme BH-Hash, and the proposed learning-based bilinear hash scheme that we call *LBH-Hash*. Notice that we use the same random projections for AH-Hash, BH-Hash, and the initialization of LBH-Hash to shed light on the effect of bilinear hashing which is equivalent to XNOR two bits. We also follow the dimension subsampling trick in [75] to accelerate EH-Hash's heavy computations in evaluating hash functions. In order to train our proposed LBH-Hash, we randomly sample 500 and 5,000 database points from **20 Newsgroups** and **Tiny-1M**, respectively. The two thresholds t_1, t_2 used for implementing explicit collision are acquired according to the following rule: compute the absolute cosine matrix C between the l sampled points $\{\boldsymbol{x}_i\}_{i=1}^{l}$ and all data points, average the top 5% values among $C_{i\cdot}$'s across $i = 1, \cdots, l$ as t_1, and average the bottom 5% values as t_2.

So as to make the referred four hashing methods work under a compact hashing mode for fair comparison, we employ a single hash table with short code length. Concretely, on **20 Newsgroups** we use 16 hash bits for EH-Hash, BH-Hash, and LBH-Hash, and 32 bits for AH-Hash because of its dual-bit hashing spirit. When applying each hashing method in an AL iteration, we perform hash lookup within Hamming radius 3 in the corresponding hash table and then scan the points in the found hash buckets, resulting in the neighbor near to the current SVM's decision hyperplane. Likewise, we use 20 bits for EH-Hash, BH-Hash, and LBH-Hash, and 40 bits for AH-Hash on **Tiny-1M**; the Hamming radius for hash lookup is set to 4. It is possible that a hashing method finds all empty hash buckets in the small Hamming ball. In that case, we apply random selection as a supplement.

We evaluate the performance of four hyperplane hashing methods in terms of:

1) the average precision (AP) which is computed by ranking the current unlabeled sample set with the current SVM classifier at each AL iteration;

2) the minimum margin (the smallest point-to-hyperplane distance $|\boldsymbol{w}^\top \boldsymbol{x}|/\|\boldsymbol{w}\|$) of the neighbor returned by a hyperplane hashing algorithm at each AL iteration;

3) the number of queries among a total of 300 for every class that receive nonempty hash lookups;

4) the number of database points that a hyperplane hashing algorithm visit in the found hash buckets per query.

The former two results are averaged over all classes and 5 runs, and the latter two are

Table 6.4: Results on **20 Newsgroups** using 8 and 12 hash bits. All preprocessing and search time is recorded in second. At a fixed bit number, two highest MAP values achieved by hashing are displayed in boldface type.

single hash table	8 bits			12 bits		
Method	**MAP**	**Preprocess Time**	**Search Time**	**MAP**	**Preprocess Time**	**Search Time**
AH-Hash	0.7142	0.07	0.07	0.7047	0.13	0.05
EH-Hash	0.7588	383.0	3.41	0.7580	384.1	3.38
BH-Hash	**0.8140**	0.06	0.16	**0.7892**	0.10	0.10
LBH-Hash	**0.8184**	326.8	0.17	**0.8162**	506.7	0.10
Random	0.6999	–	–	—		
Exhaustive	0.8426	–	0.52	—		

averaged over 5 runs. We report such results in Figure 6.7 and Figure 6.8, which clearly show that:

1) LBH-Hash achieves the highest mean AP (MAP) among all compared hashing methods, and even outperforms exhaustive selection at some AL iterations;

2) LBH-Hash accomplishes the minimum margin closest to that by exhaustive selection;

3) LBH-Hash enjoys almost all nonempty hash lookups (AH-Hash gets almost all empty lookups).

The superior performance of LBH-Hash corroborates that the proposed bilinear hash function and the associated learning technique are successful in utilizing the underlying data information to yield compact yet discriminative hash codes. The higher collision probability accrues to the randomized BH-Hash which already works well and outperforms the existing randomized methods AH-Hash and EH-Hash in most situations.

Finally, we report the computational efficiency in Tables 6.4, 6.5, 6.6 and 6.7 which again corroborate both accuracy and speed advantages of LBH-Hash. We keep a single hash table and adopt varying hash code length, ranging from 8 bits to 20 bits, for each of four compared hyperplane hashing methods. To perform hash lookups, we set the Hamming radius to 1, 2, 3, and 4 for 8, 12, 16, and 20 hash bits, respectively. In Tables 6.4-6.7, mean average

Table 6.5: Results on **20 Newsgroups** using 16 hash bits. All preprocessing and search time is recorded in second. Two highest MAP values achieved by hashing are displayed in boldface type.

single hash table	16 bits		
Method	**MAP**	**Preprocess Time**	**Search Time**
AH-Hash	0.7074	0.17	0.05
EH-Hash	0.7346	385.2	3.33
BH-Hash	**0.7752**	0.14	0.07
LBH-Hash	**0.8011**	677.2	0.06
Random	0.6999	–	–
Exhaustive	0.8426	–	0.52

precision (MAP) ($[0, 1]$) is the mean of the SVM's average precision after 300 AL iterations over all classes and 5 runs; "Preprocess Time" (in second) refers to the preprocessing time for a hashing method to compress all of the database points to hash codes (for LBH-Hash, such time includes the time spent on learning the hash functions from the training data); "Search Time" (in second) refers to the average search time per query. Unlike the other methods, AH-Hash always uses twice hash bits to comply with its dual-bit hashing theme.

Let us consider the reported MAP first. Tables 6.4 and 6.5 reveal that on **20 Newsgroups**,

$$\text{Random} < \text{AH} - \text{Hash} < \text{EH} - \text{Hash} < \text{BH} - \text{Hash} < \text{LBH} - \text{Hash} < \text{Exhaustive}$$

where $<$ means inferior MAP. On **Tiny-1M**, Tables 6.6 and 6.7 indicate that

$$\text{Random} \approx \text{AH} - \text{Hash} < \text{EH} - \text{Hash} < \text{BH} - \text{Hash} < \text{LBH} - \text{Hash} < \text{Exhaustive}$$

when using bits smaller than 20, and

$$\text{Random} \approx \text{AH} - \text{Hash} < \text{BH} - \text{Hash} < \text{EH} - \text{Hash} < \text{Exhaustive} < \text{LBH} - \text{Hash}$$

when using 20 bits. LBH-Hash consistently surpasses the other three hashing methods in MAP.

Table 6.6: Results on **Tiny-1M** using 8 and 12 hash bits. All preprocessing and search time is recorded in second. At a fixed bit number, two highest MAP values achieved by hashing are displayed in boldface type.

single hash table	8 bits			12 bits		
Method	**MAP**	**Preprocess Time**	**Search Time**	**MAP**	**Preprocess Time**	**Search Time**
AH-Hash	0.2488	1.0	0.12	0.2417	1.4	0.10
EH-Hash	0.3182	577.2	0.98	0.3147	884.1	0.54
BH-Hash	**0.3213**	1.2	0.86	**0.3208**	1.4	0.58
LBH-Hash	**0.3252**	296.6	1.38	**0.3313**	419.4	0.98
Random	0.2440	–	–	—		
Exhaustive	0.3356	–	14.2	—		

Second, it is observed that AH-Hash and BH-Hash are both efficient considering preprocessing time, while EH-Hash and LBH-Hash need much longer preprocessing time. The preprocessing time of EH-Hash is $O(d^2n + dkn)$ due to the $O(d^2 + dk)$-complexity EH hash function computation for each database point. The preprocessing time of LBH-Hash is $O(2dkn + (dl + l^2)Tk)$ in which l ($\ll n$) is the number of the sampled training points and T is the number of optimization iterations. It is noted that EH-Hash's time is quadratic in the data dimension d ($k \ll d$) while LBH-Hash's time is linear in d. For those really large scale data collections with $O(10^3)$ or even higher dimension, the quadratic dimension dependence of EH-Hash will trigger unaffordable computational costs, but our proposed hashing approaches including both BH-Hash and LBH-Hash will enjoy the linear dependence on dimension. For example, EH-Hash's preprocessing time is about twice longer than LBH-Hash's on the **Tiny-1M** dataset.

Although LBH-Hash takes longer preprocessing time than BH-Hash as shown in Tables 6.4-6.7, the reported time is mostly spent on offline training and indexing. In practice, the online search (query) time is more critical. Considering the search time, all of the compared hashing methods except EH-Hash are much faster than exhaustive search. AH-Hash is fastest, but it incurs many empty hash lookups, as disclosed in Figures 6.7(c) and 6.8(c).

Table 6.7: Results on **Tiny-1M** using 16 and 20 hash bits. All preprocessing and search time is recorded in second. At a fixed bit number, two highest MAP values achieved by hashing are displayed in boldface type.

single hash table	16 bits			20 bits		
Method	**MAP**	**Preprocess Time**	**Search Time**	**MAP**	**Preprocess Time**	**Search Time**
AH-Hash	0.2404	1.6	0.10	0.2444	1.7	0.09
EH-Hash	0.3094	1058.1	0.22	**0.3075**	1343.6	0.15
BH-Hash	**0.3141**	1.8	0.16	0.3010	2.0	0.12
LBH-Hash	**0.3341**	586.5	0.95	**0.3469**	883.7	0.88
Random	0.2440	–	–	—		
Exhaustive	0.3356	–	14.2	—		

On **20 Newsgroups** that is very high-dimensional (over 20,000 dimensions), EH-Hash is slowest due to the above-mentioned high time complexity of its hash function computation, and even slower than exhaustive search. On **20 Newsgroups**, BH-Hash and LBH-Hash demonstrate almost the same fast search speed; on **Tiny-1M**, LBH-Hash is slower than BH-Hash but is acceptably fast for online search.

6.8 Summary and Discussion

We have addressed a seldom studied problem, hyperplane hashing, by proposing a novel bilinear hash function which allows efficient search of database points near a hyperplane query. Even using random projections, the proposed hash function enjoys a higher probability of collision than the existing randomized methods. By learning the projections further, we can improve a set of bilinear hash functions collaboratively for multiple bits. Better than the randomized hash functions, these learning-based hash functions yield compact yet discriminative codes which permit substantial savings in both storage and time needed during nearest neighbor search. Large-scale active learning experiments carried out on one document database and one image database have demonstrated the superior comprehensive performance of the developed compact hyperplane hashing approach.

For future work, it will be nice to develop theoretical guarantees first for the learned bilinear hash functions despite the difficulty. Second, by means of appropriate adjustments (e.g., design a novel hash function like $h(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{j=1}^{k} \boldsymbol{u}_j^\top \boldsymbol{x}\boldsymbol{x}^\top \boldsymbol{v}_j\right)$), our proposed bilinear hash function is not only suitable for dealing with point-to-hyperplane search but also capable of resolving many intricate search problems such as point-to-subspace search, subspace-to-subspace search, and so on. In addition, it will be very interesting to integrate traditional linear hash functions and bilinear hash functions into a hierarchical hashing framework which could tackle some complicated search problems involving both point-to-point and point-to-hyperplane searches.

Part III

Conclusions

Chapter 7

Conclusions

7.1 Summary of Contributions

This thesis is dedicated to developing scalable machine learning techniques for large-scale classification and nearest neighbor search over gigantic databases. We have focused on two groups of techniques: scalable classification with graphs, presented in Part I, and nearest neighbor search with hashing, covered in Part II.

We have demonstrated, both in theory and practice, effective and efficient solutions with clear performance gains in large-scale experiments. Specifically, we have developed the following methods for classification.

a1. Large Graph Construction: The prior graph-based semi-supervised learning (GSSL) methods scale poorly with the data size because of the quadratic time complexity for exact neighborhood graph construction, which prevents wide adoption of GSSL. We have presented approximate neighborhood graphs, called *Anchor Graphs*, which can be efficiently constructed in linear time. Our experiments have shown that Anchor Graphs exhibit high fidelity to conventional kNN graphs yet with much shorter construction time. Based on Anchor Graphs, we have developed a simple GSSL algorithm *Anchor Graph Regularization* (AGR) which has demonstrated promising performance gains compared to the state-of-the-art GSSL algorithms. Both time and memory needed by AGR grow linearly with the data size. The proposed Anchor Graphs can be readily applied to other general graph-driven machine learning problems, such as dimensionality reduction, clustering, manifold-based

ranking, etc.

a2. Large-Scale Semi-Supervised Learning: The proposed Anchor Graphs also enable us to develop a novel solution for learning scalable semi-supervised kernel classifiers. Kernel machines are known to be more robust than the linear counterparts due to their ability in separating practical data points that are mostly linearly inseparable, but training nonlinear kernel machines is often costly. Using the idea of Anchor Graphs, we have developed several low-rank kernel generation methods which result in a very simple linearization process of the original nonlinear kernel, thereby transforming the expensive kernel machine training task to a standard linear SVM training task. Since training nonlinear kernel machines in semi-supervised settings can be reduced to training linear SVMs in supervised settings, the computational cost for classifier training is substantially reduced, only costing $O(l)$ time where l is the number of labeled examples and much smaller than the total data size n. The low-rank kernels bear closed-form expressions, taking linear time $O(n)$ to generate them. Our experiments have manifested that a linear SVM using a linearized low-rank kernel exhibits superior classification accuracy over state-of-the-arts and also outperforms AGR.

In the second part of the thesis, we focus on large-scale nearest neighbor search using hashing techniques. Specifically, we have proposed new theories and algorithms for unsupervised hashing, supervised hashing, and hyperplane hashing.

b1. Unsupervised Hashing: We have developed a practical graph-based unsupervised hashing approach which respects the underlying manifold structure of the input data to capture semantic neighborhoods on manifolds and return meaningful nearest neighbors for a query. We further showed that Anchor Graphs can overcome the computationally prohibitive steps of building and manipulating large graph Laplacians by approximating graph adjacency matrices with low-rank matrices. As such, the hash functions can be efficiently obtained by thresholding the lower (smoother) eigenfunctions of the Anchor Graph Laplacian in a hierarchical fashion. Experimental comparison showed that the proposed Anchor Graph Hashing (AGH) enjoys significant performance gains over the state-of-the-art hashing methods in finding semantically similar neighbors.

b2. Supervised Hashing: We have developed a kernel-based supervised hashing

(KSH) approach to incorporate supervised information in availability. We summarize the success of KSH to three main aspects: (1) kernel-based hash functions were exploited to handle linearly inseparable data; (2) an elegant objective function designed for supervised hashing was skillfully formulated based on code inner products instead of Hamming distances; and (3) a greedy optimization algorithm was deployed to solve the hash functions efficiently. Large-scale image search experiments have demonstrated that KSH surpasses our unsupervised method AGH and state-of-the-arts by a large margin in searching both semantic neighbors and metric neighbors. In theory, KSH does not need any special assumptions about the data other than a predefined kernel function. In practice, we have found that on some datasets where manifolds are not evident, KSH works better than AGH which depends on the manifold assumption.

b3. Hyperplane Hashing: We have addressed the seldom studied problem hyperplane hashing which aims at speeding up the point-to-hyperplane search process. We have designed a specialized bilinear hash function which allows efficient search of database points near a hyperplane query. Even when using random projections, the proposed hash function enjoys a higher probability of collision than the existing randomized hash functions. By learning the projections further, we can achieve a series of meaningful bilinear hash functions. Better than randomized hash functions, these hash functions via learning yield compact yet discriminative codes which permit substantial savings in both storage and time needed during nearest neighbor search. Large-scale active learning experiments have demonstrated the superior comprehensive performance of the developed compact hyperplane hashing approach.

As a bridge connecting Part I and Part II, our compact hyperplane hashing approach described in Chapter 6 can directly benefit classification in an active learning environment.

7.2 Future Work

Despite the significant progress made in this thesis, there remain several open exciting challenges for large-scale machine learning. In the following, we discuss some promising topics that we will proceed to on our future research agenda.

c1. Ensemble Anchor Graphs: We have found in Chapter 2 that an increased number of anchors lead to a sparser Anchor Graph. Hence, the number of anchors used in the implementations may control the quality of the resulting Anchor Graph. However, incorporating more anchors into the construction of an Anchor Graph will inevitably lead to a higher computational complexity. One preliminary idea is to construct a group of Anchor Graphs in parallel and let each Anchor Graph account for only a subset of the whole data set. The subsequent interesting issue is how to merge these Anchor Graphs in a principled way. We call this approach *Ensemble Anchor Graphs* and will address this in the future work.

c2. Large-Scale Supervised Learning: Although there has been progress in training large-scale linear classifiers efficiently, linear classifiers suffer from limited discriminating power. However, training nonlinear classifiers at a large scale is usually computationally difficult although it could result in higher classification accuracy. To this end, we will address the scalability issue of supervised nonlinear classifier training by generalizing the *anchor* idea used in this thesis to develop a hierarchical classification model *Anchor Vector Machine* (AVM). The proposed AVM model consists of two layers, in which the first layer aims at incorporating a priori kernel by running sparse Gaussian process regression using the anchor vectors in the low-level feature space. The second layer performs *Local Anchor Coding* (LAC) in the high-level semantic space by exploiting the Gaussian process regression outputs and the class labels of the anchor vectors. The first layer helps achieve the complexity reduction, while the second layer can compensate for the performance loss incurred in the first layer due to the use of the sparse approximation to the full-size Gaussian process regression.

c3. Subspace-to-Subspace Hashing: Till now, we have discussed two key paradigms for hashing: point-to-point hashing presented in Chapter 4 and Chapter 5, and point-to-hyperplane hashing presented in Chapter 6. We believe that the bilinear hash function proposed in Chapter 6 is not only suitable for dealing with point-to-hyperplane search but also capable of resolving many intricate search problems such as point-to-subspace search, subspace-to-subspace search, and so on, if we make appropriate adjustments to the bilinear hash function. For example, we may design a novel hash function like $h(\boldsymbol{x}) =$

$\text{sgn}\left(\sum_{j=1}^{k} \boldsymbol{u}_j^\top \boldsymbol{x}\boldsymbol{x}^\top \boldsymbol{v}_j\right)$, which can handle subspace queries with rank k. Then, we may adopt randomized or learning-based hashing algorithms to seek the required projection vectors $\{(\boldsymbol{u}_j, \boldsymbol{v}_j)\}$ which can then be used to fulfill subspace-to-subspace hashing.

Part IV

Bibliography

Bibliography

[1] `http://www.gaussianprocess.org/gpml/data/`.

[2] M. Aly, M. Munich, and P. Perona. Distributed kd-trees for retrieval from very large image collections. In *Proceedings of British Machine Vision Conference*, 2011.

[3] M. Aly, M. Munich, and P. Perona. Indexing in large scale image collections: Scaling properties and benchmark. In *Proceedings of IEEE Workshop on Applications of Computer Vision*, pages 418–425, 2011.

[4] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, January 2008.

[5] A. Argyriou, M. Herbster, and M. Pontil. Combining graph laplacians for semi-supervised learning. In *Advances in Neural Information Processing Systems 18*, 2005.

[6] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45(6):891–923, November 1998.

[7] A. Azran. The rendezvous algorithm: multiclass semi-supervised learning with markov random walks. In *Proceedings of the Twenty-Fourth International Conference on Machine learning*, pages 49–56, 2007.

[8] R. Basri, T. Hassner, and L. Zelnik-Manor. Approximate nearest subspace search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(2):266–278, February 2011.

[9] S. Basu, M. Bilenko, and R. Mooney. A probabilistic framework for semi-supervised clustering. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 59–68, 2004.

[10] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, June 2003.

[11] M. Belkin and P. Niyogi. Towards a theoretical foundation for laplacian-based manifold methods. *Journal of Computer and System Sciences*, 74(8):1289–1308, December 2008.

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

[13] Y. Bengio, O. Delalleau, N. L. Roux, J.-F. Paiement, P. Vincent, and M. Ouimet. Learning eigenfunctions links spectral embedding and kernel pca. *Neural Computation*, 16(10):2197–2219, October 2004.

[14] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[15] A. Blum and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 19–26, 2001.

[16] A. Blum, J. D. Lafferty, M. R. Rwebangira, and R. Reddy. Semi-supervised learning using randomized mincuts. In *Proceedings of the Twenty-first International Conference on Machine learning*, pages 13–20, 2004.

[17] A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, September 2005.

[18] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[19] A. Z. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences*, pages 21–29, 1997.

[20] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, June 2000.

[21] D. Cai. `http://www.zjucadcg.cn/dengcai/Data/TextData.html`.

[22] C. Campbell, N. Cristianini, and A. J. Smola. Query learning with large margin classifiers. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 111–118, 2000.

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

[24] O. Chapelle, V. Sindhwani, and S. S. Keerthi. Optimization techniques for semi-supervised support vector machines. *Journal of Machine Learning Research*, 9:203–233, February 2008.

[25] O. Chapelle, J. Weston, and B. Scholkopf. Cluster kernels for semi-supervised learning. In *Advances in Neural Information Processing Systems 15*, pages 585–592, 2002.

[26] O. Chapelle and A. Zien. Semi-supervised classification by low density separation. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pages 57–64, 2005.

[27] M. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 380–388, 2002.

[28] J. Chen, H.-R. Fang, and Y. Saad. Fast approximate $k$nn graph construction for high dimensional data via recursive lanczos bisection. *Journal of Machine Learning Research*, 10:1989–2012, September 2009.

[29] K. Chen and S. Wang. Semi-supervised learning via regularized boosting working on multiple semi-supervised assumptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):129–143, January 2011.

[30] X. Chen and D. Cai. Large scale spectral clustering with landmark-based representation. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[31] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng. Nus-wide: A real-world web image database from national university of singapore. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, 2009.

[32] O. Chum, J. Philbin, M. Isard, and A. Zisserman. Scalable near identical image and shot detection. In *Proceedings of the 6th ACM International Conference on Image and Video Retrieval*, pages 549–556, 2007.

[33] F. Chung. *Spectral Graph Theory*. CBMS Regional Conference Series in Mathematics, American Mathematical Society, 1997.

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

[35] R. R. Coifman, S. Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. W. Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proceedings of the National Academy of Sciences*, 102(21):7426–7431, May 2005.

[36] R. Collobert, F. Sinz, J. Weston, and L. Bottou. Large scale transductive svms. *Journal of Machine Learning Research*, 7:1687–1712, August 2006.

[37] S. I. Daitch, J. A. Kelner, and D. A. Spielman. Fitting a graph to vector data. In *Proceedings of the 26th International Conference on Machine Learning*, pages 201–208, 2009.

[38] S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 537–546, 2008.

[39] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th ACM Symposium on Computational Geometry*, pages 253–262, 2004.

[40] O. Delalleu, Y. Bengio, and N. L. Roux. Efficient non-parametric function induction in semi-supervised learning. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pages 96–103. 2005.

[41] W. Dong, M. Charikar, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web*, pages 577–586, 2011.

[42] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li. Modeling lsh for performance tuning. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 669–678, 2008.

[43] G. Druck and A. McCallum. High-performance semi-supervised learning using discriminatively constrained generative models. In *Proceedings of the 27th International Conference on Machine Learning*, pages 319–326, 2010.

[44] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the ℓ_1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, pages 272–279, 2008.

[45] K. Eshghi and S. Rajaram. Locality sensitive hash functions based on concomitant rank order statistics. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 221–229, 2008.

[46] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[47] L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611, April 2006.

[48] R. Fergus, L. Fei-Fei, P. Perona, and A. Zisserman. Learning object categories from internet image searches. *Proceedings of the IEEE*, 98(8):1453–1466, August 2010.

[49] R. Fergus, Y. Weiss, and A. Torralba. Semi-supervised learning in gigantic image collections. In *Advances in Neural Information Processing Systems 22*, pages 522–530, 2009.

[50] J. H. Friedman, J. L. Bentley, and R. A. Finke. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[51] M. Gavish, B. Nadler, and R. R. Coifman. Multiscale wavelets on trees, graphs and high dimensional data: Theory and applications to semi supervised learning. In *Proceedings of the 27th International Conference on Machine Learning*, pages 367–374, 2010.

[52] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529, 1999.

[53] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, November 1995.

[54] A. B. Goldberg, X. Zhu, A. Furger, and J.-M. Xu. Oasis: Online active semi-supervised learning. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[55] G. H. Golub and C. F. van Loan. *Matrix Computations, 3rd Edition*. Johns Hopkins University Press, 1996.

[56] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 817–824, 2011.

[57] A. Guillory and J. Bilmes. Label selection on graphs. In *Advances in Neural Information Processing Systems 22*, pages 691–699, 2009.

[58] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Second Edition, Springer, 2009.

[59] J. He, M. Li, H. Zhang, H. Tong, and C. Zhang. Generalized manifold-ranking-based image retrieval. *IEEE Transactions on Image Processing*, 15(10):3170–3177, October 2006.

[60] J. He, W. Liu, and S.-F. Chang. Scalable similarity search with optimized kernel hashing. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1129–1138, 2010.

[61] J. He, R. Radhakrishnan, S.-F. Chang, and C. Bauer. Compact hashing with joint optimization of search accuracy and time. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 753–760, 2011.

[62] X. He, D. Cai, and P. Niyogi. Laplacian score for feature selection. In *Advances in Neural Information Processing Systems 18*, 2005.

[63] X. He and P. Niyogi. Locality preserving projections. In *Advances in Neural Information Processing Systems 16*, 2003.

[64] X. He, S. Yan, Y. Hu, P. Niyogi, and H. Zhang. Face recognition using laplacianfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(3):328–340, March 2005.

[65] M. Hein, J.-Y. Audibert, and U. von Luxburg. From graphs to manifolds - weak and strong pointwise consistency of graph laplacians. In *Proceedings of the 18th Annual Conferecnce on Learning Theory*, pages 470–485, 2005.

[66] M. Hein, J.-Y. Audibert, and U. von Luxburg. Graph laplacians and their convergence on random neighborhood graphs. *Journal of Machine Learning Research*, 8:1325–1368, 2007.

[67] M. Hein and M. Maier. Manifold denoising. In *Advances in Neural Information Processing Systems 19*, pages 561–568, 2006.

[68] M. Herbster, G. Lever, and M. Pontil. Online prediction on large diameter graphs. In *Advances in Neural Information Processing Systems 21*, pages 649–656, 2008.

[69] M. Herbster and M. Pontil. Prediction on a graph with a perceptron. In *Advances in Neural Information Processing Systems 19*, pages 577–584, 2006.

[70] M. Herbster, M. Pontil, and L. Wainer. Online learning over graphs. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, pages 305–312, 2005.

[71] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

[72] W. H. Hsu, L. S. Kennedy, and S.-F. Chang. Video search reranking via information bottleneck principle. In *Proceedings of ACM International Conference on Multimedea*, pages 35–44, 2006.

[73] S.-J. Huang, R. Jin, and Z.-H. Zhou. Active learning by querying informative and representative examples. In *Advances in Neural Information Processing Systems 23*, pages 892–900, 2010.

[74] P. Indyk. Chapter 39: Nearest-neighbor searching in high dimensions. *Handbook of Discrete and Computational Geometry, J. E. Goodman and J. O'Rourke (Editors), CRC Press, Second Edition*, April 2004.

[75] P. Jain, S. Vijayanarasimhan, and K. Grauman. Hashing hyperplane queries to near points with applications to large-scale active learning. In *Advances in Neural Information Processing Systems 23*, pages 928–936, 2010.

[76] T. Jebara, J. Wang, and S.-F. Chang. Graph construction and b-matching for semi-supervised learning. In *Proceedings of the 26th International Conference on Machine Learning*, pages 441–448, 2009.

[77] H. Jegou, M. Douze, and C. Schmid. Improving bag-of-features for large scale image search. *International Journal of Computer Vision*, 87(3):191–212, May 2010.

[78] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, January 2011.

[79] H. Jegou, M. Douze, C. Schmid, and P. Perez. Aggregating local descriptors into a compact image representation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3304–3311, 2010.

[80] M. Ji, T. Yang, B. Lin, R. Jin, and J. Han. A simple algorithm for semi-supervised learning with improved generalization error bound. In *Proceedings of the 29th International Conference on Machine Learning*, 2012.

[81] Y.-G. Jiang, J. Wang, and S.-F. Chang. Lost in binarization: query-adaptive ranking for similar image search with compact codes. In *Proceedings of the 1st International Conference on Multimedia Retrieval*, 2011.

[82] Y. Jing and S. Baluja. VisualRank: Applying PageRank to large-scale image search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1877–1890, November 2008.

[83] T. Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 200–209, 1999.

[84] T. Joachims. Transductive learning via spectral graph partitioning. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 290–297, 2003.

[85] R. Johnson and T. Zhang. Graph-based semi-supervised learning and spectral kernel design. *IEEE Transactions on Information Theory*, 54(1):275–288, January 2008.

[86] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, July 2002.

[87] M. Karlen, J. Weston, A. Erkan, and R. Collobert. Large scale manifold transduction. In *Proceedings of the 25th International Conference on Machine Learning*, pages 448–455, 2008.

[88] J. M. Kleinberg, A. Slivkins, and T. Wexler. Triangulation and embedding using small sets of beacons. In *Proceedings of 45th Symposium on Foundations of Computer Science*, pages 444–453, 2004.

[89] W. Kong and W.-J. Li. Double-bit quantization for hashing. In *Proceedings of the 26th AAAI Conference on Artificial Intelligenc*, 2012.

[90] S. Korman and S. Avidan. Coherency sensitive hashing. In *Proceedings of IEEE International Conference on Computer Vision*, pages 1607–1614, 2011.

[91] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Toronto University, April 2009.

[92] B. Kulis, S. Basu, I. Dhillon, and R. Mooney. Semi-supervised graph clustering: A kernel approach. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, pages 457–464, 2005.

[93] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *Advances in Neural Information Processing Systems 22*, pages 1042–1050, 2009.

[94] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(6):1092–1104, June 2012.

[95] B. Kulis, P. Jain, and K. Grauman. Fast similarity search for learned metrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2143–2157, December 2009.

[96] N. Kumar, L. Zhang, and S. Nayar. What is a good nearest neighbors algorithm for finding similar patches in images? In *Proceedings of the 10th European Conference on Computer Vision*, pages 364–378, 2008.

[97] S. Kumar, M. Mohri, and A. Talwalkar. Ensemble nyström method. In *Advances in Neural Information Processing Systems 22*, pages 1060–1068, 2009.

[98] S. Kumar, M. Mohri, and A. Talwalkar. Sampling methods for the nyström method. *Journal of Machine Learning Research*, 2012.

[99] B. Kveton, M. Valko, A. Rahimi, and L. Huang. Semi-supervised learning with max-margin graph cuts. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, pages 421–428, 2010.

[100] J. Lafferty and L. Wasserman. Statistical analysis of semi-supervised regression. In *Advances in Neural Information Processing Systems 20*, pages 801–808, 2007.

[101] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2169–2178, 2006.

[102] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[103] D. T. Lee and C. K. Wong. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9(1):23–29, 1977.

[104] F. Li, G. Lebanon, and C. Sminchisescu. Chebyshev approximations to the histogram χ^2 kernel. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2424–2431, 2012.

[105] M. Li, J. T. Kwok, and B.-L. Lu. Making large-scale nyström approximation possible. In *Proceedings of the 27th International Conference on Machine Learning*, pages 631–638, 2010.

[106] P. Li and A. C. König. Theory and applications of b-bit minwise hashing. *Communications of the ACM*, 54(8):101–109, August 2011.

[107] Z. Li, X.-M. Wu, and S.-F. Chang. Segmentation using superpixels: A bipartite graph partitioning approach. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 789–796, 2012.

[108] R.-S. Lin, D. A. Ross, and J. Yagnik. Spec hashing: Similarity preserving algorithm for entropy-based coding. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 848–854, 2010.

[109] D. Liu, X.-S. Hua, L. Yang, M. Wang, and H.-J. Zhang. Tag ranking. In *Proceedings of the 18th International Conference on World Wide Web*, pages 351–360, 2009.

[110] D. Liu, S. Yan, Y. Rui, and H.-J. Zhang. Unified tag analysis with multi-edge graph. In *Proceedings of ACM International Conference on Multimedea*, pages 25–34, 2010.

[111] T. Liu, A. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. In *Advances in Neural Information Processing Systems 17*, 2004.

[112] W. Liu and S.-F. Chang. Robust multi-class transductive learning with graphs. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 381–388, 2009.

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

[114] W. Liu, Y.-G. Jiang, J. Luo, and S.-F. Chang. Noise resistant graph ranking for improved web image search. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 849–856, 2011.

[115] W. Liu, J. Wang, and S.-F. Chang. Robust and scalable graph-based semisupervised learning. *Proceedings of the IEEE*, 100(9):2624–2638, September 2012.

[116] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2074–2081, 2012.

[117] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1–8, 2011.

[118] W. Liu, J. Wang, Y. Mu, S. Kumar, and S.-F. Chang. Compact hyperplane hashing with bilinear functions. In *Proceedings of the 29th International Conference on Machine Learning*, 2012.

[119] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[120] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe lsh: Efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 950–961, 2007.

[121] M. Maier, U. von Luxburg, and M. Hein. Influence of graph construction on graph-based clustering measures. In *Advances in Neural Information Processing Systems 21*, pages 1025–1032, 2008.

[122] P. K. Mallapragada, R. Jin, A. K. Jain, and Y. Liu. Semiboost: Boosting for semi-supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(11):2000–2014, November 2009.

[123] G. S. Mann and A. McCallum. Simple, robust, scalable semi-supervised learning via expectation regularization. In *Proceedings of the 24th International Conference on Machine Learning*, pages 593–600, 2007.

[124] S. Melacci and M. Belkin. Laplacian support vector machines trained in the primal. *Journal of Machine Learning Research*, 12:1149–1184, 2011.

[125] H. Q. Min and V. Sindhwani. Vector-valued manifold regularization. In *Proceedings of the 28th International Conference on Machine Learning*, pages 57–64, 2011.

[126] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[127] Y. Mu, X. Chen, X. Liu, T.-S. Chua, and S. Yan. Multimedia semantics-aware query-adaptive hashing with bits reconfigurability. *International Journal of Multimedia Information Retrieval*, 1(1):59–70, April 2012.

[128] Y. Mu, J. Shen, and S. Yan. Weakly-supervised hashing in kernel space. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3344–3351, 2010.

[129] Y. Mu, J. Sun, T. X. Han, L. F. Cheong, and S. Yan. Randomized locality sensitive vocabularies for bag-of-features model. In *Proceedings of the 11th European Conference on Computer Vision*, pages 748–761, 2010.

[130] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proceedings of International Conference on Computer Vision Theory and Application*, pages 331–340, 2009.

[131] B. Nadler, S. Lafon, R. R. Coifman, and I. G. Kevrekidis. Diffusion maps, spectral clustering and reaction coordinates of dynamical systems. *Applied and Computational Harmonic Analysis*, 21:113–127, 2006.

[132] B. Nadler, N. Srebro, and X. Zhou. Statistical analysis of semi-supervised learning: The limit of infinite unlabelled data. In *Advances in Neural Information Processing Systems 22*, pages 1330–1338, 2009.

[133] S. A. Nene, S. K. Nayar, and H. Murase. Columbia object image library (coil-20). Technical report, Columbia University, CUCS-005-96, February 1996.

[134] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer Academic Publishers, 2003.

[135] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, pages 849–856, 2001.

[136] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2161–2168, 2006.

[137] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. In *Proceedings of the 28th International Conference on Machine Learning*, pages 353–360, 2011.

[138] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, May-June 2001.

[139] S. M. Omohundro. Five balltree construction algorithms. Technical report, International Computer Science Institute, November 1989.

[140] N. Panda, K.-S. Goh, and E. Y. Chang. Active learning in very large databases. *Multimedia Tools and Applications*, 31(3):249–267, December 2006.

[141] R. Panigrahy. Entropy based nearest neighbor search in high dimensions. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1186–1195, 2006.

[142] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *Advances in Neural Information Processing Systems 22*, pages 1509–1517, 2009.

[143] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems 20*, 2007.

[144] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, December 2000.

[145] R. Salakhutdinov and G. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics*, pages 412–419, 2007.

[146] R. Salakhutdinov and G. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, July 2009.

[147] H. A. Sanchez, J. M. Sotoca, and A. M. Uso. Semi-supervised learning from a translation model between data distributions. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 1165–1170, 2011.

[148] G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 839–846, 2000.

[149] B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[150] G. Shakhnarovich. *Learning Task-Specific Similarity*. PhD thesis, MIT, 2006.

[151] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. The MIT Press, 2006.

[152] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter-sensitive hashing. In *Proceedings of the 9th IEEE International Conference on Computer Vision*, pages 750–759, 2003.

[153] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, August 2000.

[154] C. Silpa-Anan and R. Hartley. Optimised kd-trees for fast image descriptor matching. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[155] V. Sindhwani and S. S. Keerthi. Large scale semi-supervised linear svms. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 477–484, 2006.

[156] V. Sindhwani, S. S. Keerthi, and O. Chapelle. Deterministic annealing for semi-supervised kernel machines. In *Proceedings of the Twenty-Third International Conference on Machine Learning*, pages 841–848, 2006.

[157] V. Sindhwani, P. Niyogi, and M. Belkin. Beyond the point cloud: from transductive to semi-supervised learning. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, pages 824–831, 2005.

[158] A. Singh, R. D. Nowak, and X. Zhu. Unlabeled data: Now it helps, now it doesn't. In *Advances in Neural Information Processing Systems 21*, pages 1513–1520, 2008.

[159] K. Sinha and M. Belkin. Semi-supervised learning using sparse eigenfunction bases. In *Advances in Neural Information Processing Systems 22*, pages 1687–1695, 2009.

[160] D. A. Spielman and S.-H. Teng. Solving sparse, symmetric, diagonally-dominant linear systems in time $o(m^{1.31})$. In *Proceedings of the 44th Symposium on Foundations of Computer Science*, pages 416–427, 2003.

[161] D. A. Spielman and S.-H. Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

[162] R. F. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6(4):579–589, 1991.

[163] C. Strecha, A. M. Bronstein, M. M. Bronstein, and P. Fua. Ldahash: Improved matching with smaller descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1):66–78, January 2012.

[164] D. E. Sturim, D. A. Reynolds, E. Singer, and J. P. Campbel. Speaker indexing in large audio databases using anchor models. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 429–432, 2001.

[165] J. Su, J. S. Shirab, and S. Matwin. Large scale text classification using semisupervised multinomial naive bayes. In *Proceedings of the 28th International Conference on Machine Learning*, pages 97–104, 2011.

[166] A. D. Szlam, M. Maggioni, and R. R. Coifman. Regularization on graphs with function-adapted diffusion processes. *Journal of Machine Learning Research*, 9:1711–1739, 2008.

[167] M. Szummer and T. Jaakkola. Partially labeled classification with markov random walks. In *Advances in Neural Information Processing Systems 14*, pages 945–952, 2001.

[168] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 563–576, 2009.

[169] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000.

[170] K. Terasawa and Y. Tanaka. Spherical lsh for approximate nearest neighbor search on unit hypersphere. *Algorithms and Data Structures, Lecture Notes in Computer Science*, 4619:27–38, 2007.

[171] X. Tian, G. Gasso, and S. Canu. A multi-kernel framework for inductive semi-supervised learning. In *Proceedings of the 19th European Symposium on Artificial Neural Networks*, pages 65–70, 2011.

[172] D. Ting, L. Huang, and M. I. Jordan. An analysis of the convergence of graph laplacians. In *Proceedings of the 27th International Conference on Machine Learning*, pages 1079–1086, 2010.

[173] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2:45–66, November 2001.

[174] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: a large dataset for non-parametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, November 2008.

[175] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large databases for recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[176] I. W. Tsang and J. T. Kwok. Large-scale sparsified manifold regularization. In *Advances in Neural Information Processing Systems 19*, pages 1401–1408, 2006.

[177] M. Valko, B. Kveton, L. Huang, and D. Ting. Online semi-supervised learning on quantized graphs. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, pages 606–614, 2010.

[178] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):480–492, March 2012.

[179] N. Verma, S. Kpotufe, and S. Dasgupta. Which spatial partition trees are adaptive to intrinsic dimension? In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 565–574, 2009.

[180] S. Vijayanarasimhan and K. Grauman. Large-scale live active learning: Training object detectors with crawled data and crowds. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1449–1456, 2011.

[181] S. N. Vitaladevuni, P. Natarajan, R. Prasad, and P. Natarajan. Efficient orthogonal matching pursuit using sparse random projections for scene and video classification. In *Proceedings of IEEE International Conference on Computer Vision*, pages 2312–2319, 2011.

[182] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[183] F. Wang and C. Zhang. Label propagation through linear neighborhoods. *IEEE Transactions on Knowledge and Data Engineering*, 20(1):55–67, January 2008.

[184] J. Wang, T. Jebara, and S.-F. Chang. Graph transduction via alternating minimization. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1144–1151, 2008.

[185] J. Wang, Y.-G. Jiang, and S.-F. Chang. Label diagnosis through self tuning for web image search. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1390–1397, 2009.

[186] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for large scale search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012.

[187] J. Wang, J. Wang, G. Zeng, Z. Tu, R. Gan, and S. Li. Scalable k-nn graph construction for visual descriptors. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1106–1113, 2012.

[188] J. Wang and Y. Xia. Fast graph construction using auction algorithm. In *Proceedings of the 28th Annual Conference on Uncertainty in Artificial Intelligence*, 2012.

[189] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. Locality-constrained linear coding for image classification. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3360–3367, 2010.

[190] M. Wang, F. Sha, and M. I. Jordan. Unsupervised kernel dimension reduction. In *Advances in Neural Information Processing Systems 23*, pages 2379–2387, 2010.

[191] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems 21*, pages 1753–1760, 2008.

[192] J. Weston, C. S. Leslie, D. Zhou, A. Elisseeff, and W. S. Noble. Semi-supervised protein classification using cluster kernels. In *Advances in Neural Information Processing Systems 16*, 2003.

[193] C. K. I. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*, pages 682–688, 2000.

[194] M. Wu and B. Scholkopf. Transductive classification via local learning regularization. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, pages 628–635, 2007.

[195] L. Xiong, J. Li, and C. Zhang. Discriminant additive tangent spaces for object recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[196] B. Xu, J. Bu, C. Chen, D. Cai, X. He, W. Liu, and J. Luo. Efficient manifold ranking for image retrieval. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 525–534, 2011.

[197] H. Xu, J. Wang, Z. Li, G. Zeng, S. Li, and N. Yu. Complementary hashing for approximate nearest neighbor search. In *Proceedings of IEEE International Conference on Computer Vision*, pages 1631–1638, 2011.

[198] J. Yagnik, D. Strelow, D. A. Ross, and R.-S. Lin. The power of comparative reasoning. In *Proceedings of IEEE International Conference on Computer Vision*, pages 2431–2438, 2011.

[199] Y. Yang, F. Nie, D. Xu, J. Luo, Y. Zhuang, and Y. Pan. A multimedia retrieval framework based on semi-supervised ranking and relevance feedback. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(4):723–742, April 2012.

[200] Y. Yang, F. Wu, F. Nie, H. T. Shen, Y. Zhuang, and A. G. Hauptman. Web and personal image annotation by mining label correlation with relaxed visual graph embedding. *IEEE Transactions on Image Processing*, 21(3):1339–1351, March 2012.

[201] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, 1993.

[202] K. Yu, S. Yu, and V. Tresp. Blockwise supervised inference on large graphs. In *ICML Workshop on Learning with Partially Classified Training Data*, pages 40–46. 2005.

[203] S. X. Yu and J. Shi. Multiclass spectral clustering. In *Proceedings of IEEE International Conference on Computer Vision*, pages 313–319, 2003.

[204] K. Zhang, J. T. Kwok, and B. Parvin. Prototype vector machine for large scale semi-supervised learning. In *Proceedings of the 26th International Conference on Machine Learning*, pages 1233–1240, 2009.

[205] K. Zhang, I. W. Tsang, and J. T. Kwok. Improved nyström low-rank approximation and error analysis. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1232–1239, 2008.

[206] T. Zhang, A. Popescul, and B. Dom. Linear prediction models with graph regularization for web-page categorization. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 821–826, 2006.

[207] B. Zhao, F. Wang, and C. Zhang. Efficient maximum margin clustering via cutting plane algorithm. In *Proceedings of the 8th SIAM International Conference on Data Mining*, pages 751–762, 2008.

[208] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Scholkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems 16*, pages 321–328, 2003.

[209] D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Scholkopf. Ranking on data manifolds. In *Advances in Neural Information Processing Systems 16*, pages 169–176, 2003.

[210] X. Zhou, M. Belkin, and N. Srebro. An iterated graph laplacian approach for ranking on manifolds. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 877–885, 2011.

[211] X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2008.

[212] X. Zhu, Z. Ghahramani, and J. D. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 912–919, 2003.

[213] X. Zhu and A. B. Goldberg. *Introduction to Semi-Supervised Learning*. Morgan & Claypool Publishers, 2009.

[214] X. Zhu, J. S. Kandola, and Z. Ghahramani. Nonparametric transforms of graph kernels for semi-supervised learning. In *Advances in Neural Information Processing Systems 17*, 2004.

[215] X. Zhu, J. Lafferty, and Z. Ghahramani. Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. In *ICML Workshop on The Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, pages 58–65. 2003.

[216] X. Zhu and J. D. Lafferty. Harmonic mixtures: combining mixture models and graph-based methods for inductive and scalable semi-supervised learning. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, pages 1052–1059, 2005.