Chapter 2

Background

2.1 Introduction

In this chapter I provide an overview of existing research that is crucial for understanding the contributions made in this thesis. The chapter begins in Section 2.3 with an introduction to nearest neighbour (NN) search, why the problem is important and how it can be solved. This introduction is then followed by a discussion in Section 2.3 as to why a relaxed version of the problem is required, known commonly as approximate nearest neighbour (ANN) search. In Section 2.4, I describe a seminal method, Locality Sensitive Hashing (LSH), for solving the ANN search problem in a time constant in the number of data-points. The limitations of LSH are discussed and I use those drawbacks as a motivation for a review of a host of more recently proposed algorithms for ANN search that demonstrate a higher retrieval effectiveness on the task of image retrieval. I divide this latter part of the review into methods for binary quantisation (Section 2.5) and projection function learning (Section 2.6), mirroring the two stages of hashcode generation first introduced in Chapter 1.

2.2 Preliminaries and Notation Definition

This thesis adheres to the standard typography for vectors $\mathbf{x}$ (lowercase bold) and matrices $\mathbf{X}$ (uppercase bold). The ijth entry of matrix $\mathbf{X}$ is denoted by an uppercase, non-bold letter X_{ij}. Vectors $\mathbf{x} = [x_1, x_2 \ldots, x_N]^\intercal$ are assumed to be column vectors formed by stacking N scalar values. $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^\intercal$ signifies the stacking of the N column vectors $\left\{\mathbf{x}_i \in \mathbb{R}^D\right\}_{i=1}^{N}$ *row-wise* to form matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$. I use the notation $\mathbf{x}^c = \mathbf{X}_{\bullet c}$ to refer to the vector of elements in the c^{th} column of matrix $\mathbf{X}$. In a similar

manner $\mathbf{x}_r = \mathbf{X}_{r\bullet}$ denotes the vector of elements in the r^{th} row of matrix $\mathbf{X}$. Functions are indicated by lowercase, non-bold letters e.g $d(.,.)$. I summarise the notation used throughout this thesis in Table A.1 situated in Appendix A.

The related work described in this chapter all share the same problem definition. We are given a dataset consisting of N points $\mathbf{X} \in \mathbb{R}^{N \times D} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^\intercal$ where each data-point point $\mathbf{x}_i \in \mathbb{R}^D$ is a D-dimensional vector of real-valued features. The objective is to construct K hash functions $\left\{h_k : \mathbb{R}^D \rightarrow \{0,1\}\right\}_{k=1}^K$ the output of which can be concatenated as $[h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \ldots, h_K(\mathbf{x}_i)]$ to yield a binary embedding function $\left\{g_l : \mathbb{R}^D \rightarrow \{0,1\}^K\right\}$ that maps each data-point $\mathbf{x}_i$ to a K-bit binary hashcode $\mathbf{b}_i \in \{0,1\}^K$. For the embedding functions to be useful for nearest neighbour search we will require the bits to be selected in such a way that similar points $\mathbf{x}_i, \mathbf{x}_j$ will have similar hashcodes $\mathbf{b}_i, \mathbf{b}_j$, as measured by an appropriate distance function in the hashcode space such as the Hamming distance. I dedicate the remainder of this chapter to describing how similarity preserving hash functions are constructed by relevant models from the literature. A birds-eye-view of the structure of this chapter is shown in Figure 2.1.

While the hashing models discussed in this chapter are evaluated solely on image datasets, they are by no means restricted to this particular data-type. A powerful property of the discussed hashing models is their applicability to data whose instances can be represented as vectors of a certain dimensionality, and this includes text and audio data. We may find, however, that the relative performance of the models changes depending on the data-type of interest. For example, high dimensional and sparse textual vectors are expected to cause scaling issues for the eigendecomposition-based models described in this chapter, which work particularly well on the low-dimensional, dense feature vectors found in the field of Computer Vision. An investigation into how the described models perform on datasets of different types (text, audio, vision) would be an interesting avenue for future work.

2.3 Approximate Nearest Neighbour (ANN) Search

In this section I first formally define the problem of nearest neighbour (NN) search which I informally introduced in Chapter 1. I will then examine the relaxed version of NN search known as *approximate* NN search, the field upon which this thesis builds, and describe how it differs from alternative algorithms for solving the NN search problem.

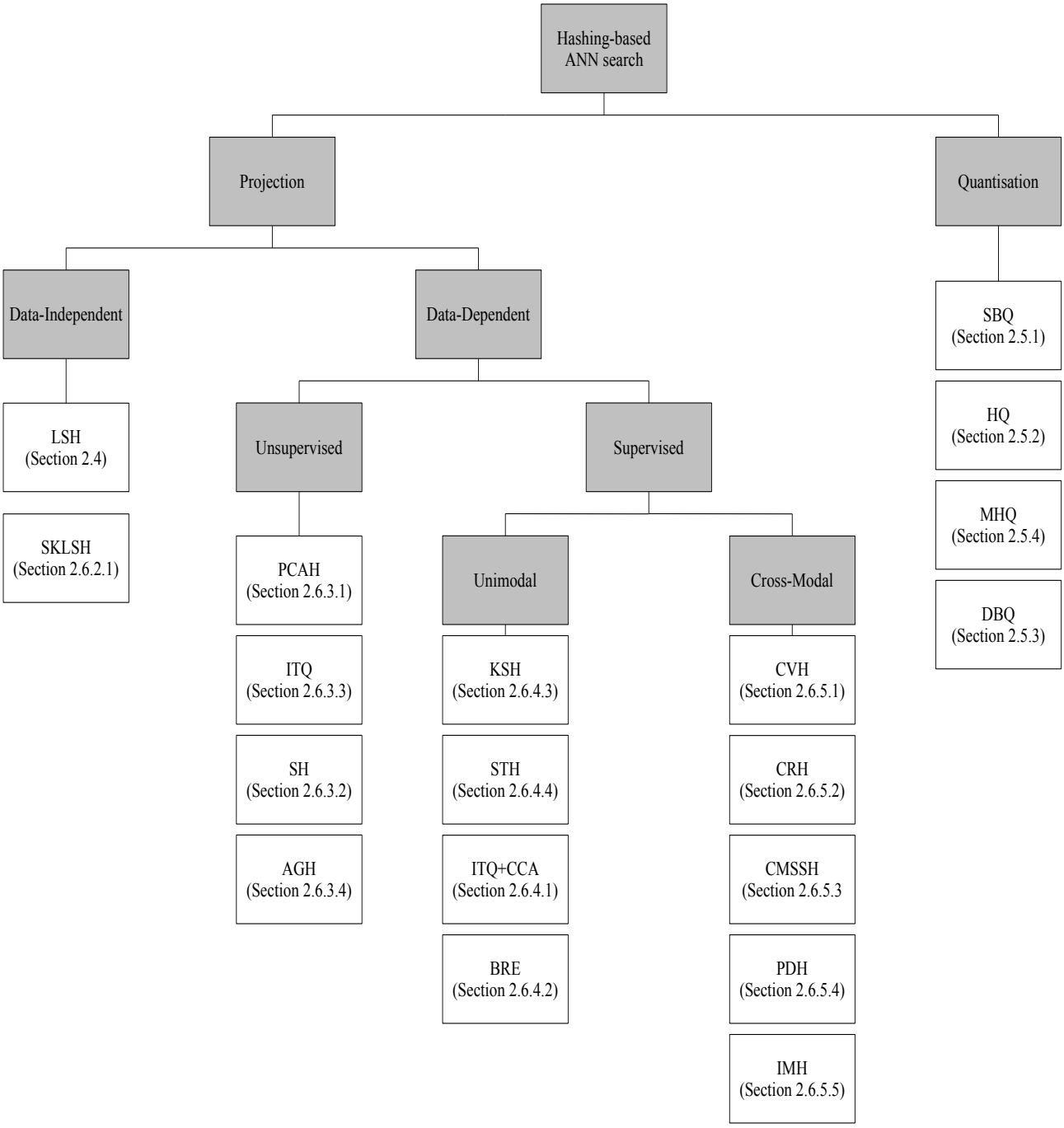


Figure 2.1: Overview of one possible categorisation of the field of hashing-based ANN search. The main categories are shown in the grey boxes while the actual models themselves are highlighted in white alongside their relevant section number in this chapter.

Nearest neighbour search can be defined as the problem of retrieving the closest data-point $NN(\mathbf{q})$ to a query $\mathbf{q} \in \mathbb{R}^D$ in a database of N data-points $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^\intercal$ where $\mathbf{x}_i \in \mathbb{R}^D$. The similarity between data-points is defined by a distance function of interest $\{d(.,.) : \mathbb{R}^D \times \mathbb{R}^D \rightarrow [0,1]\}$. This variant of the problem is also known as 1-NN search and is specified mathematically in Equation 2.1

$$NN(\mathbf{q}) = argmin_{\mathbf{x}_i \in \mathbf{X}} d(\mathbf{x}_i, \mathbf{q}) \tag{2.1}$$

It is straightforward to generalise this problem definition to return the closest K neighbours to the query. This variant is popularly referred to as k-NN search and is a fundamental component in a wide range of different machine learning methods including non-parametric kernel density estimation (Bishop (2006), Ulz and Moran (2013), Moran and Lavrenko (2014)). The distance function $d(.,.)$ between the data-points is typically computed using a generic distance metric such as the l_p-norm (Equation 2.2)

$$\begin{aligned} d_{pnorm}(\mathbf{x}_i, \mathbf{x}_j) = & \quad \|\mathbf{x}_i - \mathbf{x}_j\|_\rho \\ = & \quad \left(\sum_{k=1}^{D} |x_{ik} - x_{jk}|^\rho \right)^{\frac{1}{\rho}} \end{aligned} \tag{2.2}$$

The parameter $\rho \in \mathbb{R}_+$. Setting $\rho = 1$ yields the Manhattan distance and $\rho = 2$ gives the Euclidean distance while $\rho < 1$ introduces the Minkowski family of fractional distances. The cosine distance presented in Equation 2.3 is another popular distance metric for NN search that has proven particularly effective for document retrieval (Manning et al. (2008), Ravichandran et al. (2005))

$$d_{cosine}(\mathbf{x}_i, \mathbf{x}_j) = 1 - \frac{\sum_{k=1}^{D} x_{ik} x_{jk}}{\sqrt{\sum_{k=1}^{D} x_{ik}^2} \sqrt{\sum_{k=1}^{D} x_{jk}^2}} \tag{2.3}$$

We will also come across the Hamming distance extensively in this thesis as it is the de-facto metric for comparing binary strings (Equation 2.4)

$$d_{hamming}(\mathbf{b}_i, \mathbf{b}_j) = \sum_{k=1}^{D} \delta[b_{ik} \neq b_{jk}] \tag{2.4}$$

The function $\delta(.) = 1$ if its argument is true, and 0 otherwise. The Hamming distance therefore counts the number of corresponding dimensions (bits) that are *not equal* in the two hashcodes.

These generic distance metrics do not adapt to the distribution of the data, measuring the distances between data-points in the same way regardless of the specifics of the dataset. In applying both metrics in practice we implicitly hope that the resulting distances correlate well with the specific notion of similarity required for the domain. For example, in the field of image annotation that the Euclidean distance between the feature representation of two images can tease apart an image of a cat from that of a dog. In many cases this is an unrealistic assumption that leads to low retrieval effectiveness (Kulis (2013); Moran and Lavrenko (2014)). Distance metric learning is an active research field dedicated to learning distance metrics tuned to a specific dataset. These methods typically learn a scaling and a rotation of the data so that the Euclidean distance in the transformed space correlates better with, for example, class-based supervision. Perhaps unsurprisingly learnt metrics have been shown to greatly improve the quality of NN retrieval over and above their non data-adaptive counterparts such as the l_ρ-norm (Kulis (2013)). I pick up this thread again in Section 2.6 where I reveal how this important idea of data-dependent distance functions has inspired recent developments in the field of hashing-based ANN search.

To search for NNs to a query we need to construct a data-structure or algorithm that takes our selected notion of distance and retrieves data-points that are close to the query under that specific distance metric. Brute force search is a straightforward algorithm for solving the nearest neighbour search problem with any desired distance metric. In brute-force search the distance to every data-point in the database is computed and the data-point(s) with the smallest distance to the query returned as the nearest neighbour(s). The advantages of brute force search are its simplicity of implementation and its guarantee that the closest nearest neighbours will eventually be retrieved. However, exhaustively comparing the query to every data-point in the database gives a linear $O(ND)$ time complexity which quickly makes brute force search intractable for nearest neighbour search across datasets with many data-points (N) and a moderate to high dimensionality (D). In this situation a more informed approach to the nearest neighbour search problem is required.

The generality and importance of nearest neighbour search, described in detail in the motivation for this thesis in Chapter 1, ensures that the problem remains an active research area within many scientific disciplines including Information Retrieval (IR) and Computer Vision. Efficient multidimensional indexing data-structures for NN search have been proposed for data-points of low-dimensionality (usually $D \leq 10$), with some of the more well known examples of this kind being the KD-tree (Bentley

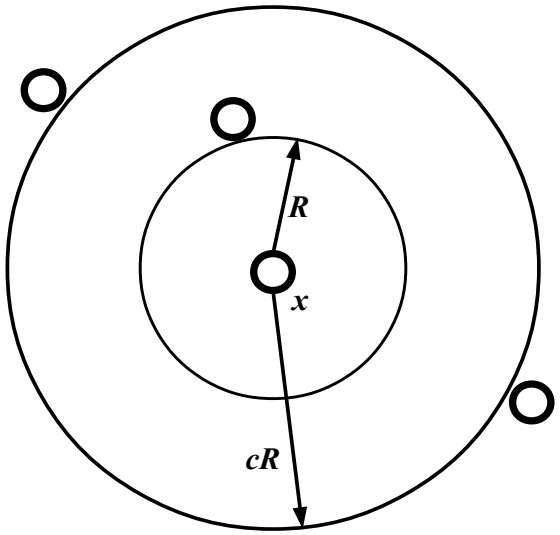


Figure 2.2: The (c,R)-approximate NN problem: in many applications it is acceptable to return a data-point (indicated with the circles) within a distance cR of the query point $\mathbf{x}$, where R is the distance to the exact NN and the approximation factor $c > 1$.

(1975)), quad-tree (Finkel and Bentley (1974)), X-tree (Berchtold et al. (1997)) and SR-tree (Katayama and Satoh (1997)). Unfortunately it has been shown that methods relying on a space partitioning or clustering of the input feature space can do no better than brute-force search in high dimensions (Weber et al. (1998)). This result severely limits the applicability of these algorithms to image and document collections where it is not uncommon to find feature representations with hundreds, thousands or indeed millions of dimensions. The impossibility of retrieving *exact* nearest neighbours in sub-linear time in high dimensional spaces is one particular incarnation of the well-known *"curse of dimensionality"* (Minsky and Papert (1969)).

Algorithms for *approximate* nearest neighbour search circumvent the curse of dimensionality by relaxing the need for an optimal (exact) solution to the problem, in return for a substantially improved bound on the query time. In many practical scenarios, for example detecting a large number of object classes (Dean et al. (2013)) or matching variable sized sets of features (Grauman and Darrell (2007)), retrieving sub-optimal nearest neighbours is an entirely acceptable compromise for a greatly reduced query time. In the theoretical computer science literature the problem is commonly referred to as the (c,R)-approximate NN decision problem. In this problem definition, we are happy to accept a nearest neighbour within distance cR of the query, where c is an approximation factor ($c > 1$) and R is the distance to the exact NN (Figure 2.2). The (c,R)-approximate NN decision problem is formalised in Andoni and Indyk (2008);

Petrovic (2012) and defined in Definition 2.3.1:

Definition 2.3.1. Randomised c-approximate R-near neighbour problem: given a set of N data-points in a D dimensional space, return each cR-nearest neighbour of the query data-point $\mathbf{q}$ with probability 1-δ, where $\delta > 0, R > 0$.

The approximation factor c effectively determines the degree of sub-optimality in the returned nearest neighbours that we are willing to tolerate. The greater the value of c the more distant the returned nearest neighbour might be from the optimal nearest neighbour, with the advantage of a reduction in the query time. This clear trade-off between effectiveness and efficiency lies at the heart of effective algorithms for solving the (c,R)-approximate nearest neighbour problem.

The R-near neighbour reporting problem (Andoni and Indyk (2008); Petrovic (2012)) is similar but without the approximation factor c (Definition 2.3.2)

Definition 2.3.2. Randomised R-near neighbour problem: given a set of N data-points in a D dimensional space, return each R-nearest neighbour of the query data-point $\mathbf{q}$ with probability 1-δ, where $\delta > 0, R > 0$

In Section 2.4, I will introduce Locality Sensitive Hashing (LSH), a family of algorithms that provide a concrete method for solving these approximate nearest neighbour search decision problems in constant time per query.

2.4 Locality Sensitive Hashing (LSH)

The objective for any successful model for hashing-based ANN search is to pre-process the database $\mathbf{X} \in \mathbb{R}^{N \times D}$ so that at query-time nearest neighbours can be found more efficiently than a simple brute force search over the entire database. In this section I introduce the core ideas behind Locality Sensitive Hashing (LSH) (Indyk and Motwani (1998)), one of the most influential algorithms for ANN search and the first to provide a sub-linear time solution to the randomised c-approximate R-near neighbour problem. LSH has found wide-application in vision problems, from recognising 100,000 object classes on a single machine (Dean et al. (2013)), to pose estimation (Shakhnarovich et al. (2003)), bag-of-words indexing (Chum et al. (2008)) and shape matching (Grauman and Darrell (2004)). The hashing models I introduce later in this literature review (Sections 2.5-2.6) can all be thought of as extensions of LSH that try and overcome certain disadvantages with the original algorithm. Given the central importance of LSH, I

therefore spend a considerable proportion of this review in discussing the algorithm in detail. I will begin by giving a general overview of LSH, independent of the similarity metric of interest. In Section 2.4.1 I will then discuss a concrete instantiation of LSH for the inner product similarity that will be of central importance in this thesis.

The key idea behind LSH is to pre-process the database by assigning hashcodes to each data-point in such a way that data-points that are closer in $\mathbb{R}^D$ under some distance metric $\left\{d(.,.) : \mathbb{R}^D \times \mathbb{R}^D \rightarrow [0,1]\right\}$ have a higher probability of colliding in the same hashtable bucket than data-points that are much further apart in $\mathbb{R}^D$. LSH therefore transforms nearest neighbour search into the process of examining the contents of a small set of hashtable buckets, which is likely to be many times more efficient than exhaustive brute force search over every data-point. The question then arises as to how LSH generates hashcodes (i.e. the indices into the hashtable buckets) which are the same for data-points that are "close" in the original feature-space. To achieve this property, LSH uses what is known as locality sensitive hash functions $\left\{h_k : \mathbb{R}^D \rightarrow U\right\}$ that map $\mathbb{R}^D$ to some universe U (for example, binary bits or positive integers). The locality sensitive hash functions are drawn uniformly at random from a hash function family $\mathcal{H}$ (Definition 2.4.1)

Definition 2.4.1. Locality sensitive hash function family: a hash function family $\mathcal{H}$ is deemed (R, cR, P_1, P_2) sensitive if for any two data-points $\mathbf{p}$, $\mathbf{q} \in \mathbb{R}^D$:

$$\begin{aligned} &if \quad d(\mathbf{p},\mathbf{q}) \leq R \quad then \quad Pr_{\mathcal{H}}(h(\mathbf{p}) = h(\mathbf{q})) \geq P_1 \\ &if \quad d(\mathbf{p},\mathbf{q}) \geq cR \quad then \quad Pr_{\mathcal{H}}(h(\mathbf{p}) = h(\mathbf{q})) \leq P_2 \end{aligned}$$

where $Pr_{\mathcal{H}}(h(\mathbf{p}) = h(\mathbf{q}))$ refers to the probability that two data-points hash to the same value given a hash function $h(.)$ chosen uniformly at random from $\mathcal{H}$. If a locality sensitive hash function family is to be useful for nearest neighbour search then we require $P_1 > P_2$ and $c > 1$. In other words there should be a *high* probability P_1 of two data-points $\mathbf{p} \in \mathbb{R}^D, \mathbf{q} \in \mathbb{R}^D$ close by to each other (i.e. $d(\mathbf{p},\mathbf{q}) \leq R$) in $\mathbb{R}^D$ colliding in the same hashtable bucket. Conversely there should be a *low* probability P_2 of more distant data-points (i.e. $d(\mathbf{p},\mathbf{q}) \geq cR$) colliding in the same hashtable bucket. In this way the output of a hash function $h(.)$ chosen uniformly at random from $\mathcal{H}$ is intimately tied to the spatial arrangement of the data-points in $\mathbf{X}$ as measured under a distance metric of interest $\left\{d(.,.) : \mathbb{R}^D \times \mathbb{R}^D \rightarrow [0,1]\right\}$. Ideally we would like that $P_1 = 1$ and $P_2 = 0$ so that all data-points that are within $d(\mathbf{p},\mathbf{q}) \leq R$ of the query map to the same hashtable bucket and all data-points with distance $d(\mathbf{p},\mathbf{q}) \geq cR$ map to a different hashtable bucket. Note, the case $R < d(\mathbf{p},\mathbf{q}) < cR$ remains unaddressed,

but nevertheless R and cR can be made close at the expense of making P_1 and P_2 undesirably close (Rajaraman and Ullman (2011)).

Fortunately it is possible to construct a wide variety of useful hash function families that have the property that $P_1 > P_2$ and $c > 1$. For example, locality sensitive hash function families have so far been introduced for many distance functions of prime interest such as the L_p distance in $\mathbb{R}^D$ for $p \in [0,2)$ (Datar et al. (2004)), cosine distance (inner product similarity) (Charikar (2002)), Jaccard distance (Broder (1997)) and L_2-norm on the unit hypersphere (Terasawa and Tanaka (2007)). Choosing the locality sensitive hash function family $\mathcal{H}$ is an important decision that needs to be considered when implementing an LSH system in practice. In a similar way that selecting an appropriate distance function for brute force search is application dependent, so too is choosing a locality sensitive hash function family. For example, the hash function family for the *inner product similarity*, which draws its hash functions uniformly from a unit sphere, has proven to be successful for detecting new events in high-volume document streams (Petrovic (2012), Osborne et al. (2014)). I will expand on this particular hash function family in more detail in Section 2.4.1. The LSH hash function family for the Euclidean distance (Datar et al. (2004)), which randomly samples hash functions from a D dimensional zero mean unit variance Gaussian distribution, has also found wide applicability in applications such as pose estimation (Shakhnarovich et al. (2006); Matei et al. (2006)). This hash function family relies on the *Johnson-Lindenstrauss* lemma (Johnson and Lindenstrauss (1984)) as a guarantee that there will be limited distortion to the pairwise distances in the lower-dimensional embedding space.

The usefulness of any locality sensitive hash function family for nearest neighbour search is dependent on the *gap* between P_1 and P_2 which dictates the collision probabilities between points in the range $[0,R]$ in which the R-near neighbours are to be found and (cR,∞). If the gap between P_1 and P_2 is small then a query will have a similar probability of mapping to the hashtable bucket of a distant data-point as it will be to a close-by data-point. Without a sufficient difference between P_1 and P_2 the quality of nearest neighbour search under LSH will be poor with a high number of false positives and false negatives. The gap between P_1 and P_2 can be *amplified* by concatenating together K randomly selected hash functions to create an embedding function into a K dimensional space. This multidimensional embedding function is given by Equation 2.5

$$g_l(\mathbf{q}) = [h_1(\mathbf{q}), h_2(\mathbf{q}), \ldots, h_K(\mathbf{q})] \tag{2.5}$$

where $g_l(.)$ is drawn uniformly at random from function family $\mathcal{G} = \left\{\mathbb{R}^D \rightarrow U^K\right\}$ and $h_k \in \mathcal{H}$. This concatenation of K hash functions increases the gap between P_1 and P_2, amplifying the difference between the probabilities of collisions between nearby and far data-points. For an embedding function $g_l(.)$ the probability $P(g_l(\mathbf{q}) = g_l(\mathbf{p}))$ that the hashcodes for any two distant data-points $\mathbf{p} \in \mathbb{R}^D, \mathbf{q} \in \mathbb{R}^D$ with $d(\mathbf{p},\mathbf{q}) \geq cR$ will match is given by $P_2^{'} = P_2^K$. This reduction in the number of false positives with increasing K is the underlying motivation for using multiple bits in a hashcode: as K increases there is a gradually lower probability that distant data-points will collide in the same hashtable bucket as the query $\mathbf{q}$. However, increasing K also reduces the probability of collision between nearby data-points (by $P_1^{'} = P_1^K$), and so while the precision increases through elimination of false positives we will also suffer a decrease in recall due to the introduction of false negatives. For a judicious choice of K, if $P_1 > P_2$, it is possible to keep probability $P_1^{'}$ bounded significantly away from zero, while moving probability $P_2^{'}$ close to zero (Rajaraman and Ullman (2011)).

In practice for a large enough hashcode length K, we might find that very few close by data-points $(d(\mathbf{p},\mathbf{q}) < R)$ collide in the same hashtable bucket as no data-points are likely to share an identical hashcode. The number of buckets in a single hashtable grows at an exponential rate (2^K) as the hashcode length is increased and so many of these buckets will be empty for a large enough setting of K. The other LSH parameter is the number L of embedding functions sampled from $\mathcal{G}$, with each embedding function indexing into one of L independent hashtables. The value of L can be increased to counteract the lower level of recall that arises from a longer hashcode length K. The probability that two hashcodes will collide in the same hashtable bucket for *at least one* hashtable is then given by the expression $P_1^{''} = (1 - (1 - P_1^K)^L)$. Even though using multiple hashtables will increase probabilities $P_1^{''}$ and $P_2^{''}$, it is possible to set L so as to increase probability $P_1^{''}$ towards one, while also keeping $P_2^{''}$ bounded significantly away from one (Rajaraman and Ullman (2011)). Therefore, the parameters K and L can be set in combination so as to cause probability $P_1^{''}$ to be close to one, while moving $P_2^{''}$ close to zero, which is the property we seek for an ideal locality sensitive hash function (an illustration of this effect for various settings of K and L is shown in Figures 2.3-2.4). Of course, the higher the values of K and L the greater the computation time required for the actual hashing.

The setting of L and K permits the practitioner trade-off of the precision and recall achieved while choosing an appropriate overall computational cost. One possible strategy for setting L and K is to use the probabilistic bounds on the failure probability

Algorithm 1: LSH PRE-PROCESSING STEP (PETROVIC (2012))

Input: Data-points $\mathbf{X} \in \mathbb{R}^{N \times D}$, L embedding functions $[g_1(.), \ldots, g_L(.)]$ with $g_l(.) = \{h_{l1}(.), \ldots, h_{lK}(.)\}$, $h_{lk}(.)$ selected uniformly from family $\mathcal{H}$

Output: Data-points indexed into the buckets of L hashtables H

```
for i ← 1 to L do
    for j ← 1 to |X| do
        Insert x_j into bucket H[i][g_i(x_j)]
    end
end
return H
```

offered by LSH. The setting of L and K can be found by firstly deciding on an acceptable probability $\delta < (1 - P_1^K)^L$ of LSH failing to find an R-nearest neighbour with a specified similarity (P_1) to the query. The setting of L guaranteeing the failure probability δ for a given hashcode length K is then given by $L \geq \lceil log(\delta)/log(1 - P_1^K) \rceil$. The E^{2}LSH[1] package recommends choosing the hashcode length K to minimize the mean query time for all data-points a dataset. Some LSH implementations attempt to eliminate the need to choose these parameters altogether, see for example LSH-forest (Bawa et al. (2005)). For the practitioner, Petrovic (2012) provide an enlightening discussion on how the best fitting L and K parameters were chosen for an LSH-based event detection system. This system was successfully used for real-time detection, tracking, and monitoring of automatically discovered events in social media streams (Osborne et al. (2014)).

Having chosen the desired hash function family $\mathcal{H}$ and the setting of K and L there are two final steps to using LSH for nearest neighbour search: *pre-processing*, in which the database points are hashed using the L multidimensional embedding functions $\{g_l : \mathbb{R}^D \rightarrow \{0,1\}\}_{l=1}^{L}$ into the buckets of L hashtables $g_l(\mathbf{p})$ for $l = \{1 \ldots L\}$; and *querying*, where the query is also hashed using the same hash functions and the nearest neighbours retrieved from the colliding hashtable buckets $\{g_1(\mathbf{q}), \ldots, g_l(\mathbf{q})\}$. Typically, the distance (e.g. Euclidean or cosine) from the query to each of the data-points in this candidate list of nearest neighbours is then computed and any data-points $> R$ discarded. The pre-processing step is presented in Algorithm 1 while the querying process is presented in Algorithm 2. The presentation of the pre-processing and

[1] http://www.mit.edu/~andoni/LSH/

Algorithm 2: LSH QUERYING STEP (PETROVIC (2012))

Input: Query $\mathbf{q} \in \mathbb{R}^D$, Database $\mathbf{X} \in \mathbb{R}^{N \times D}$, L functions $[g_1(.), \ldots, g_L(.)]$ with $g_l = \{h_{l1}(.), \ldots, h_{lK}(.)\}$ and h_k selected uniformly at random from a hash function family $\mathcal{H}$, hashtables H

Output: The set S of R (strategy 2) nearest neighbours of $\mathbf{q}$

```
for i ← 1 to L do
    for j ← 1 to |H[i][g_i(q)]| do
        Retrieve next data-point x_j from bucket H[i][g_i(q)]
        if (d(x_j, q) < R) then                    // Query strategy 2
            Put x_j into retrieved set S
        end
    end
end
return S
```

querying algorithms has been inspired by a similar specification in Petrovic (2012). There are two LSH querying strategies and both are directly related to the two variants of the approximate nearest neighbour decision problem presented in Definitions 2.3.1-2.3.2. The strategy presented in Algorithm 2 solves the R-near neighbour reporting problem (Definition 2.3.2) as all data-points in the colliding hashtable buckets are examined. In the unlikely worst case scenario this latter strategy may cause the search to examine every data-point in the database. The randomised c-approximate R-near neighbour problem (Definition 2.3.1) is solved by stopping the search after the first $L^{'} = 3L$ data-points have been retrieved. This strategy comes with an $O(L)$ bound on the query time.

2.4.1 LSH with Sign Random Projections

In this dissertation I will be primarily interested in the locality sensitive hash function family for the *inner product similarity* which traditionally has been used as a baseline for comparison by existing research in the learning to hash literature. The inner product similarity is defined in Equation 2.6.

$$\begin{aligned} d(\mathbf{p},\mathbf{q}) &= \sum_{k=1}^{D} p_k q_k \\ &= \mathbf{p}^T \mathbf{q} \end{aligned} \tag{2.6}$$

Equation 2.6 can also be interpreted as the cosine similarity between two L_2 *normalised* vectors mapped on the unit sphere. The cosine similarity measures the closeness between two data-points based on the angle (θ) between their respective vectorial representations in the D-dimensional space, which could be a GIST descriptor (Oliva and Torralba (2001)) for an image or a TF-IDF vector for a document. As the angle between the two vectors widens their cosine similarity decreases, and vice-versa. The locality sensitive hash function family for the cosine similarity $\mathcal{H}_{cosine}$ is formulated by intersecting the space with K hyperplanes drawn randomly from a multidimensional Gaussian distribution with mean zero and unit variance. Depending on what side of a hyperplane a data-point falls, its hashcode is appended with either a '0' or a '1'. The intuition is as follows: the greater the angle between a query $\mathbf{q} \in \mathbb{R}^D$ and a database point $\mathbf{p} \in \mathbb{R}^D$ the more probable it is that the space between the vectors will be partitioned by a randomly drawn hyperplane. The greater the angle, the more often the intervening space will be partitioned by random hyperplanes and the lower the number of bits the hashcodes will share in common. We have achieved the desired property: the output of a hash function (randomly drawn hyperplane) is less likely to match as the angle between two data-points is increased.

More formally, a randomly drawn hash function h_k from $\mathcal{H}_{cosine}$ has the specification given in Equation 2.7

$$\begin{aligned} h_k(\mathbf{q}) &= \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{q})) \\ &= q_k(p_k(\mathbf{q})) \end{aligned} \tag{2.7}$$

where sgn is the sign function adjusted so that $sgn(0) = -1$, $\mathbf{w}_k \in \mathbb{R}^D$ denotes the normal vector of hyperplane $\mathbf{h}_k$. I denote as $\left\{p_k : \mathbb{R}^D \to \mathbb{R}\right\}$ the *projection function* (a dot product in this case) that maps a data-point onto a randomly chosen dimension. Here $\left\{q_k : \mathbb{R} \to \{0,1\}^B\right\}$ denotes the *quantisation function* (thresholding at zero with the sign function in this case) that converts the projection of a data-point into one or more binary bits. Equation 2.7 is the mathematical procedure for determining the position of a data-point with respect to a separating hyperplane, with a '0' or a '1' output depending on the side. With the hash function as specified in Equation 2.7, Goemans and Williamson (1995) showed that the probability of a collision is given as

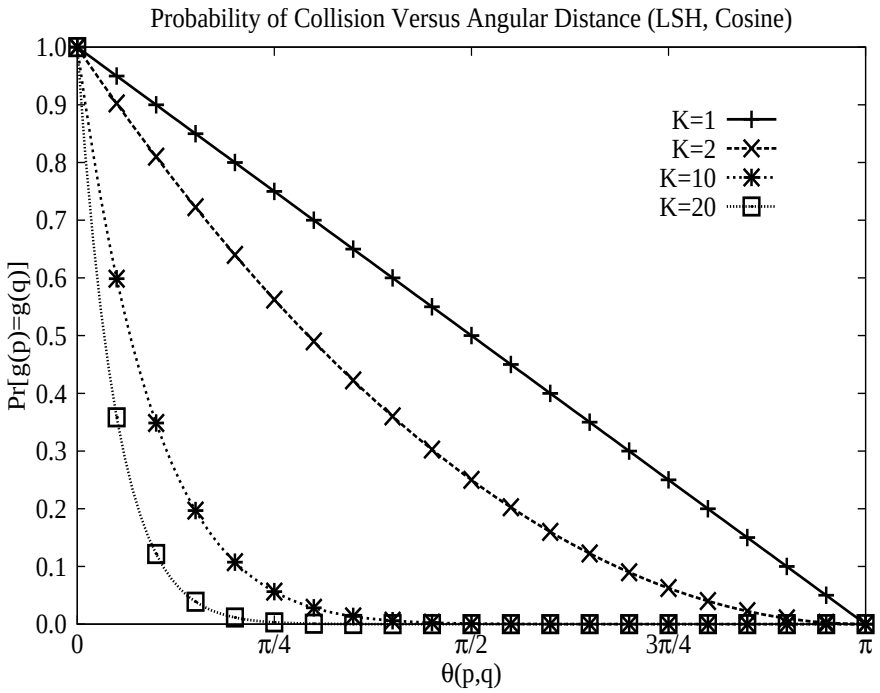


Figure 2.3: Visualising the probability of two hashcodes $g(\mathbf{p}), g(\mathbf{q})$ matching against the length of the hashcode key ($K = 1, 2, 10, 20$). The hash function family is $\mathcal{H}_{cosine}$. As the hashcode length becomes longer the two data-points must be close together (in terms of angle) in order for their collision probability to be high. This is intuitive because if we draw more hyperplanes (bits) there is a greater chance of the two data-points falling on different sides of at least one of the hyperplanes, particularly if the data-points are spaced further apart. This figure is adapted from a similar chart in Petrovic (2012).

in Equation 2.8.

$$Pr_{\mathcal{H}_{cosine}}(h(\mathbf{p}) = h(\mathbf{q})) = 1 - \frac{\theta(\mathbf{p}, \mathbf{q})}{\pi} \tag{2.8}$$

Equation 2.8 operationalises our earlier intuition of there being a lower collision probability with a greater angle θ (in radians) when applying a hash function of the form given in Equation 2.7. $\mathcal{H}_{cosine}$ is therefore a $(R, cR, 1 - R/\pi, 1 - cR/\pi)$-sensitive hash function family, where the angular distance R is measured in radians. The amplification strategy I discussed in Section 2.4 for increasing the P_1, P_2 gap works equally as well for $\mathcal{H}_{cosine}$. As before, choosing the length of K with an appropriate setting of the number of hashtables L is crucial for retrieval effectiveness and efficiency in an end-application. I illustrate the locality sensitive nature of $\mathcal{H}_{cosine}$ in Figures 2.3-2.4. In these figures I change the value of K and L and observe the effect on the probability of a collision occurring in the hashtables.

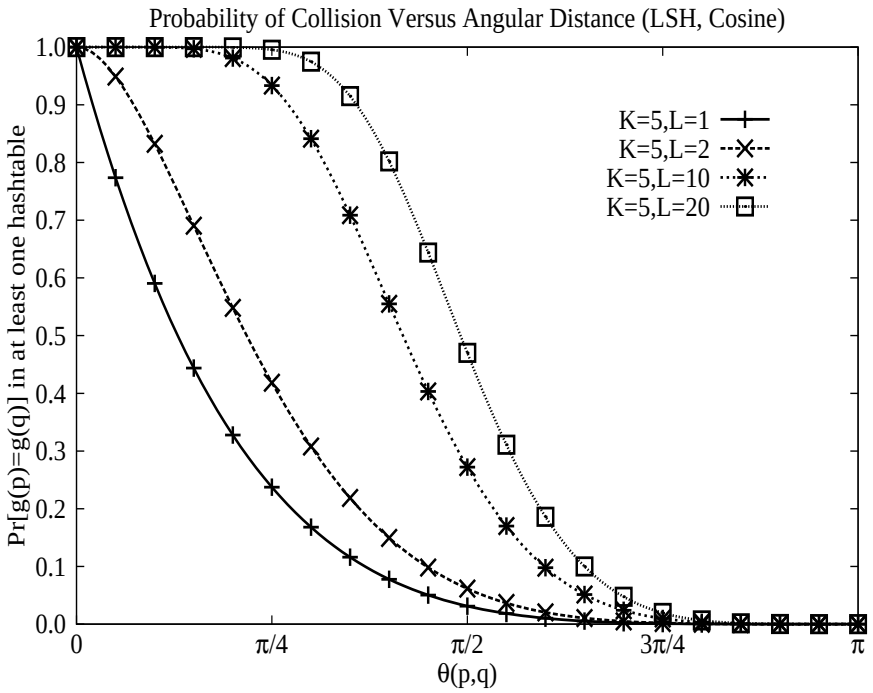


Figure 2.4: Probability of collision resulting from varying the number of hashtables ($L = 1, 2, 10, 20$) for fixed hashcode length $K = 5$. The hash function family is $\mathcal{H}_{cosine}$. We can see that as the number of hashtables increase there is a higher probability of two data-points being close by colliding in at least one of the hashtable buckets, and a very low probability of more distant data-points colliding. This is expected as with more hashtables we are more likely to find a situation where none of the randomly generated hyperplanes separate the two data-points. This figure is adapted from a similar chart in Petrovic (2012).

The query time complexity is also dependent on L and K. For a single query data-point the retrieval cost can be characterised by $O(KDL) + O(1) + O(\frac{NDL}{2^K})$. This is made up of the *hashing time* (time spent generating the hashcodes) $O(KDL)$, the *lookup time* ($O(1)$ for a good hashtable implementation) and the *candidate test time* ($O(\frac{NDL}{2^K})$), which is the time taken to exhaustively compute the distance from the query to the colliding data-points and assuming a uniform distribution of data-points to buckets. As K is increased the hashing time will increase but the candidate test time will fall as the hash functions become more selective. Increasing the number of hashtables will increase both the hashing time and candidate test time with the benefit of increasing the probability that close-by data-points will collide in at least one bucket of a hashtable.

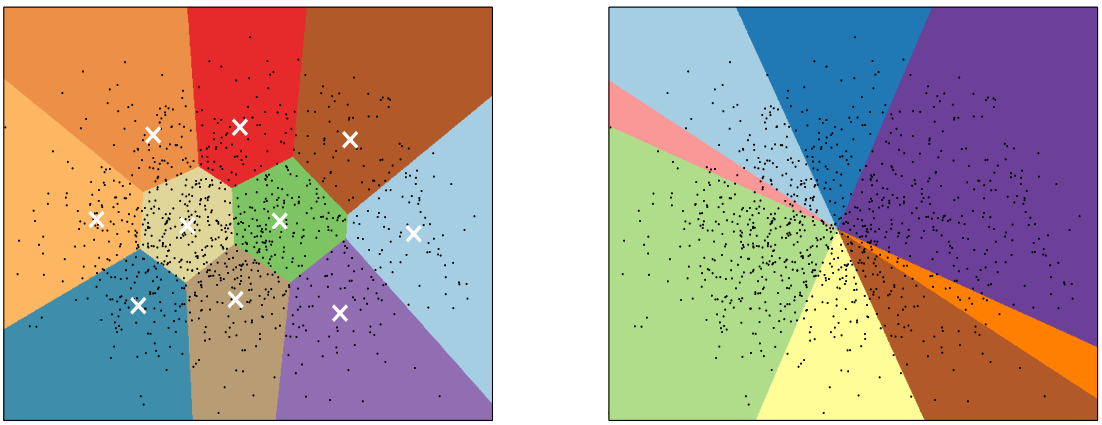
Equation 2.7 provides the foundation upon which the rest of this review and indeed thesis is formed. I propose novel formulations for defining the projection function

$\{p_k : \mathbb{R}^D \rightarrow \mathbb{R}\}$ and the quantisation function $\left\{q_k : \mathbb{R} \rightarrow \{0,1\}^B\right\}$ so that retrieval effectiveness is maximised, while also maintaining scalability of the algorithms to large datasets. More specifically I will firstly challenge the notion that a sign function is an optimal quantisation strategy for converting the real-valued projection in Equation 2.7 to binary (Section 2.5 and Chapters 4-5) and secondly, that randomly sampled hyperplanes produce optimal hashcodes (Section 2.6 and Chapter 6). On the latter point, it is well known that LSH hyperplanes tend to lack discriminative power with many hyperplanes (bits) and many hashtables being required for an adequate level of precision and recall (Wang et al. (2012)). This inefficiency is due to their *data-independent* nature where the hashing hyperplanes are generated without regard to the data distribution. This issue has recently stimulated research into hashing methods that learn hash functions adapted to the distribution of the data. I discuss state-of-the-art data-driven hash functions in Sections 2.5-2.6.

2.5 Quantisation for Nearest Neighbour Search

In this section I review previous related research in the field of binary quantisation for hashing-based ANN search. We saw in Section 2.4.1 that one of the two crucial steps in generating LSH-based binary hashcodes involves converting real-valued projections into binary bits. In this section I study in depth recently proposed algorithms for reducing the information loss resulting from the discretisation of real-numbers into binary, and specifically methods that attempt to do better than simply taking the sign function in Equation 2.7. I will attempt to put on a firm grounding exactly what I mean by better later in this section. Generally speaking, quantisation refers to the process of reducing the cardinality of a representation (such as numbers on the real-line) to a finite and discrete set of symbols (e.g. binary bits). Quantisation has been extensively studied particularly within the field of information theory (Gray and Neuhoff (2006)) and has also found wide engineering application given the impossibility of storing and manipulating numeric values to infinite precision. This review will necessarily only focus on quantisation methods that have been specifically used in hashing-based ANN search methods.

Two main categories of quantisation have been proposed for nearest neighbour search: *scalar* and *vector* quantisation, which are differentiated by whether the input and output of the quantisation is a scalar or a vector quantity. Scalar quantisation is frequently applied to quantise the real-values (projections) resulting from the dot prod-



(a) **Vector Quantisation** (b) **Scalar Quantisation**

Figure 2.5: In the context of nearest neighbour search two variations on quantisation are typically employed: vector quantisation (Figure (a)) partitions the feature space into Voronoi cells (Jegou et al. (2011)). Centroids are marked with a white cross while data-points are shown as black dots. The distance between query and database points is computed by determining the distance to their closest centroids. In contrast, scalar quantisation (Figure (b)) is frequently used to binarise a real-valued projection resulting from a dot product of a data-point with a hyperplane normal vector. The space is partitioned with multiple such hyperplanes and each usually contribute 1-bit to the final hashcode. The hashcode is effectively the index of the polytope-shaped region containing the associated data-point. The data-points appearing in this example are a 2D PCA projection of the CIFAR-10 image dataset.

uct of the feature vector of each data-point onto the normal vectors to a set of random hyperplanes partitioning the feature space (Figure 2.5). As we will discover in Section 2.5.1, each dot product yields a scalar value which is then subsequently quantised into binary $(0/1)$ by thresholding. In contrast, for vector quantisation, the feature representation of a data-point is associated with its nearest centroid, out of a set of centroids discovered by the k-means algorithm (Lloyd (1982)). In this way each input vector (data-point) is represented by a much smaller set of codebook vectors (centroids). K-means divides the space into Voronoi regions forming a more flexible data-driven partitioning. I illustrate the partitions formed by vector quantisation and a hyperplane based scalar quantisation in Figure 2.5. I will not cover vector quantisation any further in this thesis, but I will mention in passing that it has been found to be more effective for nearest neighbour search in Computer Vision tasks due to lower reconstruction error (Jegou et al. (2011)). The downside is the need to store a lookup table at test time to

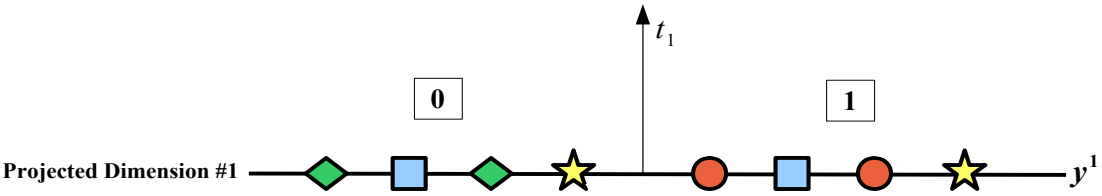


Figure 2.6: Single bit quantisation (SBQ) uses one threshold $t_1 \in \mathbb{R}$ to binarise a projected dimension: projected values (indicated by the coloured shapes) lower than the threshold (on the left) are assigned a '0' bit, while projected values greater than the threshold (on the right) are assigned a '1' bit.

read off inter-cluster Euclidean distances using the centroid indices[2]. This computation has been found to be 10-20 times slower than the Hamming distance computation between the binary hashcodes on standard datasets (He et al. (2013)). Scalar quantisation needs no such *decoding* step as the distances are computed directly from the hashcodes, an advantage that has proved beneficial in applications such as mobile product search (Feng (2012)). Only very recently have researchers attempted to combine the strengths of both approaches in a unified quantisation algorithm: the reader is encouraged to consult He et al. (2013) and references therein for more detail on interesting work in this direction.

In the context of hashing-based ANN search a scalar quantiser $\left\{q_k : \mathbb{R} \rightarrow \{0,1\}^B\right\}$ maps a real-valued projection $y_i \in \mathbb{R}$ to a single (Section 2.5.1) or multi-bit (Sections 2.5.3-2.5.4) binary codeword $\left\{\mathbf{c}_i = \{0,1\}^B | \mathbf{c}_i \in \mathcal{C}, i \in \{1,2,\ldots,T+1\}\right\}$ with T denoting the number of quantisation thresholds, B denoting the number of bits per projected dimension and $\mathcal{C}$ is the binary codebook. In this dissertation I follow Kong et al. (2012) by defining a *projected dimension* $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$ as the set of real-valued projections $\left\{y_i^k \in \mathbb{R}\right\}_{i=1}^{N_{trd}}$ of all data-points $[\mathbf{x}_1, \mathbf{x}_2 \ldots \mathbf{x}_{N_{trd}}]$ for a given hyperplane $\mathbf{h}_k$, where a projection $y_i^k \in \mathbb{R}$ is obtained by a dot product $y_i^k = \mathbf{w}_k^\intercal \mathbf{x}_i$. The quantisation function q_k binarises each projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$ independently by positioning one or more thresholds at selected points along the dimension. Projected values falling into a given thresholded region are assigned the codeword of that region. A simple illustration of this process is shown in Figure 2.6 where the projected dimension is shown as a line with a sampling of data-points (indicated by the coloured shapes) superimposed. In this toy example the quantiser uses a single threshold to partition the

[2]Another advantage of partitioning the space with hyperplanes is the exponential number (2^K) regions formed using just K-hyperplanes. Vector quantisation would require 2^K centroids for a similar partition granularity. Jegou et al. (2011) show how 2^K centroids can be learnt efficiently for large K using *product* quantisation.

Method	Encoding	Optimisation	Thresholds (T)	Complexity	Section
SBQ	0/1	Mean thresholding	1	$O(1)$	2.5.1
HQ	00/01/10/11	Spectral partitioning	1 and 2	$O(CN^{+}_{trd})$	2.5.2
DBQ	00/10/11	Squared error	2	$O(N_{trd}\log N_{trd})$	2.5.3
MHQ	NBC	1D K-means	3+	$O(2^{B}N_{trd})$	2.5.4

Table 2.1: Existing single (SBQ) and multi-threshold (HQ, DBQ, MHQ) quantisation schemes categorised along the three main dimensions of variability. NBC stands for Natural Binary Encoding and is explained in Section 2.5.4. C is the number of anchor points, N_{trd} is the number of training data-points, N^{+}_{trd} is the number of training data-points with positive projected value for the given projected dimension and B is the number of bits per projected dimension. Time complexity is for positioning thresholds along a single projected dimension.

projected dimension into two disjoint regions. The projections falling in the region below the threshold are given the codeword '0' while the projections falling in the region above the threshold are assigned the codeword '1'. The codebook for this example is $\{\mathbf{c}_i = \{0,1\}\,|\mathbf{c}_i \in \mathcal{C}, i \in \{1,2\}\}$. In this way, quantisation transforms projected values that live on the real-line into a discrete set of codewords from the specified codebook $\mathcal{C}$. More formally I denote as $\mathbf{t}_k = [t_1, t_2, \ldots, t_T]$ the set of threshold positions along a single projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$ where $t_i \in \mathbb{R}$ with $t_1 \leq t_2 \ldots \leq t_T$. The two extremities of a projected dimension are denoted as $t_0 = -\infty$ and $t_{T+1} = +\infty$. The thresholds $\{t_i \in \mathbb{R}\}_{i=1}^{T}$ partition a given projected dimension into T+1 disjoint regions $\mathbf{r}_i = \{y_j | t_{i-1} < y_j \leq t_i, y_j \in \mathbf{y}_k\}$ where $i \in \{1 \ldots T+1\}$. Most existing scalar quantisation schemes use $T = 2^B - 1$ thresholds for a budget of B bits per projected dimension. Each of the resulting $T+1$ thresholded regions $\{\mathbf{r}_i \subset \mathbf{y}^k\}_{i=1}^{T+1}$ are associated with a unique codeword $\mathbf{c}_i \in \mathcal{C}$.

The retrieval effectiveness resulting from quantisation is greatly affected by the selected codebook and the positioning of the quantisation thresholds (Moran et al. (2013a); Kong et al. (2012); Kong and Li (2012a)). The encoding scheme must ensure that the relative distances between the data-points in the input space are maintained in the resulting binary hashcodes. For example, if two data-points are distant in the original feature space then their assigned codewords should also be distant in Hamming space, and vice-versa for close data-points. Ideally the encoding scheme for the thresholded regions should impart a smooth, gradual increase in Hamming distance as

the distance between the data-points in the original feature space increases. Correct positioning of the thresholds is also important as a threshold dividing an area dense in true nearest neighbours will result in related data-points falling into different regions and being assigned different codewords, increasing the Hamming distance of their resulting hashcodes. If the threshold positions and/or the encoding scheme are sub-optimal then the related data-points will be assigned hashcodes with large Hamming distance severely limiting the effectiveness of any hashing algorithm using that quantisation scheme. The state-of-the-art quantisation algorithms I review in this section all propose an encoding scheme and threshold optimisation algorithm that seek to faithfully preserve the relative distances between data-points in their corresponding binary hashcodes.

The previous paragraph hints at three key properties that can be used to distinguish and categorise existing methods of scalar quantisation[3]: the encoding scheme used to assign symbols to each thresholded region, the manner in which the threshold positions are determined, that is, whether a learning scheme is employed to optimise the positioning, and the number of thresholds allocated per dimension. In Table 2.1 I provide an overview of existing quantisation methods as categorised along these three dimensions of variability. In the following sections I describe these existing quantisation schemes in detail. Specifically, in Section 2.5.1 I introduce the traditional method of Single Bit Quantisation (SBQ) and in Sections 2.5.2-2.5.4 I describe the more recent multi-threshold quantisation schemes: Hierarchical Quantisation (HQ), Double Bit Quantisation (DBQ) (Section 2.5.3) and Manhattan Hashing Quantisation (MHQ) (Section 2.5.4).

2.5.1 Single Bit Quantisation (SBQ)

Single Bit Quantisation (SBQ) is the method of binarisation employed in the vast majority of existing hashing methods. A single threshold t_k partitions a projected dimension $\mathbf{y}^k$ into two regions, with a '0' bit assigned to projected values lower than the threshold and a '1' bit assigned to projected values equal to or greater than the threshold. More formally given a set of k hyperplane normal vectors $[\mathbf{w}_1 \ldots \mathbf{w}_K]$, the k^{th} hashcode bit for a data-point $\mathbf{x}_i$ is generated by SBQ as given in Equation 2.9.

[3] I will use the term *quantisation* to refer to scalar quantisation throughout the remainder of this thesis.

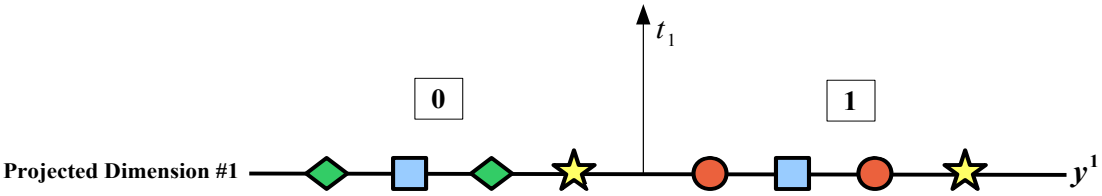


Figure 2.7: Single Bit Quantisation (SBQ) uses one threshold t_1 to binarise a projected dimension: projected values (indicated by the coloured shapes) lower than the threshold (on the left) are assigned a 0 bit, while projected values greater than the threshold (on the right) are assigned a 1 bit.

$$h_k(\mathbf{x}_i) = \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k)) \tag{2.9}$$

In this quantisation scheme each hyperplane contributes one bit towards the hash-code for a data-point. The data is typically *zero-centred* and the projected dimensions are thresholded at the mean ($t_k = \frac{1}{N_{trd}} \sum_{i=1}^{N_{trd}} \mathbf{w}_k^\intercal \mathbf{x}_i$). For zero centered data this equates to the threshold being placed directly at zero ($t_k = 0$). No learning mechanism is used to optimise the placement of the threshold in SBQ, although in some cases it might be placed at the median of the data distribution rather than at the mean. Given the absence of a threshold optimisation step SBQ is a computationally inexpensive operation requiring $O(1)$ time[4] for threshold learning and $O(1)$ time for encoding a novel query data-point. SBQ is further illustrated with a toy example in Figure 2.7.

The multi-threshold quantisation algorithms I describe in Section 2.5.2-2.5.4 all seek to overcome a fundamental limitation of SBQ which arises from the use of a single threshold for binarisation. In some cases SBQ may assign different bits to data-points that are located close together along a projected dimension, while data-points that are located much further apart can be assigned the same bits (Kong et al. (2012); Kong and Li (2012a); Moran et al. (2013a,b)). This is contrary to the fundamental objective of hashing in which close-by data-points should be assigned identical bits. Given this, it should be expected that this limitation of SBQ can lead to reduced retrieval effectiveness. I experimentally confirm that this is the case in Chapter 4. This problem with SBQ is easily illustrated by considering a hypothetical true nearest neighbour data-point pair in Figure 2.8. In this diagram the data-points a and b indicated by the yellow stars are very close to the threshold but lie on opposite sides. Even though both are close in the projected space they are assigned opposite bits, increasing the Ham-

[4]This increases to $O(N_{trd})$ time for threshold learning if the median is used as the threshold.

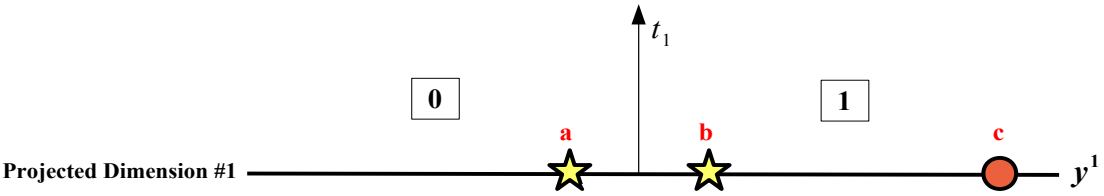
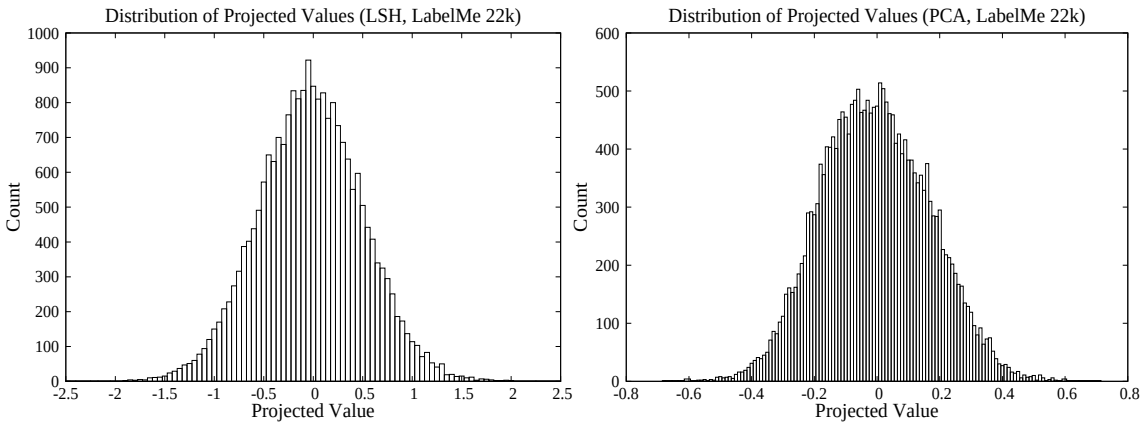


Figure 2.8: The problem with Single Bit Quantisation (SBQ): true nearest neighbours such points a, b are assigned different bits despite being close together along the projected dimension. Conversely points b, c located above the threshold, which are non-nearest neighbours, are assigned the same bit (1), even though they are more distantly spaced along the projected dimension.

ming distance of their hashcodes. The hash function, by placing the projections of the pair a, b nearby along the projected dimension, has indicated that the corresponding data-points were close together (as deemed by our distance metric of interest) in the original feature space[5]. Despite this, SBQ assigns both opposite bits, effectively destroying the neighbourhood structure encoded in the projections. The opposite is true for the data-points b, c indicated by the yellow star and red circle located above the threshold. These non-nearest neighbours are far apart along the projected dimension, indicating that the hash function determined they were more distant in the original feature space. Nevertheless, SBQ has assigned the same bit to both data-points b, c ensuring their resulting hashcodes are closer together in terms of Hamming distance.

Unfortunately, this issue is likely to surface often in practice given that vanilla SBQ places a threshold directly at zero and the highest point density along a projected dimension also usually happens to be in the region around zero. This pattern is true for many projection functions commonly employed in practice (Figure 2.9). Partitioning a projected dimension into multiple regions, and assigning each region a multi-bit encoding is an effective means of overcoming this issue with SBQ. The *modus-operandi* of all multi-threshold quantisation schemes, including my novel contributions presented in Chapter 4 and Chapter 5, is maximal preservation of the neighbourhood structure encoded in the projected dimension through a multi-bit codebook and a threshold optimisation algorithm. I will now discuss one of the first proposed multi-threshold quantisation schemes in Section 2.5.2.

[5]We are of course relying here on the hash function placing data-points that are close-by in the original feature space close by along the projected dimension. Randomised LSH projection functions guarantee this in expectation while other projection functions seek to explicitly learn the hyperplanes so that related data-points are encouraged to have similar projections. I cover the latter data-dependent methods in Section 2.6.



(a) **Locality Sensitive Hashing (LSH)** (b) **Principal Component Analysis (PCA)**

Figure 2.9: Distribution of projected values for two randomly chosen LSH (Figure (a)) and PCA (Figure (b)) projected dimensions on the LabelMe 22k image dataset (Torralba et al. (2008)). The images in this dataset are encoded as GIST features (Oliva and Torralba (2001)). The region of highest projected value density is typically around zero, as is clearly the case for these two dimensions. The Double Bit Quantisation algorithm (DBQ) Kong and Li (2012a) explicitly avoids placing a threshold close to zero as, given the high density of points in that region, this is likely to separate many true nearest neighbours. DBQ is described in Section 2.5.3.

2.5.2 Hierarchical Quantisation (HQ)

Liu et al. (2011) were the first to introduce the concept of multi-threshold quantisation for hashing in which a single hyperplane contributes multiple bits to the hashcode. Their quantisation algorithm, dubbed Hierarchical Quantisation (HQ), was introduced as a means of quantising projections resulting from their Anchor Graph Hashing (AGH) model. It uses only $\lfloor K/2 \rfloor$ of the available hyperplanes, assigning two bits per hyperplane. I describe the projection learning component of AGH in detail in Section 2.6.3.4, while in this section I focus solely on the quantisation algorithm assuming that we have already obtained the desired projected dimensions $\left\{\mathbf{y}^k \in \mathbb{R}^{N_{trd}}\right\}_{k=1}^{K}$. HQ consists of two steps performed in a sequence: in the first step traditional SBQ is applied (Section 2.5.1), thresholding a given projected dimension $\mathbf{y}^k$ at zero ($t_1 = 0$). Step one produces the first bit of the double-bit encoding for a hyperplane. In the second step the projected dimension is quantised again, this time using two new thresholds (t_2, t_3) that further partition the two regions formed by SBQ at the first step.

The two thresholds (t_2, t_3) are jointly optimised so as to minimise the number of

related data-points falling on opposite sides of the dividing threshold resulting from applying SBQ in the first step. Both quantisation steps are illustrated in Figure 2.10. Liu et al. (2011) formulate the threshold optimisation as a graph partitioning problem on the graph Laplacian $\mathbf{L} = \mathbf{I} - \hat{\mathbf{S}}$ of the low-rank *approximate* adjacency matrix $\hat{\mathbf{S}}$[6]. The neighbourhood structure is encoded by $\hat{\mathbf{S}}$, where $\hat{S}_{ij} > 0$ indicates that i and j are neighbours, and $\hat{S}_{ij} = 0$ indicates they are not. $\hat{\mathbf{S}}$ is approximate in the sense that it is not constructed by computing the N_{trd}^2 distances between the N_{trd} data-points, but rather is constructed from an anchor graph $\mathbf{Z}$, a sparse matrix that holds the similarities between the N_{trd} data-points and a small set of C anchor points where $C \ll N_{trd}$ (Equation 2.10):

$$Z_{ij} = \begin{cases} \dfrac{exp(-d^2(\mathbf{x}_j, \mathbf{c}_i)/\gamma)}{\sum\limits_{i' \in \langle j \rangle} exp(-d^2(\mathbf{x}_j, \mathbf{c}_{i'}))/\gamma)} & \text{if } i \in \langle j \rangle \\ 0 & otherwise \end{cases} \tag{2.10}$$

where γ is the kernel bandwidth, $\{d(.,.) : \mathbb{R}^D \times \mathbb{R}^D \to [0,1]\}$ is any distance function of interest such as the L_2-norm and $\langle j \rangle \in \{1 \ldots R\}$ are the indices of the $R \ll C$ nearest anchors to $\mathbf{x}_j$ under distance metric $d(.,.)$. The C anchor points $\{\mathbf{c}_i \in \mathbb{R}^D\}_{i=1}^{C}$ can be found by running the k-means algorithm over data-points in a training dataset. Using the anchor graph, the adjacency matrix $\hat{\mathbf{S}}$ can be computed as $\hat{\mathbf{S}} = \mathbf{Z}\Lambda^{-1}\mathbf{Z}^\intercal$ where $\Lambda_{kk} = \sum_{i=1}^{N_{trd}} Z_{ik}$. This latter expression approximates the affinity between data-points $\mathbf{x}_i, \mathbf{x}_j$ as the inner product between their individual affinities to the C centroids. Compared to the full adjacency matrix, $\hat{\mathbf{S}}$ is sparse and low-rank which brings computational advantages when extracting the required graph Laplacian eigenvectors (Section 2.6.3.4). Liu et al. (2011) construct an eigenvalue problem involving $\mathbf{Z}$ to solve for the K graph Laplacian eigenvectors $\mathbf{y}^k$ of the approximate adjacency matrix $\hat{\mathbf{S}}$. In the context of AGH these eigenvectors are the projected dimensions that are thresholded to yield the hashcodes of the training data-points (Equation 2.11).

$$h_k(\mathbf{x}_i) = \begin{cases} \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{x}_i - t_2)), & \text{if } \mathbf{w}_k^\intercal \mathbf{x}_i \geq 0 \\ \frac{1}{2}(1 + sgn(-\mathbf{w}_k^\intercal \mathbf{x}_i + t_3)), & \text{if } \mathbf{w}_k^\intercal \mathbf{x}_i < 0 \end{cases} \tag{2.11}$$

Binarising a graph Laplacian eigenvector has the effect of partitioning the graph

[6]The rank of a matrix is the number of linearly independent rows or columns.

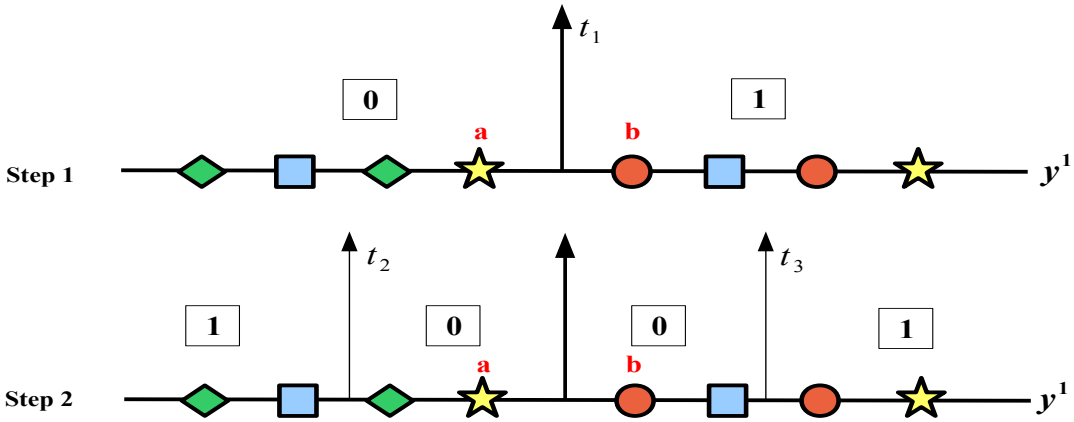


Figure 2.10: Illustration of the Hierarchical Quantisation (HQ) algorithm of Liu et al. (2011). I use the data-points a (yellow star) and b (red circle) as examples. The quantisation proceeds in two steps: in the first step SBQ thresholds the projected dimension into two regions generating the first bit of the two bit encoding for the data-points. For data-point a this is '0' and for data-point b this is '1'. In the second step the regions formed by SBQ are further partitioned with t_2, t_3 using a dynamic threshold optimisation algorithm. Data-point a is assigned '0' again and b is now assigned '0', yielding hashcodes '00' and '10', respectively. Nearby data-points falling on opposite sides of the SBQ threshold in Step 1 are therefore more likely to have the same bit assigned in Step 2, which is the case for our two example data-points. This is the central tenet of the HQ algorithm.

encoded by $\hat{\mathbf{S}}$ into two groups, with each of the K eigenvectors forming a different bi-partitioning of the graph (Shi and Malik (2000)). In the context of hashing-based approximate nearest neighbour search, the hope is that many of these graph cuts will result in true nearest neighbours being within the same partition. The eigenvectors with the highest eigenvalues are generally unreliable and do not produce an effective partitioning of the graph (Shi and Malik (2000)). This observation motivates the creation of the two-step quantisation algorithm of Liu et al. (2011) in which the lowest eigenvectors are responsible for generating most of the hashcode bits. If we denote $\mathbf{y}^{k+}$ as the positive projected values of projected dimension $\left\{\mathbf{y}^{k+} \in \mathbb{R}^{N_{trd}} | y_i^k \geq t_1\right\}$, and $\left\{\mathbf{y}^{k-} \in \mathbb{R}^{N_{trd}} | y_i^k < t_1\right\}$ the negative projected values, the objective of the second level threshold optimisation is to minimise Equation 2.12

$$\begin{aligned} &\text{argmin}_{t_2,t_3} \quad \mathbf{f}^\intercal \mathbf{L} \mathbf{f} \\ &\text{where } \mathbf{f} = \begin{bmatrix} \mathbf{y}^+ - t_2 \mathbf{1}_1 \\ -\mathbf{y}^- + t_3 \mathbf{1}_2 \end{bmatrix} \\ &\text{subject to } \mathbf{1}^\intercal \mathbf{f} = 0 \end{aligned} \tag{2.12}$$

Intuitively the objective function is attempting to position thresholds t_2, t_3 so that two conditions are met. Firstly, connected nodes in $\hat{\mathbf{S}}$, that is true nearest neighbours, stay as close as possible along the projected dimension (as $\mathbf{f}^\intercal \mathbf{L} \mathbf{f} = \sum_{ij} \hat{S}_{ij} (f_i - f_j)^2$), and secondly, there is an equal number of opposing bits ('0' and '1's) when the projected dimension is binarised with thresholds t_2, t_3. This latter balance constraint ($\mathbf{1}^\intercal \mathbf{f} = 0$) has previously been shown to maximise the information captured by the bits (Weiss et al. (2008)). Liu et al. (2011) demonstrate that by setting to zero the derivatives of Equation 2.12, the two thresholds t_2, t_3 minimising Equation 2.12 can be computed in closed form.

While Liu et al. (2011) find their multi-threshold quantisation algorithm more effective than SBQ it suffers from lack of generality to other projection functions, being entirely tied to the quantisation of graph Laplacian eigenvectors. The computational complexity of solving for t_2, t_3 is approximately $O(CN^+_{trd})$ (Liu et al. (2011)), where N^+_{trd} denotes the number of positive projected values constituting $\mathbf{y}^{k+}$. Given the learnt thresholds the time taken to generate a bit for a novel query point is $O(1)$. HQ effectively front loads the available bit budget onto the lowest graph Laplacian eigenvectors. Liu et al. (2011) demonstrate that only using half the number of eigenvectors and assigning each with two bits yields higher retrieval effectiveness than using all available eigenvectors and assigning each a single bit. Typically, the intrinsic dimension of many datasets of interest is low and so the lower graph Laplacian eigenvectors (those with the smallest eigenvalues) are likely to capture most of the neighbourhood structure, with the higher eigenvectors being more informative of the input space. This insight is the seed that sparked the recent interest in multi-threshold quantisation algorithms for ANN search and is the inspiration behind our novel contributions in Chapter 5. I will examine subsequent research contributions in this area in chronological order continuing next to the Double Bit Quantisation (DBQ) algorithm of Kong and Li (2012a).

2.5.3 Double Bit Quantisation (DBQ)

Double-Bit Quantisation (DBQ) (Kong and Li (2012a)) allocates two thresholds per projected dimension and assigns two bits to the three resulting thresholded regions. Unlike HQ (Section 2.5.2) it has the useful advantage of not being tied to any specific projection function. For a K-bit hashcode DBQ therefore only uses $\lfloor K/2 \rfloor$ of the number of hyperplanes as SBQ. DBQ uses the binary encoding scheme illustrated in Figure 2.11 which ensures that any two adjacent regions only differ by unit Hamming distance. This property is crucial if the relative distances between the data-points are to be maintained in the underlying binary encoding, a key requirement for maximising retrieval effectiveness. DBQ also proposes a novel adaptive thresholding algorithm for finding an optimal setting of the quantisation thresholds t_1, t_2. Given a particular instantiation of the quantisation thresholds t_1, t_2, three sets $\mathbf{r}_1$, $\mathbf{r}_2$, $\mathbf{r}_3$ are defined each containing the projected values falling within the corresponding region, that is: $\mathbf{r}_1 = \{y_i | y_i \leq t_1, y_i \in \mathbf{y}_k\}$, $\mathbf{r}_2 = \{y_i | t_1 < y_i \leq t_2, y_i \in \mathbf{y}_k\}$, $\mathbf{r}_3 = \{y_i | t_2 < y_i, y_i \in \mathbf{y}_k\}$. The objective function $\mathcal{J}_{dbq}$ of DBQ is to minimise the sum of squared Euclidean distances of the projected values falling within the three thresholded regions (Equation 2.13)

$$\mathcal{J}_{dbq}(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) = \sum_{y_i \in \mathbf{r}_1} (y_i - \mu_1)^2 + \sum_{y_j \in \mathbf{r}_2} (y_j - \mu_2)^2 + \sum_{y_l \in \mathbf{r}_3} (y_l - \mu_3)^2 \tag{2.13}$$

where μ_i denotes the mean of the projected values in region $\mathbf{r}_i$. As the area of highest point density along a projected dimension is in the region of zero (Figure 2.9), DBQ avoids placing a threshold at zero by setting $\mu_2 = 0$ enforcing the property that $t_1 < 0$ and $t_2 > 0$. Given this $\mathcal{J}_{dbq}$ can then be simplified as in Equation 2.16 (Kong and Li (2012a))

$$\begin{aligned}
\mathcal{J}_{dbq}(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) &= \sum_{y_i \in \mathbf{y}_k} y_i^2 - 2 \sum_{y_i \in \mathbf{r}_1} y\mu_1 + \sum_{y_i \in \mathbf{r}_1} \mu_1^2 - 2 \sum_{y_l \in \mathbf{r}_3} y\mu_3 + \sum_{y_l \in \mathbf{r}_3} \mu_3^2, && (2.14) \\
&= \sum_{y_i \in \mathbf{y}_k} y_i^2 - |\mathbf{r}_1|\mu_1^2 - |\mathbf{r}_3|\mu_3^2, && (2.15) \\
&= \sum_{y_i \in \mathbf{y}_k} y_i^2 - \frac{(\sum_{y_i \in \mathbf{r}_1} y_i)^2}{|\mathbf{r}_1|} - \frac{(\sum_{y_i \in \mathbf{r}_3} y_i)^2}{|\mathbf{r}_3|} && (2.16)
\end{aligned}$$

where $|\mathbf{r}_i|$ is the number of projected values (data-points) in region $\mathbf{r}_i$. Given that $\sum_{y_i \in \mathbf{y}_k} y_i^2$ is a constant minimising $\mathcal{J}_{dbq}$ is equivalent to maximising $\mathcal{J}'_{dbq}$ (Equation 2.17).

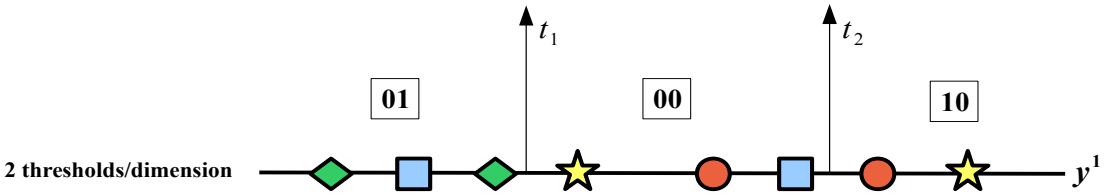


Figure 2.11: Double Bit Quantisation allocates two thresholds t_1, t_2 to binarise a projected dimension. The resulting three thresholded regions are assigned the two-bit encoding shown. This encoding ensures that adjacent regions are only separated by a Hamming distance of 1.

$$\mathcal{I}'_{dbq}(\mathbf{r}_1,\mathbf{r}_3) = \frac{(\sum_{y_i\in\mathbf{r}_1} y_i)^2}{|\mathbf{r}_1|} + \frac{(\sum_{y_j\in\mathbf{r}_3} y_j)^2}{|\mathbf{r}_3|} \tag{2.17}$$

The objective function $\mathcal{I}'_{dbq}$ is maximised by the adaptive thresholding strategy presented in Algorithm 3. Algorithm 3 initialises the thresholds t_1, t_2 to values close to zero, and then gradually moves both thresholds apart: t_1 is moved towards negative infinity ($-\infty$) while t_2 is moved towards positive infinity ($+\infty$). The objective function $\mathcal{I}'_{dbq}$ is evaluated at every projected value along the projected dimension. In tandem to this the algorithm attempts to maintain the invariant that the mean of region $\mathbf{r}_2$ is zero i.e. $\mu_2 = 0$ (Line 9), which ensures that neither threshold partitions the densest region of the projected dimension located around zero. The thresholds are moved while maintaining this property by gradually shifting data-points from regions $\mathbf{r}_1$ and $\mathbf{r}_3$ into $\mathbf{r}_2$ (Lines 8-13): if the sum of projected values in $\mathbf{r}_2$ is below zero then a positive projected value from $\mathbf{r}_3$ is moved into $\mathbf{r}_1$ to increase the sum towards zero, and vice-versa. The objective function $\mathcal{I}'_{dbq}$ is then computed (Line 15) on the new regions $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3$. If there is an increase in $\mathcal{I}'_{dbq}$ the thresholds t_1, t_2 are updated to be the largest projected values in $\mathbf{r}_1$ and $\mathbf{r}_2$, which now define the new boundaries between the regions. The algorithm terminates when all data-points have been moved into region $\mathbf{r}_2$. Note that as all data-points along the projected dimension are exhaustively examined DBQ guarantees to find t_1, t_2 that lead to the global maximum of $\mathcal{I}'_{dbq}$.

The implicit assumption made by DBQ is that the hash functions will minimise the squared Euclidean distance between true nearest neighbours in the low-dimensional projected space, which equates to the projected values of any two true nearest neighbours having low squared Euclidean distance along a given projected dimension. If this assumption is correct then a clustering of the one-dimensional projected dimension based on a squared error criterion will result in more true nearest neighbours being

Algorithm 3: DOUBLE BIT QUANTISATION (KONG AND LI (2012A))

Input: Projected values $\mathbf{y}^k \in \mathbb{R}^N$ resulting from projection onto normal vector $\mathbf{w}_k \in \mathbb{R}^D$ of hyperplane $\mathbf{h}_k \in \mathbb{R}^D$

Output: Optimised thresholds $t_1 \in \mathbb{R}, t_2 \in \mathbb{R}$

$\mathbf{r}_1 \leftarrow \{y_i | y_i \leq 0, y_i \in \mathbf{y}^k\}$
$\mathbf{r}_2 \leftarrow \emptyset$
$\mathbf{r}_3 \leftarrow \{y_i | y_i > 0, y_i \in \mathbf{y}^k\}$
Sort (ascending) projected-values in $\mathbf{r}_1$
Sort (ascending) projected-values in $\mathbf{r}_3$
$J_{max} \leftarrow 0$
$i \leftarrow 1$
while $\mathbf{r}_1 \neq \emptyset$ *or* $\mathbf{r}_3 \neq \emptyset$ **do**
    if $(sum(\mathbf{r}_2) \leq 0)$ **then**
        $\mathbf{r}_2 \leftarrow \mathbf{r}_2 \cup min(\mathbf{r}_3)$ // Remove minimum value in $\mathbf{r}_3$
    else
        $\mathbf{r}_2 \leftarrow \mathbf{r}_2 \cup max(\mathbf{r}_1)$ // Remove maximum value in $\mathbf{r}_1$
    end
    $i \leftarrow i + 1$
    $J \leftarrow \mathcal{J}'_{dbq}(\mathbf{r}_1, \mathbf{r}_3)$ // Equation 2.17
    if $(J > J_{max})$ **then**
        $t_1 \leftarrow max(\mathbf{r}_1)$
        $t_2 \leftarrow max(\mathbf{r}_2)$
        $J_{max} \leftarrow J$
    end
end
return t_1, t_2

assigned similar hashcodes (given that they will end up in the same thresholded region) versus an entirely random threshold setting. While Kong and Li (2012a) demonstrate that this is a reasonable assumption in practice, I will show in Chapter 4 that it is far from optimal and significantly improved retrieval effectiveness can be attained with a semi-supervised objective that does not entirely rely on the quality of the hash function that produces the projections. The threshold learning time complexity of DBQ is $O(N_{trd} \log N_{trd})$ which arises from the pre-processing step that sorts the projected val-

ues in regions $\mathbf{r}_1$ and $\mathbf{r}_3$ (Lines 4 and 5). DBQ has $O(1)$ time complexity when using the learnt thresholds to generate a bit for a novel query data-point.

2.5.4 Manhattan Hashing Quantisation (MHQ)

A primary disadvantage of both HQ and DBQ are their arbitrary restriction to two bits per projected dimension. Kong et al. (2012) explored the effect of introducing more bits per projected dimension in their Manhattan Hashing Quantisation (MHQ) model, which until the multi-threshold quantisation algorithms I present in Chapters 4-5, constituted the state-of-the-art in the field. MHQ permits an arbitrary allocation of bits, where for B bits per projected dimension $2^B - 1$ thresholds are used to partition the dimension into disjoint regions. To generate a hashcode of length K MHQ uses $\lfloor K/B \rfloor$ hyperplanes. In a similar manner to DBQ, MHQ introduces a new encoding scheme and threshold optimisation algorithm, both designed to increase the preservation of the relative distance between the data points in the resulting hashcodes. I describe both contributions in this section.

The binary encoding scheme advocated by MHQ is illustrated in Figure 2.12. Each region is encoded using natural binary encoding (NBC). The NBC codebook for each region is simply obtained by proceeding from left-to-right along the projected dimension starting with the region closest to $t_0 = -\infty$ and converting the integer index starting at zero (and incremented by one for each region) to its corresponding NBC. For example, in Figure 2.12 to obtain the NBC for the third region from the left for the bottom most projected dimension we convert the integer '2' to '010'. Under the constraint of Hamming distance it is clear that the NBC encoding scheme *does not* preserve the relative distance between the data-points. This is easily seen if we examine the encoding for the eight regions induced by setting seven thresholds along a projected dimension (Figure 2.12). The encoding for the fourth region from the left is 011, while the encoding for the adjacent region to the right is 100. The Hamming distance between these two regions is 3 despite both being adjacent, while the Hamming distance between region eight (111) - which is much further along the projected dimension - is only one. In effect this means that any data-points which are projected far apart along the projected dimension - and which are presumably far apart in the original feature space - will be much closer together in the Hamming space, than data-points that were projected close by along the projected dimension. Hashcodes generated with this encoding and compared using Hamming distance will yield poor quality hashcodes and low retrieval ef-

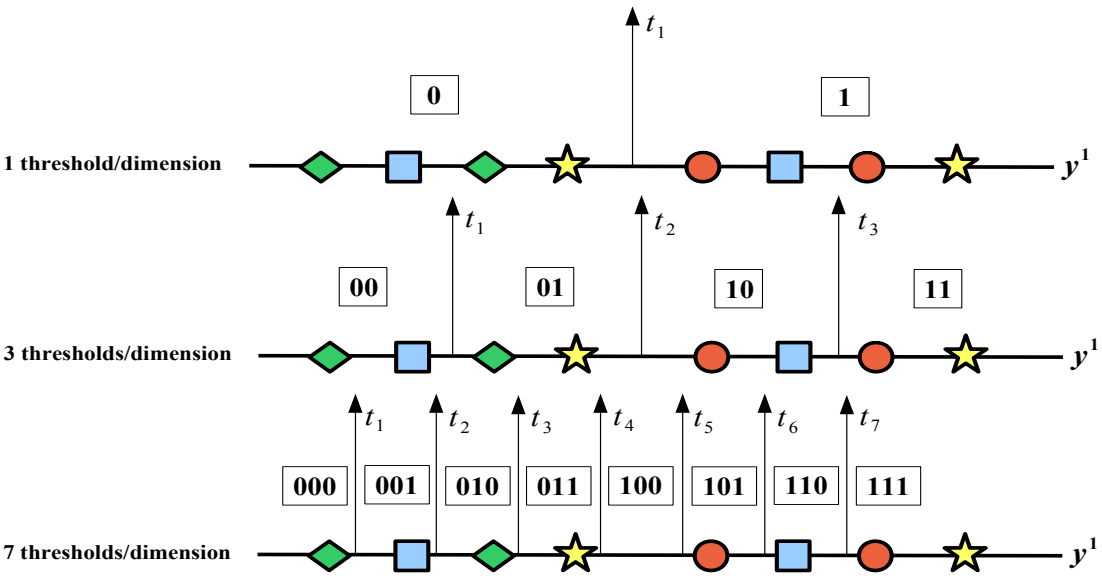


Figure 2.12: Manhattan Hashing Quantisation (MHQ) assigns $T = 2^B - 1$ thresholds per dimension, where B is the number of bits allocated per dimension. The encoding scheme for the thresholded regions is natural binary code (NBC). Each region from the left to the right is assigned an integer starting at 0 (on the left) and ending at $2^B - 1$ for the far right region. This integer index is converted to its equivalent NBC giving the codeword for that region.

fectiveness. To mitigate this issue Kong et al. (2012) propose taking the *Manhattan distance* between the integer index corresponding to a given NBC codeword, rather than the Hamming distance between the corresponding NBC codewords[7]. To illustrate how this method works I will consider the example given by Kong et al. (2012). Imagine we have generated the hashcode 000100 for data-point 1 and the hashcode 110000 for data-point 2. If $B = 2$, the Manhattan distance $\left\{d_{MHQ}(.,.) : \{0,1\}^K \times \{0,1\}^K \rightarrow \mathbb{Z}_+\right\}$ between the codewords is computed as in Equation 2.18

$$\begin{aligned} d_{MHQ}(000100, 110000) &= d_M(00,11) + d_M(01,00) + d_M(00,00) \\ &= 3 + 1 + 0 \\ &= 4 \end{aligned} \tag{2.18}$$

[7] I note in passing that *binary reflected Gray coding* would *not* be suitable as a binary codebook for nearest neighbour search. Gray coding has the special property that adjacent codewords differ by unit Hamming distance which has proved beneficial for enabling error correction in digital communication over analog channels (Gray (1953)). Gray coding is unsuitable for nearest neighbour search, however, due to the fact that codewords for data-points located much further apart can also have unit Hamming distance therefore breaking the neighbourhood structure.

If the number of bits per dimension $B = 3$ then the computation proceeds as in Equation 2.19.

$$\begin{aligned} d_{MHQ}(000100, 110000) &= d_M(000, 110) + d_M(100, 000) \\ &= 6 + 4 \\ &= 10 \end{aligned} \tag{2.19}$$

Computing the Manhattan distance between the integer indices of each region leads to remarkable increases in retrieval effectiveness as demonstrated in Kong et al. (2012). This is primarily due to the perfect preservation of the relative distance between the data-points: the codeword for each adjacent thresholded region is a unit Manhattan distance apart and there is a smooth increase in the Manhattan distance between any two regions the further apart they are along the projected dimension. Furthermore, this encoding scheme generalises easily to any desired number of thresholds because we are simply taking the integer index of each region. The obvious downside to computing the Manhattan distance between the integer indices of the thresholded regions is the slower distance computation versus computing the Hamming distance. On most modern processors the Hamming distance can be efficiently computed using a bitwise `XOR` between the hashcodes followed by a native `POPCOUNT` instruction which counts the number of bits set to one. It is not clear in Kong et al. (2012) whether or not the Manhattan distance will become a bottleneck on large datasets of millions of data points and dimensions. Some authors have recently offered evidence that this may indeed be the case (Wang et al. (2015)) by showing that the Manhattan distance requires substantially more atomic operations on the CPU than the Hamming distance. In Chapter 4, I mitigate this concern by introducing a more general quantisation model that is effective with a binary encoding scheme under the Hamming distance metric in addition to the more recently proposed MHQ NBC encoding scheme coupled with the Manhattan distance metric.

The MHQ threshold optimisation algorithm is straightforward: k-means (Lloyd (1982)) with 2^B centroids $\{c_i \in \mathbb{R}\}_{i=1}^{2^B}$ is used to cluster the projected dimension. The corresponding $2^B - 1$ thresholds $\{t_i \in \mathbb{R}\}_{i=1}^{2^B-1}$ are computed from the centroids by taking the midpoint between adjacent centroids (Equation 2.20).

$$t_i = \frac{(c_i + c_{i+1})}{2} \tag{2.20}$$

MHQ requires $O(2^B N_{trd})$ time for threshold learning along a single projected dimension and $O(1)$ time to generate a bit for a novel query point with the learnt thresholds.

2.5.5 A Link to the Discretisation of Continuous Attributes

It is interesting to consider briefly how this research area relates to the well-studied area of *discretisation of continuous attributes* in the field of machine learning (Dougherty et al. (1995), Garcia et al. (2013)). Several well-known machine learning models such as Naïve Bayes (Bishop (2006); Yang and Webb (2009)) often have continuous attributes transformed into nominal attributes by discretisation prior to learning. The mechanism by which this continuous to discrete transformation is performed shares many similarities to the quantisation process in the field of multi-threshold quantisation for hashing. More specifically the attributes (dimensions) are partitioned with a set of cut-points (thresholds) forming a non-overlapping division of the continuous domain. In a similar manner to the quantisation algorithms I discussed in this section the real-valued numbers within each thresholded region are assigned the corresponding discrete symbol representing that region. These discrete symbols must be binary for the quantisation algorithms studied in this section but need not be for the discretisation algorithms found in machine learning. The discretisation literature is broad and varied and proposes a wealth of algorithms for learning the cut-points, ranging from unsupervised (simply place the cut-points at equal intervals) through to supervised (Fayyad and Irani (1993)) and multivariate (each attribute is discretised jointly) (Mehta et al. (2005), Kerber (1992)). Given the maturity of the discretisation research field I believe that there is significant potential for these already established ideas to inform the design of future scalar quantisation algorithms for hashing.

2.5.6 A Brief Summary

In this section I introduced four recently proposed algorithms for scalar quantisation in the context of hashing-based ANN search. Each method takes a series of real-valued projections and outputs binary bits which are concatenated to form the hashcodes for the data-points. Each algorithm is similar in the sense that one or more thresholds are used to perform the binarisation: if a value is above or below a threshold it is assigned a codeword (single bit or multiple bits) of the associated region so formed. Single Bit Quantisation (SBQ) is the standard method of quantisation used by most previous hash-

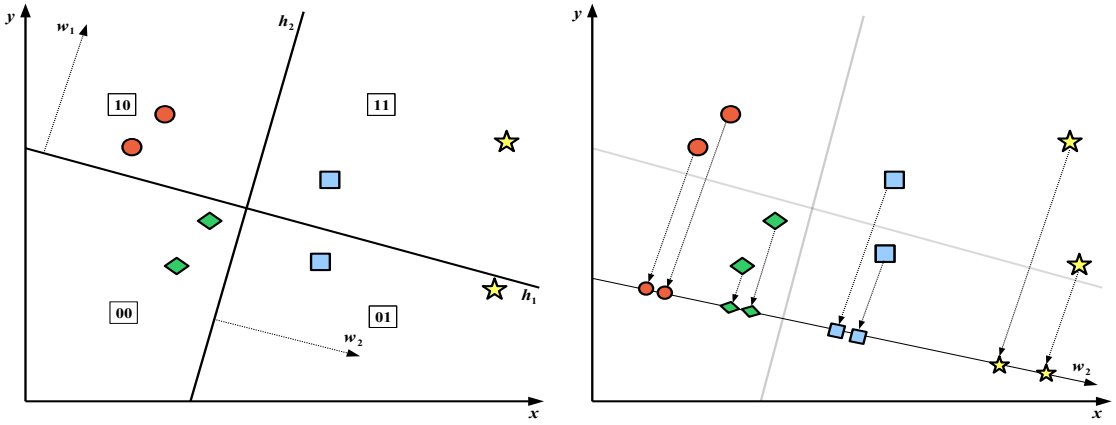
ing models (Section 2.5.1). SBQ positions a single threshold, typically at zero along a projected dimension. SBQ has the advantages of being simple and computationally efficient, but as I argued, it can lead to high quantisation errors (related data-points being assigned different bits). The multi-threshold quantisation algorithms, Hierarchical Quantisation (HQ) (Section 2.5.2), Manhattan Hashing Quantisation (MHQ) (Section 2.5.4) and Double Bit Quantisation (DBQ) (Section 2.5.3) all seek to address this issue with SBQ using novel encoding schemes and threshold optimisation algorithms. The manner in which the thresholds are optimised varied widely with each algorithm: HQ relies on a spectral graph partitioning objective, while MHQ and DBQ optimise objections related to squared error and variance minimisation. The encoding schemes also differ significantly between the three multi-threshold algorithms, but all are designed so that the relative distance between the data-points is maximally preserved in the resulting hashcodes. I now turn our attention to a family of methods in Section 2.6 that are able to generate the projections that we have just quantised.

2.6 Projection for Nearest Neighbour Search

In Section 2.4.1, I identified two main steps - projection and quantisation - that are used to generate similarity preserving hashcodes in the context of Locality Sensitive Hashing (LSH). I discussed how both steps taken together and performed in a sequence effectively check which sides of a set of hyperplanes a data-point falls, appending a '1' to the hashcode if a point falls on one side of a given hyperplane and a '0' otherwise. In Section 2.5 I reviewed prior art that focused solely on improving the quantisation step. The quantisation algorithms I examined attempt to better preserve the neighbourhood structure between the data-points during binarisation, improving upon simply taking the sign of the projections in Equation 2.21

$$h_k(\mathbf{x}_i) = \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k)) \tag{2.21}$$

where $\mathbf{w}_k \in \mathbb{R}^D$ is the hyperplane normal vector and $t_k \in \mathbb{R}$ is the quantisation threshold. Equation 2.21 is the popular linear hash function adopted in most hashing research. I discussed in Section 2.5 that, apart from Anchor Graph Hashing (Liu et al. (2011)), most quantisation models operate independently of the projection stage and assume that the projections to be binarised have already been generated by an existing projection method. In this section I review the equally important step of *projection* and focus



(a) **Feature space partitioning** (b) **Projection onto normal vector $\mathbf{w}_2$**

Figure 2.13: Illustration of the projection operation. Methods for projection partition the feature space with a set of hyperplanes. In Figure (a) I show a partitioning of a two dimensional feature space induced by two hyperplanes $\mathbf{h}_1$ and $\mathbf{h}_2$. To determine the bucket index or hashcode of a data-point it is necessary to project the data-points onto the normal vectors ($\mathbf{w}_1$, $\mathbf{w}_2$) followed by binary quantisation. In Figure (b) I show geometrically the result of projection onto normal vector $\mathbf{w}_2$. The resulting projections form projected dimension $\mathbf{y}^2 \in \mathbb{R}^{N_{trd}}$. Section 2.6 examines existing work that seek to position the hyperplanes so that many true nearest neighbours end up close to each other along the resulting projected dimensions.

on algorithms that seek to generate the projections in a way that preserves the relative distances between the data-points along the resulting projected dimensions. In the case of the linear hash function this is equivalent to positioning a set of K hyperplanes throughout the input feature space in such a way that similar data-points are likely to fall within the same polytope-shaped region. These regions constitute the hashtable buckets for indexing and retrieval. To generate a projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$ from a hyperplane $\mathbf{h}_k \in \mathbb{R}^D$ the data-points $\left\{\mathbf{x}_i \in \mathbb{R}^D\right\}_{i=1}^{N_{trd}}$ are projected onto the normal vector $\mathbf{w}_k \in \mathbb{R}^D$ using a dot product operation $\mathbf{w}_k^\intercal \mathbf{x}_i$. In Figure 2.13, I show geometrically the effect of the dot product and how a projected dimension is formed using this operation.

In Section 2.4, I introduced Locality Sensitive Hashing (LSH) a seminal early method for solving the ANN search decision problems given in Definitions 2.3.1-2.3.2. As I discussed in Section 2.4.1, LSH for the inner product similarity samples hyperplanes uniformly from the unit sphere, relying on an asymptotic guarantee that as the number of hyperplanes increases the Hamming distance between the hashcodes will

Method	Dependency	Learning Paradigm	Hash Function	Training Complexity	Properties	Section
LSH	Independent	Unsupervised	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k)$	$O(KD)$	E_2	2.4.1
SKLSH	Independent	Unsupervised	$sgn(cos(\mathbf{w}_k^\intercal \mathbf{x} + t_k) + t_{k'})$	$O(KD)$	E_2	2.6.2.1
PCAH	Dependent	Unsupervised	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k)$	$O(min(N_{trd}^2 D, N_{trd} D^2))$	E_2, E_3, E_4	2.6.3.1
AGH	Dependent	Unsupervised	$sgn(\mathbf{w}_k^\intercal \mathbf{z} + t_k)$	$O(N_{trd} CK)$	E_1, E_2, E_3, E_4	2.6.3.4
ITQ	Dependent	Unsupervised	$sgn(\mathbf{R}\mathbf{W}^\intercal \mathbf{x})$	$O(K^3)$	E_1, E_2, E_3, E_4	2.6.3.3
SH	Dependent	Unsupervised	$sgn(sin(\frac{\pi}{2} + j\pi(\mathbf{w}_k^\intercal \mathbf{x})))$	$O(min(N_{trd}^2 D, N_{trd} D^2))$	E_1, E_2, E_3, E_4	2.6.3.2
STH	Dependent	Supervised	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k)$	$O(N_{trd} DK + MN_{trd}^2 K)$	E_1, E_2, E_3, E_4	2.6.4.4
KSH	Dependent	Supervised	$sgn(\mathbf{w}_k^\intercal \kappa(\mathbf{x}) + t_k)$	$O(N_{trd} CK + N_{trd}^2 CK + N_{trd} C^2 K + C^3 K)$	E_1, E_2	2.6.4.3
BRE	Dependent	Supervised	$sgn(\mathbf{w}_k^\intercal \kappa(\mathbf{x}) + t_k)$	$O(KN_{trd}^2 + KN_{trd} \log N_{trd})$	E_1, E_2	2.6.4.2
CVH	Dependent	Supervised	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k^x)$, $sgn(\mathbf{u}_k^\intercal \mathbf{z} + t_k^z)$	$O(N_{trd} D^2 + D^3)$, D=$max(D_x, D_z)$	E_2, E_3, E_4	2.6.5.1
CMSSH	Dependent	Supervised	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k^x)$, $sgn(\mathbf{u}_k^\intercal \mathbf{z} + t_k^z)$	$O(KMN_{trd} D)$, D=$max(D_x, D_z)$	E_1, E_2	2.6.5.3
CRH	Dependent	Supervised	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k^x)$, $sgn(\mathbf{u}_k^\intercal \mathbf{z} + t_k^z)$	$O(D_x^2 D_z + D_x D_z^2 + D_z^3)$	E_1, E_2	2.6.5.2
PDH	Dependent	Supervised	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k^x)$, $sgn(\mathbf{u}_k^\intercal \mathbf{z} + t_k^z)$	$O(MN_{trd}^2 K)$	E_1, E_2, E_4	2.6.5.4
IMH	Dependent	Supervised	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k^x)$, $sgn(\mathbf{u}_k^\intercal \mathbf{z} + t_k^z)$	$O(N_{trd}^3)$	E_1, E_2, E_3, E_4	2.6.5.5

Table 2.2: Categorisation of existing projection learning algorithms. This table is inspired in part by the categorisation given in Wang et al. (2010a). See Section 2.6 for detail on the specifics of each hash function. The last five hash functions are cross-modal. N is the total number of data-points, K is the hashcode length, C are a set of anchor data-points ($C \ll N_{trd}$), N_{trd} are the number of training data-points ($C < N_{trd} \ll N$), M are the number of iterations. Typically $N_{trd} = 1000$-2000, $K = 32$-128 and $C = 300$.

reflect the cosine similarity between any two data-points[8]. Nevertheless, as I pointed out in Section 2.4, randomly sampled LSH hyperplanes tend to lack discrimination and run a high risk of partitioning regions of the input feature space dense in related data-points. In practice this means that many hyperplanes (bits) and many hash tables are required for adequate retrieval effectiveness. Unfortunately, longer hashcodes and more hashtables require a greater main memory allocation for the LSH deployment. Recently researchers have turned to the question of how best to generate more compact and discriminative hashcodes by learning hyperplanes adapted to the distribution of the data (Liu et al. (2011, 2012); Weiss et al. (2008); Gong and Lazebnik (2011); Raginsky and Lazebnik (2009); Kulis and Darrell (2009); Zhang et al. (2010b)). It is these methods that form the focus in this part of the literature review.

Existing work on projection methods for hashing-based ANN can usefully be divided into *three* sub-fields based on the degree to which the distribution of the data informs the construction of the hashing hyperplanes: *data-independent* (Section 2.6.2), *data-dependent but unsupervised* (Section 2.6.3) and *data-dependent and supervised* (Sections 2.6.4-2.6.5). The projection methods I examine in this section are categorised in Table 2.2. I segment the field into these three areas and review related work under each category in Sections 2.6.2-2.6.5. The review will take us on a journey across a wide array of truly diverse techniques for generating hash functions, from random projections, kernel functions, spectral methods to boosting. I attempt to be as thorough as possible in our coverage of existing related work. Nevertheless, the literature on projection is truly vast due to its popularity as a research topic and therefore it will be impossible to provide an exhaustive coverage here due to space constraints. Instead I focus in detail on the more well-known models across each category whose authors have made the codebase freely available to the research community. Both of these points ensure that any claims I make in this thesis are both meaningful (e.g. stemming from results collected on the same experimental framework and on the same dataset splits) and based upon results from competitive baselines. I point the interested reader to two recently published review articles of Wang et al. (2014) and Grauman and Fergus (2013) for an additional overview of this part of the field. Note further that all of the hashing models I review restrict themselves to search over a single hash table ($L = 1$) as is the tradition in the literature. Methods that explicitly *learn* multiple hashtables in a data-dependent manner are an interesting sub-field but are out of the

[8]Goemans and Williamson (1995) showed that the expected Hamming distance between two bit vectors formed by hash functions sampled from $\mathcal{H}_{cosine}$ will approximate the angle between the vectorial feature representation of the corresponding data-points in the input feature space.

scope of this review. The reader is encouraged to see Xu et al. (2011) and Liu et al. (2013) for research in this direction.

2.6.1 The Four Properties of an Effective Hashcode

Before I discuss individual models for projection, I will firstly examine several properties that contribute to making an effective hashcode for nearest neighbour search. The seminal work on Spectral Hashing (SH) by Weiss et al. (2008) first codified four properties of an effective hashcode (E_1-E_4):

- E_1: The hashcode should have *low Hamming distance* to the hashcodes of similar data-points.
- E_2: The hashcode should be *efficiently computable* for a novel query data-point.
- E_3: The bits of the hashcode should have *equal probability* of being 0 or 1.
- E_4: The different bits of the hashcode should be *pairwise independent*.

While I have previously discussed the importance of the first property (E_1) in the context of LSH (Section 2.4) and binary quantisation (Section 2.5), I have so far not discussed the remaining criteria (E_2-E_4). The second property (E_2) is crucial for applying a hashing scheme in practice. Given a novel data-point we should be able to rapidly compute its hashcode so that the overall query time is kept to a minimum. This is known in the learning to hash literature as *out-of-sample extension*. LSH has a straightforward and computationally efficient method for out-of-sample-extension: simply multiply the query data-point by the matrix where each column constitutes the normal vector of a randomly sampled hyperplane followed by sign thresholding (Section 2.4). The last two properties target the *efficiency* E_3 and *compactness* E_4 of the hashcode. Property E_3 requires each hyperplane to generate a balanced partition of the data by splitting the dataset into two partitions of equal size i.e. $\sum_{i=1}^{N_{trd}} h_k(\mathbf{x}_i) = 0$. By the principle of maximum entropy this will maximise the information captured by the associated bit (Baluja and Covell (2008)). This constraint has the desirable effect of mapping an equivalent number of data-points to each hashcode and therefore balancing the occupancy of the hashtable buckets[9]. At query time we therefore avoid the degenerate case of having to examine an unnecessarily large number of nearest neighbours

[9]Wang et al. (2012) showed how the NP-hard property E_3 could be relaxed (and therefore implemented) by showing that it is equivalent to maximising the variance for the k^{th} bit. Enforcing property E_3 might be sub-optimal, however, if it causes a cluster of related data-points to be partitioned into separate buckets. Usually such a situation can be remedied by using multiple independent hashtables.

in a given hashtable bucket. The fourth property E_4 targets hashcode compactness by eliminating any redundant bits that capture the same information on the input feature space. Ideally any hashing scheme should seek to minimise the number of bits in the hashcode to conserve storage and computation time. The vast majority of the data-dependent projection schemes introduced since the seminal work of Weiss et al. (2008) attempt to learn hashing hyperplanes that generate hashcodes with as many of these four properties as possible. I will study the extent to which these properties can be simultaneously preserved during the optimisation of the hashing hyperplanes in Sections 2.6.2-2.6.4.

2.6.2 Data-Independent Projection Methods

Aside from Locality Sensitive Hashing (LSH) which I reviewed in detail in Section 2.4, I will discuss one other data-independent hashing method in this thesis, Locality Sensitive Hashing from Shift Invariant Kernels (SKLSH) (Raginsky and Lazebnik (2009)) that extends LSH to the preservation of kernel similarity (Section 2.6.2.1).

2.6.2.1 Locality Senstive Hashing from Shift Invariant Kernels (SKLSH)

Locality Sensitive Hashing from Shift Invariant Kernels (SKLSH) extends LSH to the preservation of similarity between data-points as defined by an appropriate kernel function $\left\{\kappa : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}\right\}$ such as the Gaussian kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma \|\mathbf{x}i - \mathbf{x}_j\|^2/2)$ or the Laplacian Kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|_1/2)$ where $\gamma \in \mathbb{R}$ is the kernel bandwidth parameter. In essence the method is similar to LSH but with a different definition of the hash function family $\mathcal{H}$ due to the different similarity preservation required. The crux of this hashing model is to construct an embedding $\left\{g : \mathbb{R}^D \rightarrow \{0,1\}^K\right\}$ such that if two data-points are similar as defined by the kernel function i.e. $\kappa(\mathbf{x}_i, \mathbf{x}_j) \approx 1$ then there will be a high degree of overlap between their hashcodes i.e. $d_{hamming}(g(\mathbf{x}_i), g(\mathbf{x}_j)) \approx 0$, and vice-versa for the situation when $\kappa(\mathbf{x}_i, \mathbf{x}_j) \approx 0$. To construct a mapping with this property Raginsky and Lazebnik (2009) formulate a low-dimensional projection function given by $\Psi^K : \mathbb{R}^D \rightarrow \mathbb{R}^K$. This projection uses the random Fourier features of Rahimi and Recht (2007) that provide a guarantee that the inner product between the two transformed data-points approximates the output of a *shift invariant* kernel[10] $\Psi_k(\mathbf{x}_i) \cdot \Psi_k(\mathbf{x}_j) \approx \hat{\kappa}(\mathbf{x}_i - \mathbf{x}_j)$. The random Fourier features mapping is given in Equation 2.22

[10] A shift invariant kernel is defined as: $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \hat{\kappa}(\mathbf{x}_i - \mathbf{x}_j)$.

$$\Psi_k(\mathbf{x}_i) = \sqrt{2}cos(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k) \tag{2.22}$$

where for the Gaussian kernel $\mathbf{w}_k \sim \mathcal{N}(0, \gamma \mathbf{I}_{D\times D})$ and $t_k \sim Unif[0, 2\pi]$. The contribution of Raginsky and Lazebnik (2009) is to use this embedding as the centerpiece of a novel hash function (Equation 2.23)

$$h_k(\mathbf{x}_i) = \frac{1}{2}[1 + sgn(cos(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k) + t_{k'})] \tag{2.23}$$

where sgn denotes the sign function adjusted so that $sgn(0) = -1$ and $t_{k'} \sim Unif[-1, 1]$. Raginsky and Lazebnik (2009) provide a proof that hashing the data-points with K randomly sampled hash functions will yield a binary embedding whose Hamming distance approximates the desired shift invariant kernel similarity. As the hyperplanes are sampled randomly the training time complexity of this algorithm is a low $O(DK)$. SKLSH satisfies property E_2 of an effective hashcode, namely efficient computation of hashcodes.

2.6.3 Data-Dependent (Unsupervised) Projection Methods

In this section I will provide a critical appraisal of relevant related work that learns the hashing hyperplanes in a data-dependent manner but without the need for supervisory information in the form of user provided pairwise constraints on data-point similarity or class labels. All of the unsupervised data-dependent hashing models I review in this section learn the hashing hyperplanes by formulating a *trace minimisation/maximisation* problem which is solved in closed form as an eigenvalue problem or using singular value decomposition (SVD). These hashing methods rely directly on well established methods of linear and non-linear dimensionality reduction, specifically Principal Components Analysis (PCA) and Laplacian Eigenmaps (LapEig). Given the widespread use of matrix factorisation in the learning to hash literature, including many methods I do not review here, I give a brief introduction to this important solution strategy before I review the individual hashing algorithms themselves in Section 2.6.3.1-2.6.3.4[11].

There are effectively two main strategies for performing a dimensionality reduction on a dataset $\mathbf{X} \in \mathbb{R}^{N\times D}$ to obtain a new dataset $\mathbf{Y} \in \mathbb{R}^{N\times K}$ where $K \ll D$. The first method involves finding an explicit linear transformation of the data characterised by

[11] For more detail on trace optimisation and eigenproblems for dimensionality reduction the reader is pointed to the excellent article of Kokiopoulou et al. (2011).

a projection matrix $\mathbf{W} \in \mathbb{R}^{D \times K}$ into the lower dimensional space ($\mathbf{Y} = \mathbf{XW}$). PCA is a well-known member of this *projective* category. The second method computes a non-linear low-dimensional embedding $\mathbf{Y} \in \mathbb{R}^{N \times K}$ directly, without first finding an explicit mapping function. These latter methods, of which LapEig is a prime example, typically impose neighbourhood constraints such that close by data-points in the original space are close-by in the reduced space. Despite these differences, both categories can be neatly unified by a standard trace maximisation objective function (Equation 2.24)

$$\begin{aligned} &\text{argmax}_{\mathbf{V} \in \mathbb{R}^{N \times K}} \quad tr(\mathbf{V}^\intercal \mathbf{A} \mathbf{V}) \\ &\text{subject to } \mathbf{V}^\intercal \mathbf{1} = 0 \\ &\qquad\qquad \mathbf{V}^\intercal \mathbf{B} \mathbf{V} = \mathbf{I}^{K \times K} \end{aligned} \tag{2.24}$$

where $\mathbf{A}$ is a symmetric matrix, $\mathbf{B}$ is positive definite matrix, $\mathbf{V}$ is an orthonormal[12] matrix and $tr(A) = \sum_i A_{ii}$. The exact specification of these matrices is projection function dependent. I will concretely define $\mathbf{A}$, $\mathbf{B}$, $\mathbf{V}$ including their dimensionalities in Sections 2.6.3.1-2.6.3.4. But as a way of proving an immediate intuitive example, in the context of Principal Component Analysis (PCA) we have $\mathbf{A} = \mathbf{X}^\intercal \mathbf{X}$, $\mathbf{V} = \mathbf{W} \in \mathbb{R}^{D \times K}$ and $\mathbf{B} = \mathbf{I} \in \mathbb{R}^{D \times D}$. Therefore maximising the trace (Equation 2.24) in this case is equivalent to finding the principal directions in the data that capture the maximum variance in the input feature space.

The trace maximisation in Equation 2.24 can be solved as a general eigenvalue problem $\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{B}\mathbf{v}_i$, where $\mathbf{v}_i$ is the i^{th} eigenvector with eigenvalue λ_i (Saad (2011), Kokiopoulou et al. (2011)). This part of the learning to hash literature can now be distilled to its essence: in order to learn a set of data-dependent hash functions we shape our desired hashing optimisation problem into a form that resembles this template (Equation 2.24) and then we can simply solve for the K eigenvectors of a standard eigenvalue problem. This particular optimisation problem is easily solved using off the shelf solvers such as `eigs` or `svd` in Matlab. The main work in deriving an unsupervised data-dependent hash function can be summarised with the following standard four-step procedure:

1. Manipulating the problem into a matrix trace minimisation/maximisation (Equation 2.24).

[12] An orthonormal matrix $\mathbf{V}$ is a square matrix with real values whose columns and rows are orthogonal unit vectors. That is, $\mathbf{V}$ has the property $\mathbf{V}^\intercal \mathbf{V} = \mathbf{V}\mathbf{V}^\intercal = \mathbf{I}$, where $\mathbf{I}$ denotes the identity matrix.

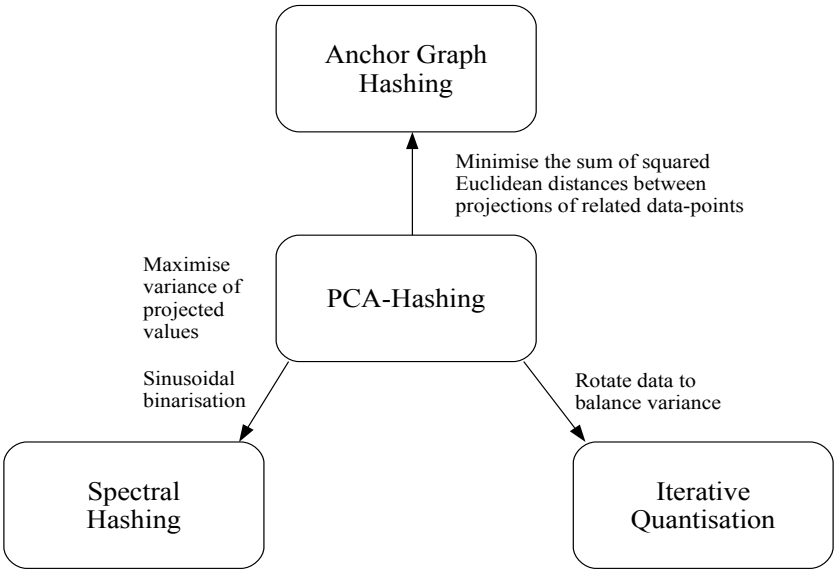


Figure 2.14: The data-dependent (unsupervised) hashing models are intimately related. This diagram illustrates one interpretation of that relationship where PCA is very much the centerpiece. The directed arcs are labelled with the operation necessary to transform one model into another model pointed to by the arc. See Section 2.6.3 for a full description of the four models.

2. Solving the optimisation objective as an eigenvalue problem or by performing a SVD. The K eigenvectors or right-singular vectors are the normal vectors of the hashing hyperplanes.

3. Dealing with the imbalanced variance resulting from the matrix factorisation.

4. Construct an out-of-sample extension in the case of a non-projective mapping.

We will see these four design principles in all four data-dependent hashing methods I review in this section. Specifically I will review PCA hashing (PCAH) (Section 2.6.3.1), Spectral Hashing (SH) (Section 2.6.3.2), Iterative Quantisation (ITQ) (Section 2.6.3.3) and Anchor Graph Hashing (AGH) (Section 2.6.3.4). In three out of four of the methods PCA extracts the directions of maximum variance which are then used as the hashing hyperplanes (PCAH, SH, ITQ). The contributions of the majority of these approaches lie in Step 3 where a sensible strategy is sought for minimising the impact of the imbalanced variance across hyperplanes, a phenomenon that reduces the quality of the hashcodes from lower principal components. The final method, AGH, takes a different tact (Section 2.6.3.4) and computes an eigenfunction extension of graph Laplacian eigenvectors, largely basing the hashcode learning on the Laplacian

Eigenmap dimensionality reduction algorithm. The objective of all of the presented algorithms is to learn K hash functions $\left\{h_k : \mathbb{R}^D \rightarrow \{0,1\}\right\}_{k=1}^{K}$ that can be concatenated to generate hashcodes for unseen data-points.

I present a diagram summarising one interpretation of the relationship between these hashing algorithms in Figure 2.14.

2.6.3.1 Principal Components Analysis Hashing (PCAH)

Principal components analysis (PCA) (Hotelling (1933)) has proven to be by far the most popular low dimensional embedding for data-dependent hashing schemes, with a large body of seminal works manipulating a PCA embedding to achieve superior retrieval accuracy over unsupervised hashing schemes (Kong and Li (2012b); Gong and Lazebnik (2011); Weiss et al. (2008); Wang et al. (2012)). I will therefore begin the review by examining the most basic instantiation of a PCA-based hashing scheme: namely computing the principal directions of the data and using the singular vectors with the highest singular values directly as the hashing hyperplanes without any further modification (Wang et al. (2010b)).

I assume without any loss of generality that the training data in $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D}$ has been centred by subtracting off the mean i.e. $\sum_{i=1}^{N_{trd}} \mathbf{x}_i = 0$. The standard maximum variance PCA objective can then be stated as in Equation 2.25

$$\begin{aligned} \text{argmax}_{\{\mathbf{w}_k \in \mathbb{R}^D\}_{k=1}^{K}} \quad & \frac{1}{N_{trd}} \sum_k \mathbf{w}_k^\intercal \mathbf{X}^\intercal \mathbf{X} \mathbf{w}_k \\ = \quad & \frac{1}{N_{trd}} tr(\mathbf{W}^\intercal \mathbf{X}^\intercal \mathbf{X} \mathbf{W}) \\ \text{subject to } & \mathbf{W}^\intercal \mathbf{W} = \mathbf{I} \end{aligned} \tag{2.25}$$

where $tr(\mathbf{A}) = \sum_i A_{ii}$ denotes the matrix trace operator and $\mathbf{W} \in \mathbb{R}^{D \times K}$ is the matrix with columns $\mathbf{w}_k$. The constraint $\mathbf{W}^\intercal \mathbf{W} = \mathbf{I}$ requires the learnt hyperplanes to be pairwise orthogonal which can be thought of as a relaxed version of the pairwise independence property for bits (property E_4 in Section 2.6.1). Equation 2.25 is identical to Equation 2.24 with $\mathbf{A} = \mathbf{X}^\intercal \mathbf{X}$, $\mathbf{V} = \mathbf{W}$ and $\mathbf{B} = \mathbf{I}$. Therefore the $\left\{\mathbf{w}_k \in \mathbb{R}^D\right\}_{k=1}^{K}$ maximising Equation 2.25 are exactly the right singular vectors with the largest singular values which can be obtained using SVD on $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D}$ in $O(min(N_{trd}^2 D, N_{trd} D^2))$ operations. The PCA solution $\mathbf{W} \in \mathbb{R}^{D \times K}$, where each column constitutes a principal component, can be interpreted as a rigid rotation of the feature space such that each succeeding coordinate captures as much of the variance of the input data as possible.

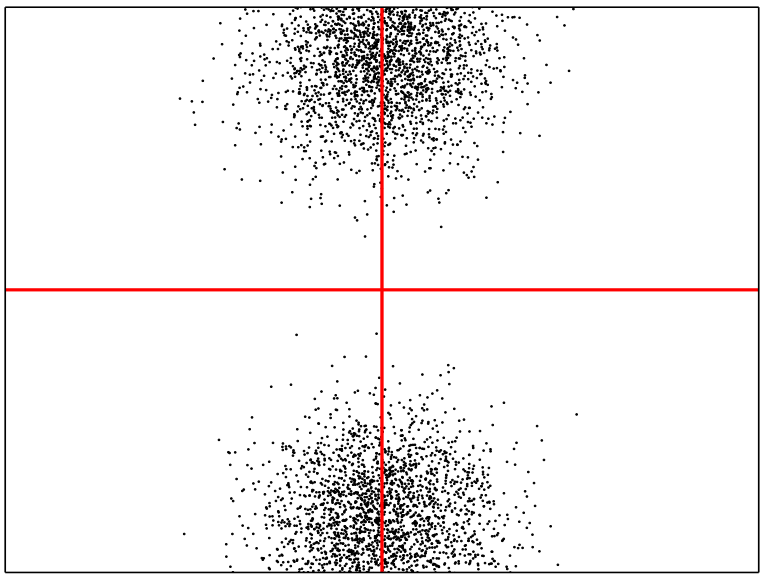


Figure 2.15: Plot displaying the PCA principal components $\mathbf{w}_1 \in \mathbb{R}^D$, $\mathbf{w}_2 \in \mathbb{R}^D$ (shown as perpendicular lines) for the data-points indicated by the black dots. Data has been aligned to the principal axes. The two principal components point in the directions of greatest variance of the data. These components are used as the vectors normal to the hashing hyperplanes in the PCA hashing (PCAH) algorithm.

For a K-bit hashcode it is common to take the K right-singular vectors with the highest singular values as the hashing hyperplanes while t_k is set to zero given that the data is mean-centered. The PCAH hash function is given in Equation 2.26.

$$h_k(\mathbf{x}_i) = \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{x}_i)) \tag{2.26}$$

Using PCA to generate hash functions can be thought of as attaining properties E_2, E_3, E_4 of an effective hashcode as identified in Section 2.6.1.

While the use of PCA is popular within the learning to hash literature, I mention here a number of disadvantages with using this matrix factorisation for generating hashcodes. Firstly, SVD is computationally expensive making this approach generally unattractive for databases with a large number of data-points and/or of a high dimensionality. Secondly, the number of bits K can never be greater than the dimensionality of the dataset D. In a practical hashing deployment, we first generate a very long hashcode for a data-point and then divide the hashcode up into L segments each of which provide the indices into the buckets of L hashtables. The fact that we must always have $K \leq D$ means that PCAH effectively places a restrictive upper bound on the number of hashtables L and hashcode lengths K we can use with this method. Finally, the singular

vectors with the lowest singular values are likely to be unreliable, capturing little variance in the feature space. Using these singular vectors as hyperplane normal vectors in Equation 2.26 is likely to result in poor quality hashcodes that do not discriminate well between data-points. This latter issue, which I term the *imbalanced variance problem*, resulted in a flurry of additional research that specifically examined how best to extract the most information from the singular vectors with the highest singular values (Sections 2.6.3.2, 2.6.3.4) or that transform the original data-space so that the learnt hyperplanes capture an equal amount of the variance (Section 2.6.3.3).

2.6.3.2 Spectral Hashing (SH)

Spectral Hashing (Weiss et al. (2008)) (SH) was one of the earliest proposed schemes for data-dependent hashing and can be seen as the spark that ignited interest in data-dependent hashing within the field of Computer Vision. SH provides a standard framework for graph-based hashing and is central to unsupervised and supervised hashing models proposed later in the learning to hash literature. I therefore spend some time in this section drilling into the fine details of the algorithm. As I discussed in Section 2.6.1, SH placed the requirements of an "effective hashcode" on a firm theoretical grounding by introducing four properties (E_1, E_2, E_3, E_4) that such hashcodes should exhibit. In contrast to simply binarising the projections onto the first K principal components as is done in PCAH (Section 2.6.3.1), a procedure which is unlikely to generate hashcodes with the desired properties, SH examines the extent to which we can integrate three of the properties E_1, E_3, E_4 directly into the optimisation problem as the objective function (E_1) and constraints (E_3, E_4). The optimisation problem introduced by SH is given in Equation 2.27

$$\begin{aligned} \text{argmin}_{\mathbf{Y}\in\mathbb{R}^{N_{trd}\times K}} \quad & \sum_{ij} S_{ij}\|\mathbf{y}_i - \mathbf{y}_j\|^2 \\ = \quad & tr(\mathbf{Y}^\intercal(\mathbf{D}-\mathbf{S})\mathbf{Y}) \\ \text{subject to } & \mathbf{Y} \in \{-1,1\}^{N_{trd}\times K} \\ & \mathbf{Y}^\intercal\mathbf{1} = \mathbf{0} \\ & \mathbf{Y}^\intercal\mathbf{Y} = N_{trd}\mathbf{I}^{K\times K} \end{aligned} \tag{2.27}$$

where $D_{ii} = \sum_j S_{ij}$ is the diagonal degree matrix of the adjacency matrix $\mathbf{S} \in \mathbb{R}^{N_{trd}\times N_{trd}}$. Weiss et al. (2008) assume that the Euclidean distance between the input data-points is to be preserved and therefore $S_{ij} = exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/\gamma^2)$ is an appropriate similarity,

where $\gamma \in \mathbb{R}$ is the kernel bandwidth parameter.

This objective function seeks to learn hashcodes $\left\{\mathbf{y}_i \in \{-1,1\}^K\right\}_{i=1}^{N_{trd}}$ where the average Hamming distance between similar neighbours is minimised, while satisfying bit balance and bit independence constraints. The constraint $\mathbf{Y}^\intercal \mathbf{1} = \mathbf{0}$ codifies property E_3 in requiring the bits to form a balanced partition of the feature space while constraint $\mathbf{Y}^\intercal \mathbf{Y} = N_{trd}\mathbf{I}^{K\times K}$ seeks bits that are pairwise uncorrelated which approximates property E_4. Unfortunately this optimisation problem is NP-hard even for a single bit, which can be proved with a reduction to the balanced graph partitioning problem which is well known to be NP-hard[13]. In order to make the optimisation problem tractable Weiss et al. (2008) use the spectral relaxation trick (Shi and Malik (2000)) removing the integrality constraint and letting the projection matrix $\mathbf{Y} \in \mathbb{R}^{N_{trd}\times K}$ consist of real numbers (Equation 2.28).

$$\begin{aligned} \text{argmin}_{\mathbf{Y}\in\mathbb{R}^{N_{trd}\times K}} \quad & tr(\mathbf{Y}^\intercal(\mathbf{D}-\mathbf{S})\mathbf{Y}) \\ \text{subject to } & \mathbf{Y} \in \mathbb{R}^{N_{trd}\times K} \\ & \mathbf{Y}^\intercal \mathbf{1} = \mathbf{0} \\ & \mathbf{Y}^\intercal \mathbf{Y} = N_{trd}\mathbf{I}^{K\times K} \end{aligned} \tag{2.28}$$

Equation 2.28 is identical to Equation 2.24 with $\mathbf{A} = \mathbf{D} - \mathbf{S}$ and $\mathbf{V} = \mathbf{Y}$. The solutions of Equation 2.28 are therefore the K eigenvectors with minimal eigenvalue of the graph Laplacian $\mathbf{D} - \mathbf{S}$[14]. The rows of the spectral embedding matrix $\mathbf{Y} \in \mathbb{R}^{N_{trd}\times K}$ can be interpreted as the coordinates of each data-point in the low-dimensional embedding. Solving Equation 2.28 ensures that data-points deemed close by the neighbourhood graph $\mathbf{S} \in \{0,1\}^{N_{trd}\times N_{trd}}$ are mapped nearby in the embedded space, preserving the local distances. The time complexity of solving Equation 2.28 is approximately $O(N_{trd}^2 K)$. Unfortunately, the graph Laplacian eigenvectors obtained in this way will only generate the hashcodes for the N_{trd} training data-points leaving open the question of out-of-sample extension. A common way of solving this problem in the context of spectral methods is to compute the Nyström extension (Bengio et al. (2004); Williams and Seeger (2001)). However without making a suitable approximation (see Section 2.6.3.4) this procedure is just as costly ($O(N_{trd}K)$) as performing a brute-force search through the database making it unattractive for encoding unseen data-points at query time. To circumvent this issue Weiss et al. (2008) make a simple approximation by

[13] The interested reader is pointed to Weiss et al. (2008) for a proof.

[14] The trivial eigenvector $\mathbf{1}$ with eigenvalue 0 is ignored.

assuming the projected data is sampled from a multi-dimensional uniform distribution. In doing so they show that an efficient out-of-sample extension can be obtained by simply computing the one-dimensional Laplacian eigenfunctions given by Equation 2.29.

$$\Psi_{kj}(y_i^k) = sin(\frac{\pi}{2} + \frac{f\pi}{b_k - a_k} y_i^k) \tag{2.29}$$

with eigenvalues given by Equation 2.30:

$$\lambda_{kf} = 1 - e^{-\frac{\gamma^2}{2}|\frac{f\pi}{b_k - a_k}|^2} \tag{2.30}$$

along the principal directions given by PCA, where $f \in \{1 \dots K\}$ is the frequency, a_k, b_k are parameters of a uniform distribution estimated for projected dimension k and $y_i^k \in \mathbb{R}$ denotes the projection of data-point $\mathbf{x}_i$ onto the k^{th} principal direction. For ease of exposition I split the SH algorithm into a training step in which the parameters of the uniform distribution approximation $\{a_k, b_k\}_{k=1}^K$ and the PCA principal directions $\left\{\mathbf{w}_k \in \mathbb{R}^D\right\}_{k=1}^K$ are estimated and an out-of-sample extension step in which the hash-codes of novel data-points are generated. Both steps are summarised in A and B below:

(A) Hash function training:

1. Extract K eigenvectors $\left\{\mathbf{w}_k \in \mathbb{R}^D\right\}_{k=1}^K$ by computing PCA on the training database $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D}$ and stack as the columns of matrix $\mathbf{W} \in \mathbb{R}^{D \times K}$.

2. Project $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D}$ onto the principal directions $\left\{\mathbf{w}_k \in \mathbb{R}^D\right\}_{k=1}^K$ by computing $\mathbf{Y} = \mathbf{X}\mathbf{W}$ where $\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}$

3. Estimate a uniform distribution $(a_k, b_k)_{k=1}^K$ for each projected dimension by computing the maximum b_k and minimum a_k extent of each dimension where $a_k = min(\mathbf{y}^k)$, $b_k = max(\mathbf{y}^k)$

4. For each projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$ compute K analytical eigenfunctions $\left\{\Psi_{kf}\right\}_{f=1}^K$ and their associated eigenvalues $\left\{\lambda_{kf} \in \mathbb{R}\right\}_{f=1}^K$ given by Equations 2.29-2.30.

5. Sort the K^2 eigenvalues and select the K analytical eigenfunctions from $\left\{\bar{\Psi}_{kj}\right\}_{k,j=1}^K$ with the smallest overall eigenvalues. Denote these as $\left\{\bar{\Psi}_k\right\}_{k=1}^K$, their corresponding normal vectors as $\left\{\bar{\mathbf{w}}_k \in \mathbb{R}^D\right\}_{k=1}^K$ and the parameters of the associated

uniform distributions $\{\bar{a}_k, \bar{b}_k\}_{k=1}^{K}$. Retain all three sets for out-of-sample extension.

(B) Out-of-sample extension:

1. Compute the K-bit hashcode $g(\mathbf{q}) = [h_1(\mathbf{q}), h_2(\mathbf{q}), \ldots, h_K(\mathbf{q})]$ for query $\mathbf{q}$ with the K hash functions defined as in Equation 2.31 using $\{\bar{\Psi}_k, \bar{\mathbf{w}}_k, \bar{a}_k, \bar{b}_k\}_{k=1}^{K}$ retained in Step 5 of the pre-processing stage. Using Equation 2.29 in the hash function can be thought of as a sinusoidal partitioning to be contrasted with the cosine partitioning of the projected dimension of SKLSH (Section 2.6.2.1).

$$h_k(\mathbf{q}) = \frac{1}{2}(1 + sgn(\bar{\Psi}_k(\bar{\mathbf{w}}_k^{\intercal}\mathbf{q})) \tag{2.31}$$

The computational complexity of this algorithm is dominated by the $O(min(N_{trd}^2 D, N_{trd} D^2))$ operations required to perform PCA on the database. SH prefers to select directions that have a large spread $|b_k - a_k|$ and low spatial frequency f. For low-dimensional data ($D \approx K$) SH commonly chooses multiple sinusoidal eigenfunctions with gradually higher frequencies for those eigenvectors that are pointing in the directions of greatest variance. To see this, note that the greater the variance of a projected dimension $\mathbf{y}^k$ the greater the range of $|b_k - a_k|$ and the lower the value of the corresponding eigenvalue given by Equation 2.30. In low-dimensional settings SH therefore has the desirable property of assigning more bits to the directions of highest variance in the input space, effectively up weighting the contribution of more informative hyperplanes in the Hamming distance computation. This somewhat overcomes the issue of PCAH in which we are progressively forced to pick orthogonal directions that capture less and less of the variance in the input space. Front loading the bits onto the most informative hyperplanes is one way of overcoming the imbalanced variance problem (Section 2.6.3.1) and usually leads to a higher retrieval effectiveness (Liu et al. (2011); Moran et al. (2013b)). The effectiveness of this variable bit allocation across hashing hyperplanes provides an inspiration for my novel variable threshold quantisation algorithm outlined in Chapter 5. In high dimensional settings ($D \gg K$) where the top eigenvectors capture a similar degree of variance, SH degenerates into PCAH by selecting each PCA hyperplane only once.

Despite the higher retrieval effectiveness versus LSH reported in Weiss et al. (2008) the unrealistic assumption of a uniform distribution has proved to be a considerable

limitation of this method. The Anchor Graph Hashing (AGH) algorithm of Liu et al. (2011) seeks to overcome this issue by making a clever approximation that permits an efficient application of the Nyström method for out-of-sample extension. I turn to AGH in Section 2.6.3.4.

2.6.3.3 Iterative Quantisation (ITQ)

While Spectral Hashing (SH) implicitly allocates more bits to the hyperplanes that capture a greater proportion of the variance in the input space in order to counteract the imbalanced variance problem, Iterative Quantisation (ITQ) seeks to balance the variance across PCA hyperplanes through a learnt rotation of the feature space. ITQ introduces an iterative scheme reminiscent of the k-means algorithm to find a rotation of the feature space $\mathbf{R} \in \mathbb{R}^{K \times K}$ so that the resulting projections onto the principal directions $\mathbf{W} \in \mathbb{R}^{D \times K}$ will minimise the quantisation error specified in matricial form in Equation 2.32

$$\begin{aligned} \text{argmin}_{\mathbf{B} \in \mathbb{R}^{N_{trd} \times K}, \mathbf{R} \in \mathbb{R}^{K \times K}} \quad & \|\mathbf{B} - \mathbf{YR}\|_F^2 \\ \text{where } & \mathbf{B} \in \{-1, 1\}^{N_{trd} \times K} \\ \text{subject to } & \mathbf{R}^\intercal \mathbf{R} = K\mathbf{I}^{K \times K} \end{aligned} \tag{2.32}$$

Equation 2.32 is similar to the orthogonal Procrustes[15] problem (Schönemann (1966)) in which we seek to transform one matrix into another using an orthogonal transformation matrix in such a way as to minimise the sum of the squares of the resulting residuals between the target matrix and the transformed matrix. In this case Equation 2.32 seeks a rotation matrix $\mathbf{R} \in \mathbb{R}^{K \times K}$ so that the squared Euclidean distance between the projection vectors $\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}$ and their associated binary vectors $\mathbf{B} = sgn(\mathbf{XW})$ is minimised, where PCA hyperplanes are stacked in the columns of $\mathbf{W}$. This optimisation is challenging as both matrices $\mathbf{B} \in \mathbb{R}^{N_{trd} \times K}$ and $\mathbf{R} \in \mathbb{R}^{K \times K}$ are initially unknown. To learn the optimal $\mathbf{R}$ we need to know optimal $\mathbf{B}$ and to learn the optimal $\mathbf{B}$ we need to know the optimal $\mathbf{R}$. This chicken and egg type problem can be solved with an iterative scheme akin to k-means that starts off with a random guess for $\mathbf{R}$, before refining the matrix through a two-step optimisation procedure in which both matrices

[15] For the interested reader this problem is named after a particular grisly Greek myth involving the protagonist Procrustes, a villain who offered unwitting travelers their much needed rest on a "magic" bed that could perfectly accommodate any visitor no matter their height. Unfortunately, Procrustes had a penchant for removing the arms and legs of his guests so that they could be perfectly accommodated on the bed.

Algorithm 4: ITERATIVE QUANTISATION (ITQ) (GONG AND LAZEBNIK (2011))

Input: Data-points $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D}$, PCA hyperplanes $\mathbf{W} \in \mathbb{R}^{D \times K}$, number of iterations M, randomly initialised rotation matrix $\mathbf{R} \in \mathbb{R}^{K \times K}$

Output: Optimised rotation matrix $\mathbf{R} \in \mathbb{R}^{K \times K}$

1 $\mathbf{Y} \leftarrow \mathbf{XW}$ `// Project data onto PCA hyperplanes`

2 **for** $m \leftarrow 1$ ***to*** M **do**

3 $\quad \mathbf{B} \leftarrow sgn(\mathbf{YR})$ `// Rotate data using` $\mathbf{R}$ `and quantise`

4 $\quad \mathbf{S\Omega\hat{S}}^\intercal \leftarrow SVD(\mathbf{B}^\intercal\mathbf{Y})$ `// Perform SVD on` $\mathbf{B}^\intercal\mathbf{Y}$

5 $\quad \mathbf{R} \leftarrow \mathbf{\hat{S}S}^\intercal$ `// Rotation minimising Eq 2.32 for fixed` $\mathbf{B}$

6 **end**

7 **return R**

are learnt individually with the other fixed (Gong and Lazebnik (2011)). The iterative ITQ algorithm is presented in Algorithm 4.

The key step in the ITQ algorithm is shown in Line 4 of Algorithm 4. In Hanson and Norris (1981) and Arun et al. (1987) it is shown that with a fixed target matrix $\mathbf{B}$ the sought after transformation $\mathbf{R}$ minimising the squared Euclidean distance can be obtained from the singular value decomposition (SVD) of matrix $\mathbf{B}^\intercal\mathbf{Y}$. With a fixed $\mathbf{R}$, Gong and Lazebnik (2011) show that the optimal $\mathbf{B}$ minimising Equation 2.32 can be obtained simply by using single bit quantisation (Section 2.5.1) (Line 3). In addition to properties E_1-E_2, ITQ approximately conserves properties E_3 and E_4 of an effective hashcode introduced in Section 2.6.1. The balanced partition property (E_3) is met by maximising the variance of the projections using PCA which was shown in Wang et al. (2010b) to be a good approximation to conserving E_3. E_4 is approximately met by computing PCA on the data as the resulting hyperplanes will be orthogonal, a relaxed version of the pairwise independence property. The most computationally expensive step of ITQ is in Line 4 where the SVD of a $K \times K$ matrix is computed. This step takes $O(K^3)$ operations, where K is the hashcode length. The learnt rotation matrix can then be used to construct an ITQ hashcode for an unseen query data-point $\mathbf{q} \in \mathbb{R}^D$ as given in Equation 2.33

$$g_l(\mathbf{q}) = \frac{1}{2}(1 + sgn(\mathbf{RW}^\intercal\mathbf{q})) \tag{2.33}$$

where I assume the data has been mean-centered so that the quantisation threshold

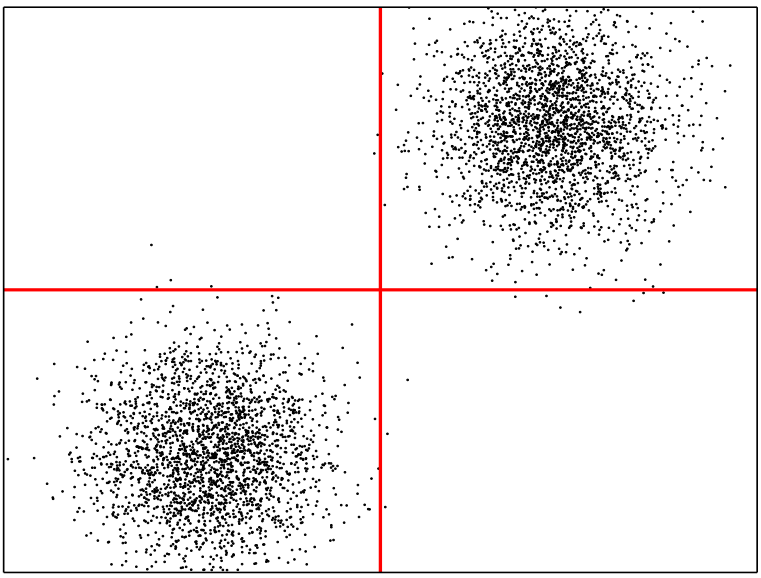


Figure 2.16: The effect of an ITQ rotation of the feature space. Here I show the same data as in Figure 2.15 but rotated by $\mathbf{R} \in \mathbb{R}^{K \times K}$ as found by ITQ over 100 iterations. The variance is more evenly distributed between the two hyperplanes (indicated as perpendicular lines) and the quantisation error is lower (no longer does a hyperplane directly cut through a cluster center). This is the optimisation objective of ITQ (Gong and Lazebnik (2011)).

$t_k = 0$.

I conclude with two personal observations on the ITQ algorithm. Firstly, such iterative two-step algorithms are a common and effective recipe for solving difficult optimisation problems within this field and crop up time and again in the literature. The need for a two-step algorithm is tied to the NP-hard problem of directly finding the optimal binary hashcodes. This issue can be tackled by making a continuous relaxation of $\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}$ as we first observed in the context of SH (Section 2.6.3.2). In this case a two-step procedure will find the best continuous approximation to the hashcodes followed by a second step that quantises the projections to generate the bits using either SBQ or one of the more sophisticated binarisation schemes introduced in Section 2.5. Being an approximation this process will produce sub-optimal hashcodes and so the challenge in most data-dependent projection models is to minimise the error in the continuous-to-binary conversion by learning the hashing hyperplanes in such a way that the resulting projections are more amenable to accurate binarisation. This algorithmic pattern is clearly evident in ITQ. Indeed, I will introduce my own novel data-dependent projection algorithm in Chapter 6 which has as its centerpiece a multi-

step iterative algorithm that allows us to solve what would otherwise be a much more challenging optimisation problem. Only recently have authors turned to the more difficult problem of formulating data-dependent hashing algorithms that optimise for the binary hashcodes directly without making a continuous relaxation, see Section 2.6.4.2 and Liu et al. (2014) for an overview.

Secondly I comment briefly on why ITQ is considered a method of projection in this thesis, rather than quantisation. In Section 2.5, I defined a quantisation algorithm as one which learns one or more thresholds along a projected dimension that are then subsequently used in a thresholding operation to convert the real-valued projections to binary. ITQ is therefore not strictly a quantisation algorithm under the definition considered in this thesis as it does not directly convert real-valued projections to binary relying instead on SBQ (Section 2.5.1) for quantisation. I therefore categorise ITQ as a method for data-dependent projection as it works directly with the PCA hyperplanes rotating the data so that the resulting projections better preserve the locality structure of the input data-space.

2.6.3.4 Anchor Graph Hashing (AGH)

I previously described the Hierarchical Quantisation (HQ) algorithm employed by Anchor Graph Hashing (AGH) in Section 2.5.2. In this section I will focus exclusively on the AGH component that learns the projection function. AGH examines the same relaxed objective function as SH which I repeat in Equation 2.34 for reading convenience

$$\begin{aligned} \text{argmin}_{\mathbf{Y}\in\mathbb{R}^{N_{trd}\times K}} \quad & tr(\mathbf{Y}^\intercal(\mathbf{D}-\mathbf{S})\mathbf{Y}) \\ \text{subject to } & \mathbf{Y}\in\mathbb{R}^{N_{trd}\times K} \\ & \mathbf{Y}^\intercal\mathbf{1}=\mathbf{0} \\ & \mathbf{Y}^\intercal\mathbf{Y}=N_{trd}\mathbf{I}^{K\times K} \end{aligned} \tag{2.34}$$

The computational bottlenecks involved with this objective function are two-fold: firstly the similarity matrix $\mathbf{S}\in\mathbb{R}^{N_{trd}\times N_{trd}}$ requires $O(N_{trd}^2 D)$ computations to construct. Secondly as for any hashing method we need to compute the hashcodes for unseen query data-points using K hash functions $\left\{h_k : \mathbb{R}^D \rightarrow \{0,1\}\right\}_{k=1}^{K}$. Unfortunately solving Equation 2.34 and binarising the resulting eigenvectors will only provide the hashcodes for the training data-points used to construct the adjacency matrix

$\mathbf{S} \in \mathbb{R}^{N_{trd} \times N_{trd}}$. We need to extend the K graph Laplacian eigenvectors to K eigenfunctions $\left\{\Psi_k : \mathbb{R}^D \rightarrow \mathbb{R}\right\}_{k=1}^{K}$ which we can combine with an appropriate quantisation method to form the hash functions that will encode any data-point (seen or unseen). As mentioned in Section 2.6.3.2 this out-of-sample extension can be derived using the Nyström method (Bengio et al. (2004); Williams and Seeger (2001)) requiring $O(N_{trd}K)$ time for one data-point. Clearly this time complexity is not amenable to online hashcode generation for out-of-sample query data-points. The key takeaway message of the AGH algorithm is that a sparse, low-rank approximation of $\mathbf{S}$ can be implicitly manipulated through operations on a *truncated* similarity matrix $\mathbf{Z} \in \mathbb{R}^{N_{trd} \times C}$ ($C \ll N_{trd}$) known as the anchor graph. The approximate similarity matrix $\hat{\mathbf{S}} \in \mathbb{R}^{N_{trd} \times N_{trd}}$, which never needs to be explicitly computed, permits eigenfunction extension of the graph Laplacian in a time independent of the number of data-points while avoiding the need to manipulate the full dense similarity matrix $\mathbf{S}$. Furthermore, by computing the Nyström extension AGH is able to avoid the unrealistic separable uniform distribution assumption made by Spectral Hashing (described in Section 2.6.3.2).

More specifically the centerpiece of the AGH method is the concept of the *anchor graph* $\mathbf{Z} \in \mathbb{R}^{N_{trd} \times C}$, an approximation of a full data affinity graph, that only consists of the similarities from N_{trd} data-points to a small set of C anchors rather than the complete pairwise similarities between N_{trd}^2 data-points. These anchors are simply computed by running k-means over the training dataset and selecting the centroids $\left\{\mathbf{c}_i \in \mathbb{R}^D\right\}_{i=1}^{C}$ as the C anchor data-points. I first presented the anchor graph formulation in Equation 2.10 in the context of the Hierarchical Quantisation (HQ) method which for convenience I repeat in Equation 2.35

$$Z_{ij} = \begin{cases} \dfrac{exp(-d^2(\mathbf{x}_j, \mathbf{c}_i)/\gamma)}{\sum\limits_{i' \in \langle j \rangle} exp(-d^2(\mathbf{x}_j, \mathbf{c}_{i'}))/\gamma)} & \text{if } i \in \langle j \rangle \\ 0 & otherwise \end{cases} \tag{2.35}$$

where γ is the kernel bandwidth, $\left\{d(.,.) : \mathbb{R}^D \times \mathbb{R}^D \rightarrow [0,1]\right\}$ is a distance function and $\langle j \rangle \in \{1 \ldots R\}$ are the indices of the $R \ll C$ nearest anchors to $\mathbf{x}_j$ under the distance metric $d(.,.)$. As the number of anchors is much less than the number of data-points ($C \ll N_{trd}$), constructing the anchor graph is $O(N_{trd}CD)$ rather than $O(N_{trd}^2 D)$ for $\mathbf{S}$. Liu et al. (2011) show that the full similarity matrix $\hat{\mathbf{S}}$ can be approximated as

$\hat{\mathbf{S}} = \mathbf{Z}\Sigma^{-1}\mathbf{Z}^\intercal$ where $\Sigma = diag(\mathbf{Z}^\intercal\mathbf{1})$. The approximate similarity matrix $\hat{\mathbf{S}}$ has the computationally attractive properties of being sparse and low rank. The low rank property is exploited in the graph Laplacian eigenvector extraction by solving the eigenvalue system of the small $C \times C$ matrix $\Sigma^{1/2}\mathbf{Z}^\intercal\mathbf{Z}\Sigma^{-1/2}$. Given a bit budget of K in the hierarchical variant of their algorithm, Liu et al. (2011) select $K^{'} = K/2$ of the C eigenvectors with the highest eigenvalues as the hashing hyperplane normal vectors. Stacking the $K^{'}$ eigenvectors columnwise in matrix $\mathbf{V}$ in descending order of eigenvalue and the corresponding eigenvalues on the diagonal of matrix Λ, the required graph Laplacian eigenvectors $\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K^{'}}$ can be computed as given in Equation 2.36.

$$\mathbf{Y} = \sqrt{N_{trd}}\mathbf{Z}\Sigma^{-1/2}\mathbf{V}\Lambda^{-1/2} = \mathbf{ZW} \tag{2.36}$$

The training time complexity of computing $\mathbf{Y}$ is $O(N_{trd}CK^{'})$. The columns of the matrix $\mathbf{W} \in \mathbb{R}^{C \times K^{'}}$ can be seen as the normal vectors of $K^{'}$ hyperplanes partitioning the space $\mathbb{R}^C$ formed by the non-linear mapping in Equation 2.35. Liu et al. (2011) show that an out-of-sample extension can be achieved in two steps: firstly, the unseen query data-point $\mathbf{q}$ is non-linearly projected into the space $\mathbb{R}^C$ by computing the similarity of $\mathbf{q}$ to the C cluster centroids $\left\{\mathbf{c}_i \in \mathbb{R}^D\right\}_{i=1}^{C}$ using Equation 2.35. This operation results in a sparse transformed vector $\mathbf{z} \in \mathbb{R}^C$ which can also be interpreted as a kernelised feature map (Murphy (2012)). This step is subsequently followed by a linear projection of $\mathbf{z}$ onto the k-th hyperplane $\mathbf{w}_k \in \mathbb{R}^C$ partitioning the space $\mathbb{R}^C$. The AGH hash function is formed from both steps (Equation 2.37)

$$h_k(\mathbf{q}) = \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal\mathbf{z})) \tag{2.37}$$

where I again assume the data is mean centered so that $t_k = 0$.

This hash function can be thought of as non-linearly mapping the data into a space where it is more likely to be linearly separable by linear decision boundaries. Given an unseen query data-point computing this out-of-sample extension takes $O(CD + CK^{'})$ operations, a testing time complexity that is a marked improvement over the $O(N_{trd}K)$ time complexity of the Nyström method[16]. Rather than generate one bit per hash function as suggested by Equation 2.37, the most accurate variant of AGH generates two bits for each of the resulting projected dimensions $\mathbf{y}^k = Y_{\bullet k}$ with $k \in [1, \ldots, K^{'}]$. We

[16] Assuming $D \ll N_{trd}$, which is generally true for the most common image features such as Gist and SIFT.

previously discussed this Hierarchical Quantisation (HQ) algorithm in detail in Section 2.5.2.

2.6.3.5 A Brief Summary

In our first foray in data-dependent hashing algorithms I surveyed a selection of the more well-known unsupervised algorithms that position the hashing hyperplanes based on the distribution of the data. I reviewed Principal Components Analysis Hashing (PCAH) (Section 2.6.3.1), Spectral Hashing (SH) (Section 2.6.3.2) and Anchor Graph Hashing (AGH) (Section 2.6.3.4). I saw that all three models reviewed are closely related in their application of a well-known dimensionality reduction method, either Principal Components Analysis (PCA) or Laplacian Eigenmaps (LapEig), to learn the hashing hyperplanes.

Three out of four of the hashing models (PCAH, SH, ITQ) used PCA, setting the hashing hyperplanes to be the right singular vectors resulting from a SVD on the data matrix. Two of these models (SH, ITQ) highlighted the issue of *variance imbalance* in which the hyperplanes capturing a smaller amount of the variance are much less reliable for hashing. The upshot of this is that PCAH retrieval effectiveness declines markedly with longer hashcode lengths due to the incorporation of lower quality hyperplanes into the hashcode generation. To counter this degradation in performance SH assigns more hashcode bits to the hyperplanes with higher variance while ITQ rigidly rotates the feature space to explicitly balance the variance across hyperplanes. All three models show higher retrieval effectiveness than PCAH which assigns 1 bit per hyperplane or simply uses the PCA hyperplanes as is.

I also discussed how the AGH algorithm took a different strategy to the PCA-based hashing algorithms by using a LapEig-inspired dimensionality reduction. In this scenario a nearest neighbour graph was built from the input data which was then used in an eigenvalue problem to extract graph Laplacian eigenvectors. Given that LapEig is a non-projective dimensionality reduction these eigenvectors were shown to yield the hashcodes for only those data-points used in the neighbourhood graph computation. An appealing property of AGH is its computationally efficient method, based on the Nyström method of Williams and Seeger (2001), for out-of-sample extension to unseen data-points.

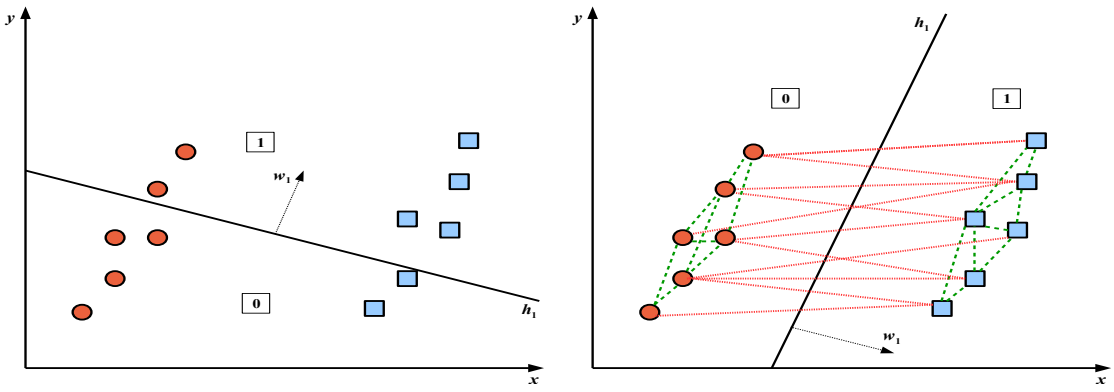
Aside from AGH which makes an honest attempt at reducing the computational complexity at training time, the downside with most of these hashing algorithms is the severe computational penalty $O(min(N^2D, ND^2))$ required for solving the SVD or

eigenvalue problem making their application intractable for large-scale datasets of high dimensionality. Indeed, as we will see in forthcoming sections most data-dependent hashing models (both supervised and unsupervised) generally rely on a matrix factorisation.

2.6.4 Data-Dependent (Supervised) Projection Methods

In Section 2.4 and Sections 2.6.2-2.6.3 I reviewed a selection of state-of-the-art data-independent and data-dependent hashing models. The data-independent models preserve a similarity, such as the cosine or a kernel similarity, that is non data-adaptive and is therefore unlikely to do very well at capturing a user-defined notion of similarity across many different tasks. Moreover, the data-dependent (unsupervised) models assume, for example, that discriminative hashcodes can be generated from projected dimensions that capture the maximum variance in the input space. This relies on variance being a quantity that can effectively distinguish between unrelated data-points, an assumption which may not be valid in many datasets of practical interest, such as image datasets collated "in the wild" from the WWW that depict images of varying topic, quality and resolution. This is exacerbated by the well-known *semantic gap* problem in computer vision which highlights the gulf between the statistics of the images captured by low-level image features such as Gist and SIFT and the high-level semantic concepts that are depicted in the image (Smeulders et al. (2000)). A robust way of linking these two domains is one of the grand challenges in the sub-fields of object recognition and image annotation (Moran and Lavrenko (2015a)), and is also important in our selected task of image retrieval.

To mitigate the difficulties arising from the semantic gap and capture the complex relationships between data-points found in real-world datasets, such as whether two images depict a cat or a person, it is generally much better to learn a hash function from a small amount of available supervision in the form of human annotated class labels or pairwise cannot-link or must-link constraints that specify which data-point pairs should or should not have the same hashcodes. In the visual search domain, Grauman and Fergus (2013) highlight potential sources of supervisory information ranging from explicit labelling of a subset of the database, to known correspondences between points in image pairs and user feedback on image search results. It is this category of hashing model that I review in this section. In general, I define a supervised hashing model as a model that leverages the same type of information (e.g. class labels, metric distances)



(a) **Unsupervised** (b) **Supervised**

Figure 2.17: Supervised versus unsupervised projection function learning. Illustration of a situation where learning hashing hyperplanes based on pairwise user provided constraints can yield a more effective bucketing of the space than a partitioning based on maximum variance. Points with similar shapes and colours are 1-nearest neighbours. In Figure (a) hyperplane $\mathbf{h}_1 \in \mathbb{R}^D$ is learnt via PCA with its normal vector $\mathbf{w}_1 \in \mathbb{R}^D$ pointing in the direction of maximum variance in the data. Projecting data-points onto the normal vector $\mathbf{w}_1 \in \mathbb{R}^D$ places related data-points (indicated by the same shapes) into different buckets. In contrast Figure (b) illustrates the effect of constraining the hyperplane positioning by using a set of must-link (show as dotted lines) and cannot-link (shown as solid lines connecting the data-points) constraints. I only show a subset of the constraints for clarity. In this case all related data-points fall within the same bucket as each other yielding a more effective partitioning of the space.

in the hash function learning algorithm that was also used to compute the groundtruth information for evaluation purposes. In Figure 2.17, I illustrate a situation in which learning hyperplanes based on pairwise labels yields a more effective bucketing of the space than one based purely on captured variance.

In a similar manner to the data-dependent (unsupervised) models, I restrict my attention to a selection of the most well-known baselines from the literature and whose authors have made the codebase freely available to the research community thereby making a fair comparison to our own methods possible under identical experimental conditions. I review ITQ with a Canonical Correlation Analysis (CCA) embedding (ITQ + CCA) (Gong and Lazebnik (2011)), Supervised Hashing with Kernels (KSH) (Liu et al. (2012)), Binary Reconstructive Embedding (BRE) (Kulis and Darrell (2009)) and Self-Taught Hashing (STH) (Zhang et al. (2010b)). These four models

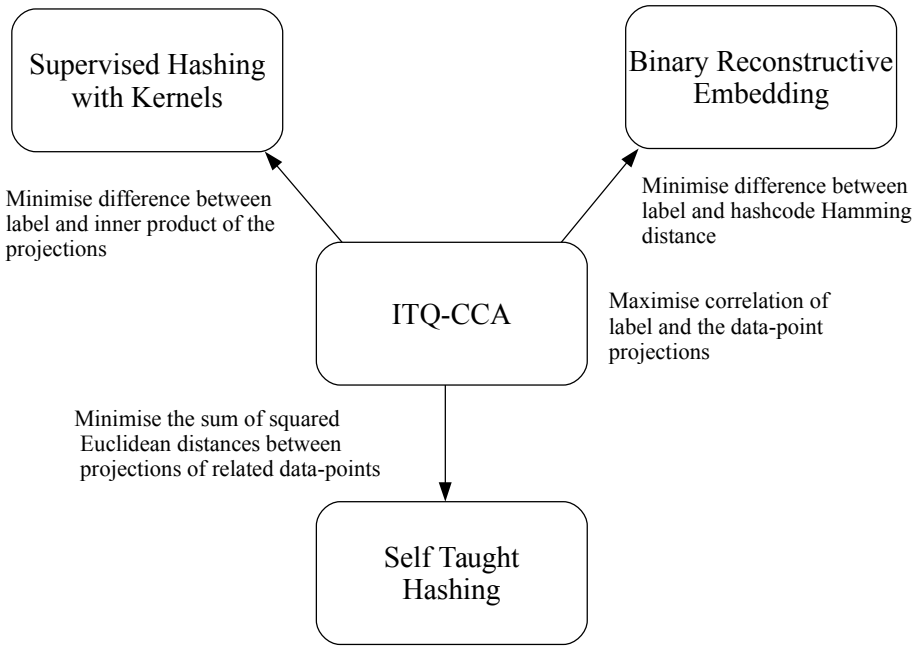


Figure 2.18: Relationship between the four supervised hashing models reviewed in this section. The labels on the arrows indicate the transformation necessary to convert between the different models. The fundamental difference between the models arises in how the available labels are related to the projections/hashcodes so as to compute an error signal to adjust the hashing hyperplanes.

fundamentally differ only in how they use the available labels to derive an error signal that can then be used to adjust the positioning of the hashing hyperplanes. For example, BRE and KSH frame similar objective functions that attempt to minimise the difference between the labels and the hashcode distances (BRE and KSH). STH uses the LapEig objective which minimises the difference between the projections of data-points with the same label while ITQ+CCA frames an objective that maximises the correlation of the labels and data-point projections. The relationship between these four supervised hashing models is summarised in Figure 2.18.

2.6.4.1 ITQ + Canonical Correlation Analysis (CCA)

I reviewed the unsupervised variant of Iterative Quantisation (ITQ) in Section 2.6.3.3. ITQ learns an orthogonal rotation matrix $\mathbf{R} \in \mathbb{R}^{K \times K}$ that transforms PCA projected data in a way that minimises the error of mapping the data to the vertices of a binary hypercube. ITQ is independent of the method for generating the orthogonal hashing hyperplanes $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K]$ where $\mathbf{w}_k \in \mathbb{R}^D$, which in the case of the origi-

nal algorithm was PCA. It is therefore straightforward to make ITQ into a supervised algorithm by using a supervised embedding to learn the hashing hyperplanes rather than PCA. Gong and Lazebnik (2011) replace PCA with Canonical Correlation Analysis (CCA) (Hardoon et al. (2003)) a well-known multi-view dimensionality reduction technique that explores the interaction between data vectors in two different feature spaces $\mathcal{X}$ and $\mathcal{Z}$. Assume we have N_{trd} training data-points in matrix $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D_x}$ and their associated labels in matrix $\mathbf{Z} \in \mathbb{R}^{N_{trd} \times D_z}$, where usually $D_x \neq D_z$. Each row of matrix $\mathbf{Z}$ is a binary indicator vector $\mathbf{z}_i \in \{0,1\}^{D_z}$ where a '1' indicates that the data-point $\mathbf{x}_i$ is tagged with that label and a '0' otherwise. The CCA algorithm finds two hyperplane normal vectors $\mathbf{w}_k \in \mathbb{R}^{D_x}$ and $\mathbf{u}_k \in \mathbb{R}^{D_z}$ so that the projections $\mathbf{X}\mathbf{w}_k$ and $\mathbf{Z}\mathbf{u}_k$ are maximally correlated (Equation 2.38).

$$\begin{aligned} \text{argmax}_{\mathbf{w}_k \in \mathbb{R}^D, \mathbf{u}_k \in \mathbb{R}^D} \quad & \frac{\mathbf{w}_k \mathbf{X}\mathbf{Z}\mathbf{u}_k}{\sqrt{\mathbf{w}_k^\intercal \mathbf{X}^\intercal \mathbf{X}\mathbf{w}_k \mathbf{u}_k^\intercal \mathbf{Z}^\intercal \mathbf{Z}\mathbf{u}_k}} \\ \text{subject to } & \mathbf{w}_k^\intercal \mathbf{X}^\intercal \mathbf{X}\mathbf{w}_k = 1 \\ & \mathbf{u}_k^\intercal \mathbf{Z}^\intercal \mathbf{Z}\mathbf{u}_k = 1 \end{aligned} \tag{2.38}$$

This objective function can be maximised by solving the following generalised eigenvalue problem (Gong and Lazebnik (2011))

$$\mathbf{X}^\intercal \mathbf{Z}(\mathbf{Z}^\intercal \mathbf{Z} + \rho \mathbf{I})^{-1} \mathbf{Z}^\intercal \mathbf{X}^\intercal \mathbf{w}_k = \lambda_k^2 (\mathbf{X}^\intercal \mathbf{X} + \rho \mathbf{I}) \mathbf{w}_k \tag{2.39}$$

where λ_k is the eigenvalue and ρ is a regularisation constant set to 0.0001 in Gong and Lazebnik (2011). Repeatedly solving Equation 2.39 for directions that are orthogonal to all previously discovered hyperplanes gives K orthogonal hyperplanes with normals $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K]$ whose positioning in the feature space have been influenced by the supervisory signal. Having learnt the hyperplanes $\mathbf{W} \in \mathbb{R}^{D \times K}$ in modality $\mathcal{X}$ the remainder of the ITQ+CCA algorithm proceeds in the same way as for the unsupervised variant of ITQ (Section 2.6.3.3). If I denote $D = max(D_x, D_z)$, then the computational complexity of ITQ+CCA is bounded by $O(N_{trd}D^2 + D^3)$. This is made up of the $O(N_{trd}D^2)$ operations required to compute the covariance matrices and the $O(D^3)$ operations arising from the matrix multiplications, inversion and solving the eigenvalue problem (Rasiwasia et al. (2014)). Following a similar line of argument to ITQ, ITQ + CCA approximately preserves properties E_1-E_4 of an effective hashcode.

2.6.4.2 Binary Reconstructive Embedding (BRE)

Binary Reconstructive Embedding (BRE) is the only projection method I consider that does not make use of the spectral relaxation trick to circumvent the NP-hard optimisation problem of learning binary hashcodes directly. We were first introduced to this continuous relaxation in the context of Spectral Hashing (Section 2.6.3.2). Without making the spectral relaxation and dropping the sign function from the optimisation objective many approaches to data-dependent hashing are discontinuous and non-differentiable. The contribution of BRE is a novel optimisation objective and a coordinate descent algorithm that solves the discrete optimisation problem directly without appealing to a continuous relaxation. As we have seen before in this literature review many methods solve for a matrix $\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}$ of real-numbers and then binarise this matrix to reveal the hashcodes using, for example, single bit quantisation (SBQ). These two steps are disconnected and there is therefore no guarantee that the real-values in $\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}$ will reliably map to accurate binary hashcodes particularly if they are close to the threshold boundary (which is typically at zero for mean centered data). BRE brings both steps into the optimisation objective by retaining the sign function. I present the *supervised* variant of the BRE objective function in Equation 2.40

$$\begin{aligned} &\text{argmin}_{\mathbf{W} \in \mathbb{R}^{D \times K}} \sum_{ij \in \mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}} \left\{ (1 - S_{ij}) - \frac{1}{K} \| g(\mathbf{x}_i) - g(\mathbf{x}_j) \|_2^2 \right\}^2 \\ &\text{subject to } g(\mathbf{x}_i) = [h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \ldots, h_k(\mathbf{x}_i)]^\intercal \\ &\text{where } h_k(\mathbf{x}_i) = \frac{1}{2} sgn(1 + \sum_{j=1}^{C} W_{jk} \kappa(\mathbf{x}_j, \mathbf{x}_i)) \end{aligned} \tag{2.40}$$

where $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$ is an adjacency matrix with $S_{ij} = 1$ indicates $\mathbf{x}_i$ and $\mathbf{x}_j$ are related and 0 otherwise. N_{trd} data-points ($C < N_{trd} \ll N$) are sampled from the dataset to construct $\mathbf{S}$ and C data-points are sampled uniformly at random as the anchor points for efficient kernel computation. $\mathbf{W} \in \mathbb{R}^{C \times K}$ is initialised randomly, κ is a kernel function $\left\{ \kappa : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R} \right\}$. Kulis and Darrell (2009) set κ to be the linear kernel in the original publication.

The objective function in Equation 2.40 attempts to make the normalised Hamming distance low for those data-point pairs with $S_{ij} = 1$, and large otherwise. No part of this objective encourages the conservation of properties E_3 and E_4 of an effective hashcode.

In the case of both objective functions a kernelised feature map $\left\{\kappa : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}\right\}$ is computed against a small number C of randomly sampled data-points from the training dataset, and the mapped data projected onto a set of K hyperplane normal vectors $\left\{\mathbf{w}_k \in \mathbb{R}^C\right\}_{k=1}^{K}$. This formulation of the hash function is similar to that of Anchor Graph Hashing (Equation 2.37) except AGH maps the data non-linearly using an RBF kernel and uses k-means centroids as the C samples to construct the kernel. The retrieval effectiveness of BRE may benefit from a non-linear kernelised feature map although this formulation was not explored in the original publication.

Perhaps the most interesting contribution of BRE is the optimisation algorithm used to minimise Equation 2.40 with the sign function intact. To optimise the non-differentiable objective function Kulis and Darrell (2009) formulate a coordinate descent algorithm that cycles through each hash function one by one and finds the value minimising Equation 2.40 of a randomly chosen element W_{jk} of each hyperplane $\mathbf{W}_{\bullet k}$, while holding the remaining hyperplanes constant. Kulis and Darrell (2009) provide a closed form solution for computing the optimal W_{jk} in $O(N_{trd}^2)$ time. This procedure is repeated for the remaining hash functions. In total one iteration through all K hash functions takes $O(KN_{trd}^2 + KN_{trd}\log N_{trd})$ operations[17]. BRE meets properties E_1-E_2 of an effective hashcode. The hashcode bits generated by BRE are correlated (property E_3 is not conserved) given that the coordinate descent algorithm cycles through each hash function in turn updating the current hash function based on the optimised hyperplane normal vectors of previously examined hash functions. The benefit of tackling the discrete optimisation problem directly has recently garnered renewed attention in Liu et al. (2014) and Shen et al. (2015).

2.6.4.3 Supervised Hashing with Kernels (KSH)

Supervised Hashing with Kernels (KSH) formulates a kernelised hash function in a similar manner to AGH (Section 2.6.3.4) and BRE (Section 2.6.4.2) but proposes an entirely different and spectrally relaxed optimisation algorithm (Liu et al. (2012)). KSH exhibits the highest retrieval effectiveness compared to the supervised hashing models I discuss in this section and frequently appears in the literature as the de-facto baseline for comparison on the standard image datasets considered in this thesis. The familiar kernelised hash function is presented in Equation 2.41

[17]For ease of presentation I assume each of the N_{trd} training data-points forms N_{trd}-1 supervisory pairs with the other N_{trd}-1 training data-points in $\mathbf{S}$. In practice, for computational tractability, BRE randomly selects a much smaller sample of pairs (e.g. $0.05N_{trd}$) for each training datapoint.

$$h_k(\mathbf{q}) = sgn(\sum_{j=1}^{C} W_{jk}\kappa(\mathbf{x}_j, \mathbf{q}) + t_k) \tag{2.41}$$

where κ is the kernel function $\kappa : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$, $t_k \in \mathbb{R}$ is a scalar threshold and $\mathbf{W} \in \mathbb{R}^{C \times K}$ is a set of K hyperplane normal vectors. As for BRE and AGH a small number of C ($C \ll N$) data-points are sampled uniformly at random from the dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$ to compute the required kernel similarities. In addition, N_{trd} data-points ($C < N_{trd} \ll N$) are sampled from the dataset to construct the adjacency matrix $\mathbf{S} \in \{-1, 1\}^{N_{trd} \times N_{trd}}$, which acts as the training samples for learning the hash functions. The objective function of KSH (Equation 2.42) is very similar to the supervised BRE objective function, the only salient difference being the removal of the sign function and the computation of the inner product $(g^\intercal(\mathbf{x}_i)g(\mathbf{x}_j))$ between a pair of hashcodes for data-points $\mathbf{x}_i, \mathbf{x}_j$, rather than the Euclidean distance.

$$\begin{aligned} &\text{argmin}_{\mathbf{W} \in \mathbb{R}^{C \times K}} \quad \sum_{ij \in \mathbf{S} \in \mathbb{R}^{N_{trd} \times N_{trd}}} \left\{ S_{ij} - \frac{1}{K} g^\intercal(\mathbf{x}_i) g(\mathbf{x}_j) \right\}^2 \\ &\text{subject to } g(\mathbf{x}_i) = [h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \ldots, h_K(\mathbf{x}_i)]^\intercal \\ &h_k(\mathbf{x}_i) = sgn(\sum_{j=1}^{C} W_{jk}\kappa(\mathbf{x}_j, \mathbf{x}_i)) \end{aligned} \tag{2.42}$$

Recall from Section 2.6.4.2 that BRE retains the sign function and tackles the resulting NP-hard optimisation problem via a coordinate descent algorithm that measures the impact of flipping bits on the objective function value. In contrast KSH drops the sign function and performs the hashcode optimisation over a continuous space that admits a more efficient parameter update via gradient descent. KSH optimises each of the K hash functions sequentially by firstly initialising each hyperplane normal $\left\{\mathbf{w}_k \in \mathbb{R}^C\right\}_{k=1}^{K}$ by solving an eigenvalue problem which is then followed by a gradient descent optimisation to further refine the hyperplanes. To see how the KSH sequential optimisation algorithm works more clearly, I drop the sign function and rewrite Equation 2.42 to iterate over the K hash functions rather than data-point pairs (Equation 2.43)

$$\begin{aligned} &\text{argmin}_{\mathbf{W} \in \mathbb{R}^{C \times K}} \quad \sum_{k=1}^{K} \|K\mathbf{S} - \mathbf{y}^k (\mathbf{y}^k)^\intercal\|_F^2 \\ &\text{where } y_i^k = \sum_{j=1}^{C} W_{jk}\kappa(\mathbf{x}_j, \mathbf{x}_i) \end{aligned} \tag{2.43}$$

where I have assumed that the data is mean centered, and therefore $t_k = 0$. Recall from Table A.1 in Appendix A that the notation $\mathbf{y}^k$ signifies the k^{th} *column* of the projection matrix $\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}$. It is possible to approximately solve Equation 2.43 by simply optimising each hyperplane individually giving K independent optimisation problems. Instead KSH opts for a solution strategy similar to that of BRE where the hyperplanes are solved in a sequential manner thereby instilling a degree of dependence between the hashcode bits. In the case of KSH this dependence is captured with a residue matrix $\mathbf{R} \in \mathbb{Z}_{+}^{N_{trd} \times N_{trd}}$ defined in Equation 2.44

$$\mathbf{R}^{k-1} = K\mathbf{S} - \sum_{l=1}^{k-1} \mathbf{y}^l (\mathbf{y}^l)^\intercal \tag{2.44}$$

The magnitude of $\mathbf{R}$ is related to the number of mismatches between the signs of data-point pairs where $S_{ij} = 1$ in the adjacency matrix. The higher the number of mismatches for a given data-point pair $(\mathbf{x}_i, \mathbf{x}_j)$ over the previous k-1 hash functions the greater the value of the corresponding element R_{ij}^{k-1} and the greater the influence that pair will have on learning of the k^{th} hash function. In this way the hash function learning is gradually biased towards correctly labelling those data-point pairs that were incorrectly labelled by hyperplanes learnt earlier in the optimisation procedure. Liu et al. (2012) show that the objective function in Equation 2.43 can be reduced to Equation 2.45

$$\begin{aligned} \text{argmax}_{\mathbf{w}_k \in \mathbb{R}^C} \quad & (\mathbf{K}\mathbf{w}_k)^\intercal \mathbf{R}^{k-1} (\mathbf{K}\mathbf{w}_k) \\ \text{where } K_{ij} &= \kappa(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to } & (\mathbf{K}\mathbf{w}_k)^\intercal (\mathbf{K}\mathbf{w}_k) = L \end{aligned} \tag{2.45}$$

where $\mathbf{K} \in \mathbb{R}^{N_{trd} \times C}$ is the kernel matrix. Comparing the form of Equation 2.45 to the standard eigenvalue problem template presented in Equation 2.24 we can immediately see that the solution to this optimisation problem is the eigenvector with the largest eigenvalue of $\mathbf{K}^\intercal \mathbf{R}^{k-1} \mathbf{K}\mathbf{w}_k = \lambda \mathbf{K}^\intercal \mathbf{K}\mathbf{w}_k$. In the KSH algorithm this eigenvector constitutes the initialisation point for the k^{th} hyperplane normal $\mathbf{w}_k \in \mathbb{R}^C$. The position of this hyperplane is further refined via gradient descent from the gradient of a sigmoid smoothed relaxation of Equation 2.45. The remaining hashing hyperplanes are then learnt by updating the residue matrix and sequentially repeating the eigenvector initialisation and gradient descent refinement steps for each.

Despite being a non-linear model, KSH maintains a computationally tractable op-

timisation algorithm with time complexity $O(NCK + N_{trd}^2 CK + N_{trd} C^2 K + C^3 K)$ by limiting the number C, N_{trd}[18] of sampled data-points used to construct the hash functions and by making a continuous (real-valued) approximation to the binary hashcodes. KSH does not enforce constraints E_3-E_4 of an effective hashcode, but does ensure E_1, E_2 with highly discriminative hashcodes and fast out-of-sample-extension to unseen query data-points.

2.6.4.4 Self-Taught Hashing (STH)

Self-taught hashing (STH) (Zhang et al. (2010b)) employs a two-step procedure for learning the hashing hyperplanes. The first step involves a Laplacian Eigenmap dimensionality reduction which is followed by a second step that learns the hyperplanes for out-of-sample extension to unseen query data-points. STH is therefore reminiscent of the unsupervised data-dependent hashing models Anchor Graph Hashing (AGH) (Section 2.6.3.4) and Spectral Hashing (SH) (Section 2.6.3.2). The first step of STH is identical to that of SH in which K graph Laplacian eigenvectors are extracted from the graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{S}$. I present the now familiar graph Laplacian optimisation objective in Equation 2.46

$$\begin{aligned} \text{argmin}_{\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}} \quad & tr(\mathbf{Y}^\intercal (\mathbf{D} - \mathbf{S}) \mathbf{Y}) \\ \text{subject to } & \mathbf{Y} \in \mathbb{R}^{N_{trd} \times K} \\ & \mathbf{Y}^\intercal \mathbf{D} \mathbf{1} = \mathbf{0} \\ & \mathbf{Y}^\intercal \mathbf{D} \mathbf{Y} = N_{trd} \mathbf{I}^{K \times K} \end{aligned} \tag{2.46}$$

where $tr(A) = \sum_i A_{ii}$ is the trace operator, $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$ is a neighbourhood graph formed from class labels, if two data-points share at least one class in common then $S_{ij} = 1$, otherwise $S_{ij} = 0$ and D is the diagonal degree matrix $D_{ii} = \sum_j S_{ij}$. For computational tractability $N_{trd} \ll N$. Note the slight difference in the constraints between Equation 2.46 and the objective of SH (Equation 2.28). The diagonal degree matrix $\mathbf{D}$ makes an appearance in the constraints of Equation 2.46, which gives a normalised cut of $\mathbf{S}$ rather than a ratio-cut (Aggarwal and Reddy (2014)) when the graph Laplacian eigenvectors are binarised. Equation 2.46 is therefore equivalent to the Laplacian Eigenmap embedding (Belkin and Niyogi (2003)). The solutions of Equation 2.46 are the eigenvectors corresponding to the lowest eigenvalues of the generalised eigenvalue

[18] Typically $N_{trd} = 1000$ and $C = 300$.

problem $L\mathbf{y}^k = \lambda \mathbf{D}\mathbf{y}^k$. The eigenvalue problem can be solved in $O(MN_{trd}^2K)$ operations using M iterations of the Lanczos algorithm (Golub and Van Loan (1996), Zhang et al. (2010b)). Note that, as for BRE (Section 2.6.4.2), it is also straightforward to frame STH as an unsupervised hashing model by computing $\mathbf{S}$ using, for example, the Euclidean distance between feature vectors in the input feature space. In the same way to SH, solving Equation 2.46 approximately preserves properties E_3 and E_4 of an effective hashcode (Section 2.6.1).

Equation 2.46 can be solved as a standard eigenvalue problem to extract the required K graph Laplacian eigenvectors $\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}$. As I discussed in the context of AGH, the spectral embedding matrix must be binarised to form the hashcodes, and only then provides the encoding for the N_{trd} data-points that formed the neighbourhood graph $\mathbf{S}$. Rather than appealing to the Nyström method (Bengio et al. (2004); Williams and Seeger (2001)), as in AGH (Section 2.6.3.4) or making a separable uniform distribution approximation as for SH (Section 2.6.3.2), STH makes the novel contribution of learning a set of K binary support vector machine (SVM) classifiers that predict the bits in the binarised spectral embedding matrix with maximum margin. The learnt classifiers provide the required hyperplane normal vectors $\left\{\mathbf{w}_k \in \mathbb{R}^D\right\}_{k=1}^K$ necessary for out-of-sample extension to unseen data-points. Training K linear SVMs takes $O(N_{trd}DK)$ time (Joachims (2006)) while out-of-sample extension (test time) is $O(DK)$ for a single test data-point.

2.6.4.5 A Brief Summary

I have reviewed four of the most prevalent methods in the literature for injecting a supervised signal into the learning of the hashing hyperplanes for unimodal ANN search. The methods reviewed included ITQ+CCA (Section 2.6.4.1), Binary Reconstructive Embedding (BRE) (Section 2.6.4.2), Supervised Hashing with Kernels (KSH) (Section 2.6.4.3) and Self Taught Hashing (STH) (Section 2.6.4.4). The underlying principle behind all of these methods is to learn a set of K hyperplanes that are informed by must-link or cannot-link constraints on data-point pairs. The hyperplanes should not partition must-link pairs, but should partition cannot-link pairs into distinct hashtable buckets. An example must-link constraint would be for two images of a cat to be placed in the same bucket, while a cannot-link constraint would demand that an image of a dog be placed in a separate bucket. We saw how these methods differ at a high-level only in how the available labels are compared to the projections/hashcodes so as to compute an error signal for further adjustment of the hashing hyperplanes. KSH and

BRE, for example, seek to minimise the difference between the label and either the inner product of the projections of the two data-points (KSH) or the Hamming distance between their binarised hashcodes (BRE). Despite the conceptual similarity between the objective functions, the optimisation algorithms used in their solution were substantially different and formed perhaps the most interesting point of departure between the different hashing models reviewed in this section. BRE for example attempted to optimise the hashing hyperplanes by remaining within the discrete hashcode space, thereby directly tackling an NP-hard optimisation problem. In contrast, KSH relaxed the objective into a continuous domain and used a gradient descent procedure to learn the hashing hyperplanes.

2.6.5 Cross-Modality Projection Methods

Locality sensitive hashing (LSH) and its kernelised variant SKLSH which were both described in Section 2.4 and Section 2.6.2.1 and the data-dependent hashing models presented in Sections 2.6.3-2.6.4 are all confined to *unimodal* retrieval where the queries and the database have identical feature representations. This means that the learnt hyperplanes only partition (bucket) the data-space from that single feature representation. This is a rather limiting restriction of many existing hashing models because much of the data found today, particularly on the internet, is associated with multiple modalities[19]. For example, consider an image from the popular photo sharing website Flickr[20] which is not only described by the raw pixel values themselves, but also with associated tags assigned by users and geolocation information sourced from the GPS system on the camera. It would clearly be very useful if we could pose a query in the form of an image and retrieve relevant tags (Figure 2.19), or give the retrieval system geographical coordinates and receive images related to that locality.

The data-dependent hashing models I describe in this section are able to hash related data-points existing across two modalities into the same hashtable buckets, thereby bringing the computational advantages of approximate nearest neighbour search to multi-modal retrieval. Denote as $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D_x}$ the feature descriptors in modality $\mathcal{X}$ and $\mathbf{Z} \in \mathbb{R}^{N_{trd} \times D_z}$ the feature descriptors in modality $\mathcal{Z}$, where usually the dimensionalities are not equal $D_x \neq D_z$. For simplicity of description I assume that both datasets have the same number of training data-points N_{trd}, and I further denote as N_{xz} the number of *paired* data-points across the modalities ($N_{xz} \leq N_{trd}$). The logical relationship

[19] I use the term 'modality' and 'feature space' interchangeably in this thesis.
[20] http://www.flickr.com

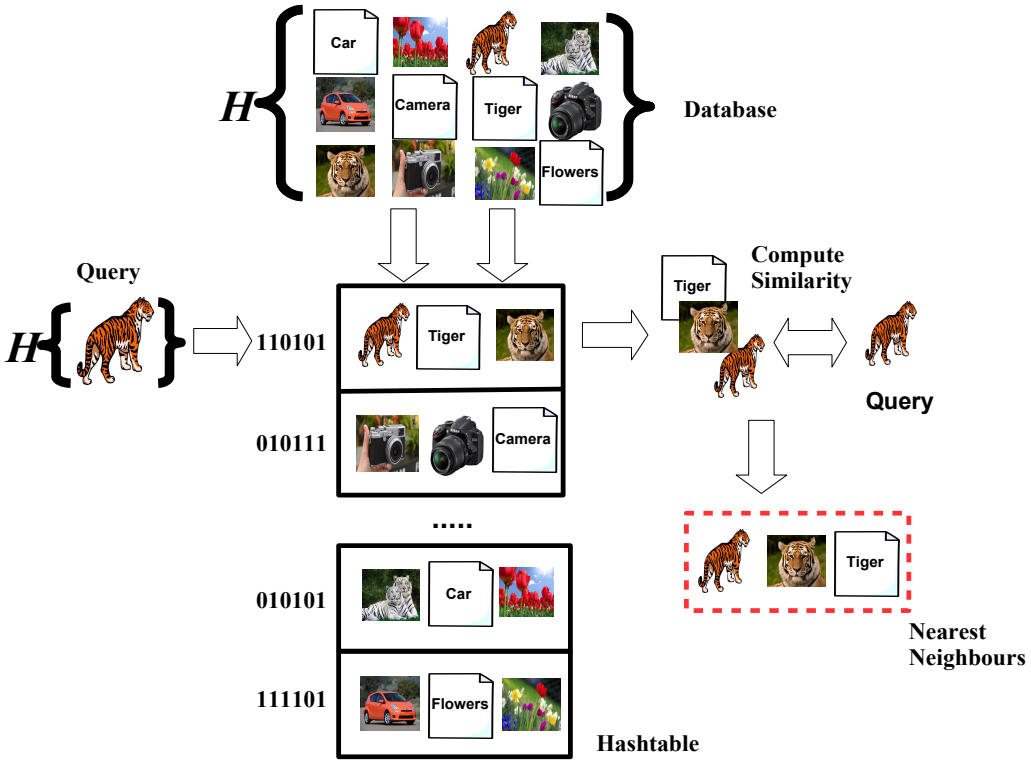
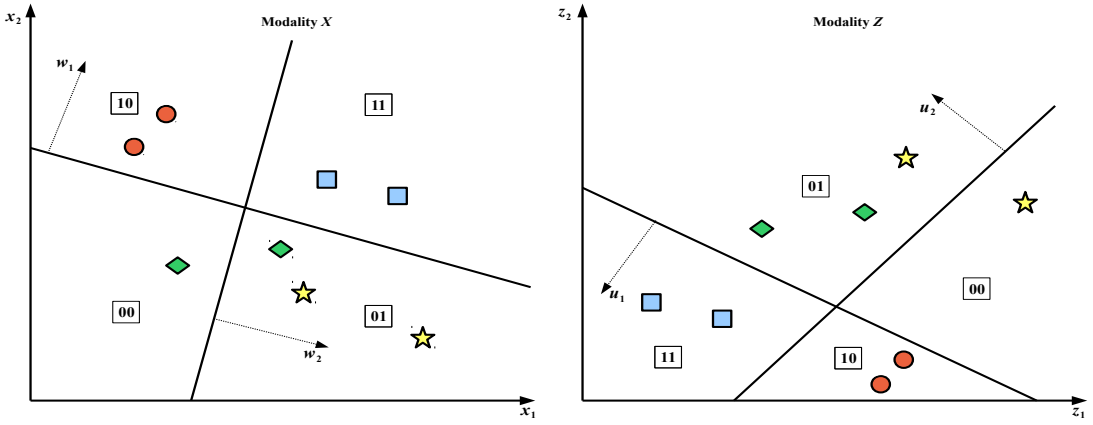


Figure 2.19: Cross-modal hashing-based ANN search. In the cross-modal variant of hashing-based ANN search we wish to partition the input-space such that similar data-points across modalities fall into the same hashtable buckets. In this diagram I show how cross-modal hash functions can be used to retrieve similar images and documents to a query image in constant time. The cross-modal hash functions $\mathcal{H}$ assign similar hashcodes to similar images and documents thereby allowing similar data-points in different modalities to collide in the same hashtable buckets.

between the data-points is encoded in an adjacency matrix $\mathbf{S} \in \{0,1\}^{N_{xz} \times N_{xz}}$, where $S_{ij} = 1$ indicates that pair $(\mathbf{x}_i, \mathbf{z}_j)$ are related, and 0 otherwise. At a high level all five of these models attempt to learn *two* sets of K hyperplanes denoted as $\mathbf{W} \in \mathbb{R}^{D_x \times K}$ and $\mathbf{U} \in \mathbb{R}^{D_z \times K}$, one set for feature space $\mathcal{X}$ and another for feature space $\mathcal{Z}$, such that similar data-points ($S_{ij} = 1$) *across* the two modalities receive similar hashcodes $d_{hamm}(g_{\mathcal{X}}(\mathbf{x}_i), g_{\mathcal{Z}}(\mathbf{z}_j)) \approx 0$, and vice-versa for dissimilar data-points ($S_{ij} = 0$). Here $\left\{g_x : \mathbb{R}^D \rightarrow \{0,1\}^K\right\}$ is the binary embedding function formed from the concatenation of K hash functions $\left\{h_k^{\mathcal{X}} : \mathbb{R}^D \rightarrow \{0,1\}\right\}_{k=1}^{K}$ for modality $\mathcal{X}$, and similar for modality $\mathcal{Z}$. This is a logical extension of the unimodal case in which we not only wish to make similar data-points *within* a modality fall into the same hashtable buckets (e.g. two images of a cat), but also similar data-points *across* the two modalities (e.g. an image of



(a) **Modality** $\mathcal{X}$ (b) **Modality** $\mathcal{Y}$

Figure 2.20: The essence of learning to hash across modalities. In Figure (a) I show the first modality $\mathcal{X}$ (e.g. image descriptor space) with data-points $\left\{\mathbf{x}_i \in \mathbb{R}^{D_x}\right\}_{i=1}^{N}$ and hyperplane normal vectors $\left\{\mathbf{w}_k \in \mathbb{R}^{D_x}\right\}_{k=1}^{K}$. In Figure (b) I show a different feature space $\mathcal{Z}$ (e.g. textual annotations) with data-points $\left\{\mathbf{z}_i \in \mathbb{R}^{D_z}\right\}_{i=1}^{N}$ and hyperplane normal vectors $\left\{\mathbf{u}_k \in \mathbb{R}^{D_z}\right\}_{k=1}^{K}$. Similar data-points within and across modalities are indicated by the same colour and shape. The goal of cross-modal hashing is to position the two sets of hyperplanes in such a way that they assign the same hashcodes to the same data-points both within and across the two modalities.

a cat and a text snippet describing a cat). In Section 2.4 and Sections 2.6.3-2.6.4, I discussed how to learn K hyperplanes that assign similar data-points similar hashcodes in the *same* modality. I saw how this is achieved by positioning the hashing hyperplanes in the input space in a way that attempts to maximise the number of true nearest neighbours within the same buckets. In this section we will see how this notion can be extended to learning *two sets* of K hyperplanes that generate similar hashcodes for related data-points in two different modalities. In practice this boils down to augmenting the objective function with a *consistency* term that ensures the two sets of hyperplanes agree on their hashcode output for similar cross-modal data-points. I provide an intuitive high-level overview of this fundamental concept in Figure 2.20.

I use the same strategy, namely a freely available codebase and widely referenced publication, to select appropriate baselines for cross-modal retrieval as I did for the unimodal baselines described in Sections 2.6.3-2.6.4. The selected baselines are the seminal Cross View Hashing (CVH) model of Kumar and Udupa (2011) (Section 2.6.5.1), Co-Regularised Hashing (CRH) (Zhen and Yeung (2012)) (Section 2.6.5.2), Predicable Dual View Hashing (PDH) (Rastegari et al. (2013)) (Section 2.6.5.4), Inter-

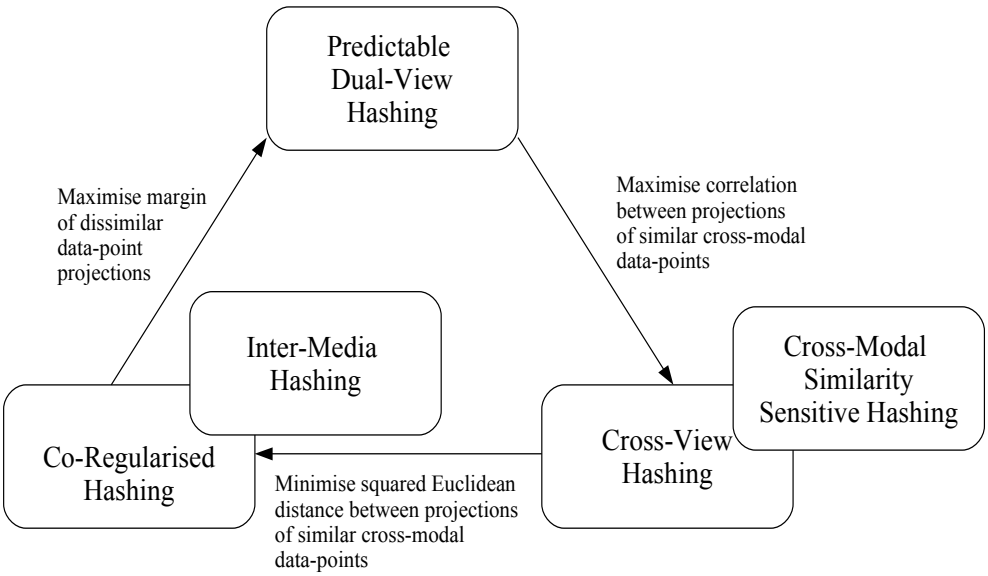


Figure 2.21: Relationship between the five cross-modal hashing models reviewed in this section. I only consider the inter-modal consistency term when relating the models (ignoring intra-modal and out-of-sample extension terms). The labels on the arcs denote the essential transform required to convert one model into the model(s) pointed to by the arc.

Media Hashing (IMH) (Song et al. (2013)) (Section 2.6.5.5) and Cross Modal Semi-Supervised Hashing (CMSSH) (Bronstein et al. (2010)) (2.6.5.3).

2.6.5.1 Cross View Hashing (CVH)

Cross View Hashing (CVH)[21] (Kumar and Udupa (2011)) is equivalent to $ITQ+CCA$ (Section 2.6.4.1) in its use of Canonical Correlation Analysis (CCA) to find two sets of hyperplanes that maximise the correlations of the projections from two different modalities. There are two differences to $ITQ+CCA$: firstly, CVH retains both sets of hyperplane normals $\mathbf{W} \in \mathbb{R}^{D_x \times K}$ and $\mathbf{U} \in \mathbb{R}^{D_z \times K}$, rather than only using the set pertaining to the visual modality; secondly, CVH does not involve a post-processing step that rotates the input feature space to balance the variance captured across the hyperplanes. The hash function for CVH is the standard linear hash function. Equation 2.47 presents the hash functions for both modalities

[21] As is standard in the literature I consider the special case of CVH where only cross-modality supervision is available and each data-point is paired with only one other in the opposing modality (Section 3.2 in Kumar and Udupa (2011)).

$$h_k^{\mathcal{X}}(\mathbf{x}_i) = \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{x}_i))$$
$$h_k^{\mathcal{Z}}(\mathbf{z}_i) = \frac{1}{2}(1 + sgn(\mathbf{u}_k^\intercal \mathbf{z}_i)) \tag{2.47}$$

Following the same argument as for ITQ+CCA, the asymptotic computational complexity of CVH is $O(N_{trd}D^2 + D^3)$ where $D = max(D_x, D_z)$. CVH was one of the first proposed cross-modal hashing models to be proposed in the literature and typically features in previous research as the de-facto baseline for comparison. The cross-modal hashing models I will review in Sections 2.6.5.2-2.6.5.5 introduce new schemes for learning both sets of hyperplane that achieve a higher retrieval effectiveness than CVH on standard image-text datasets.

2.6.5.2 Co-Regularised Hashing (CRH)

Co-Regularised Hashing (CRH) learns $2K$ cross-modal hash functions by solving K individual max-margin optimisation problems sequentially (Zhen and Yeung (2012)). Boosting (Freund and Schapire (1997)) is used in each step to coordinate the learning of the hash functions so that the pairwise constraints not met by hyperplanes constructed earlier in the optimisation sequence have a gradually higher likelihood of being met by subsequent hyperplanes. This brings about a dependence between the bits, a trait we have seen before in the context of the unimodal data-dependent hashing model KSH (Section 2.6.4.3). CRH uses the standard linear hash function (Equation 2.47) as for CVH (Section 2.6.5.1). The objective function for learning the two hyperplane normals $\mathbf{w}_k \in \mathbb{R}^{D_x}, \mathbf{u}_k \in \mathbb{R}^{D_z}$ pertaining to the same bit in modalities $\mathcal{X}$, $\mathcal{Z}$ is made up of three main terms: one to position the hyperplane normal $\mathbf{w}_k$ in modality $\mathcal{X}$, another to position the hyperplane normal $\mathbf{u}_k$ in modality $\mathcal{Z}$ and a third consistency term that forces both hyperplanes to give similar projections for related data-points. The CRH objective is presented in Equation 2.48

$$\begin{aligned}\text{argmin}_{\mathbf{w}_k \in \mathbb{R}^{D_x}, \mathbf{u}_k \in \mathbb{R}^{D_z}} \quad & \frac{1}{N_x}\sum_{i=1}^{N_x}[1 - |\mathbf{w}_k^\intercal \mathbf{x}_i|]_+ + \frac{1}{N_z}\sum_{j=1}^{N_z}[1 - |\mathbf{u}_k^\intercal \mathbf{z}_j|]_+ \\ & \gamma \sum_{i,j=1}^{N_{xz}} \alpha_{ij} S_{ij} (\mathbf{w}_k^T \mathbf{x}_i - \mathbf{u}_k^\intercal \mathbf{z}_j)^2 + \frac{\lambda_x}{2}\|\mathbf{w}_k\|^2 + \frac{\lambda_z}{2}\|\mathbf{u}_k\|^2\end{aligned} \tag{2.48}$$

where $\left\{\alpha_{ij} \in \mathbb{R}_+\right\}_{i,j=1}^{N_{trd}}$ are weights updated using Adaboost (Freund and Schapire (1997)), $\lambda_x \in \mathbb{R}_+, \lambda_z \in \mathbb{R}_+$ are regularisation constants, $[a]_+$ is equal to a if $a \geq 0$,

and 0 otherwise, and $\gamma \in \mathbb{R}_+$ is an scalar governing the importance of the cross-modal term. The intra-modality loss terms guide the projections to be away from zero by a margin so that the data-points do not lie too close to the dividing hyperplanes, thereby encouraging generalisability of the hash functions. The inter-modal loss term is intuitive in its attempt to minimise the squared difference between the projections of similar data-points across modalities[22].

Equation 2.48 is non-convex and so Zhen and Yeung (2012) minimise it in an alternate manner by solving two sub-problems: fixing $\mathbf{w}_k \in \mathbb{R}^{D_x}$ and optimising for $\mathbf{u}_k \in \mathbb{R}^{D_z}$ and vice-versa. In practice this is achieved using the Concave-Convex Procedure (CCVP) (Yuille and Rangarajan (2003)) by framing each sub-problem as a difference of convex functions. Having learnt hyperplane normals $\mathbf{w}_k$, $\mathbf{u}_k$ at the k^{th} step, the weights $\{\alpha_i \in \mathbb{R}_+\}_{i=1}^{N_{trd}}$ for the point-pairs are updated using the standard Adaboost framework (Freund and Schapire (1997)), in which the error term for Adaboost is based upon a count of the number of times the outputs of the cross-modal hash functions disagree. This entire procedure is then repeated with Equation 2.48 solved for hyperplanes $\mathbf{w}_{k+1}$, $\mathbf{u}_{k+1}$ using the updated Adaboost weights. The computational time complexity of CRH is bounded by $O(KMND)$ where $D = max(D_x, D_z)$ and M is the number of iterations required for convergence of the Pegasos solver (Shalev-Shwartz et al. (2007)). Properties E_1, E_2 of an effective hashcode (Section 2.6.1) are preserved by CRH but not properties E_3, E_4.

2.6.5.3 Cross-Modal Similarity Sensitive Hashing (CMSSH)

Cross-Modal Similarity Sensitive Hashing (CMSSH) (Bronstein et al. (2010)) presents a considerably simpler optimisation framework compared to CRH (Section 2.6.5.2) focusing entirely on ensuring the output of the cross-modal hash functions are consistent with each other, but without any specific terms for optimising the intra-modal similarity. CMSSH learns the $2K$ hash functions using a sequential procedure where, at the k^{th} step, two hyperplane normal vectors $\mathbf{w}_k \in \mathbb{R}^{D_x}, \mathbf{u}_k \in \mathbb{R}^{D_z}$ are computed with the weighted objective function presented in Equation 2.49 that effectively coerces the k^{th} pair of hash functions to correct mistakes committed by the k-1 previous hash functions

$$\text{argmax}_{\mathbf{w}_k \in \mathbb{R}^{D_x}, \mathbf{u}_k \in \mathbb{R}^{D_z}} \quad \sum_{i,j=1}^{N_{xz}} \alpha_{ij} S_{ij} sgn(\mathbf{w}_k^\intercal \mathbf{x}_i) sgn(\mathbf{u}_k^\intercal \mathbf{z}_j) \tag{2.49}$$

[22] In practice CRH also has an additional term that pushes dissimilar points further apart. I omit this here for clarity and conciseness of explanation.

where $\{\alpha_{ij} \in \mathbb{R}_+\}_{i,j=1}^{N_{xz}}$ are per data-point pair weights adjusted using Adaboost (Freund and Schapire (1997)). CMSSH is similar to CVH in that the correlation of the projections of related cross-modal data-points is maximised. As it stands Equation 2.49 is discontinuous and therefore non-differentiable making it difficult to optimise with the sign function intact. Bronstein et al. (2010) therefore make the standard spectral relaxation which we have previously seen in many other data-dependent hashing models such as KSH (Section 2.6.4.3) and SH (Section 2.6.3.2). This relaxation simply involves dropping the sign function altogether giving Equation 2.50.

$$\begin{aligned} \text{argmax}_{\mathbf{w}_k \in \mathbb{R}^{D_x}, \mathbf{u}_k \in \mathbb{R}^{D_z}} & \sum_{i,j=1}^{N_{xz}} \alpha_{ij} S_{ij} (\mathbf{w}_k^\intercal \mathbf{x}_i)(\mathbf{u}_k^\intercal \mathbf{z}_j) \\ = & \ \mathbf{w}_k^\intercal \left\{ \sum_{ij=1}^{N_{xz}} \alpha_{ij} S_{ij} \mathbf{x}_i \mathbf{z}_j^\intercal \right\} \mathbf{u}_k \\ = & \ \mathbf{w}_k^\intercal \mathbf{C} \mathbf{u}_k \end{aligned} \tag{2.50}$$

Equation 2.50 can be solved in closed form by performing an SVD on the matrix $\mathbf{C} \in \mathbb{R}^{D_x \times D_z}$ taking $O(D_x^2 D_z + D_x D_z^2 + D_z^3)$ operations[23]. The per pair weights $\{\alpha_{ij} \in \mathbb{R}_+\}_{i,j=1}^{N_{xz}}$ are then updated using Adaboost to emphasise the misclassified data-point pairs and the step repeated for the k+1 set of hyperplanes. The learnt hash functions are used in the standard linear hash function (Equation 2.47) to generate binary hashcodes for multimodal data. In a similar manner to CRH, CMSSH maintains properties E_1, E_2 of an effective hashcode, but does not seek to conserve property E_4 due to the sequential dependence of hashcode bits induced through boosting.

2.6.5.4 Predictable Dual-View Hashing (PDH)

Predictable Dual-View Hashing (PDH) (Rastegari et al. (2013)) is the closest cross-modal hashing model to my own contribution outlined in Chapter 6 and in Moran and Lavrenko (2015b). Given this close relationship I present the full specification of the PDH model in Algorithm 5 so that it is straightforward to compare and contrast both models. Similar to CRH and CMSSH, PDH solves for the $2K$ hashing hyperplanes sequentially by solving for a pair of hyperplane normal vectors $\mathbf{w}_k \in \mathbb{R}^{D_x}, \mathbf{u}_k \in \mathbb{R}^{D_z}$ at the k^{th} step. PDH solves for the hyperplanes using an SVM-based formulation in which

[23] As $\mathbf{C} \in \mathbb{R}^{D_x \times D_z}$ may be non-square the solution is obtained via an SVD rather than the standard eigenvalue problem used for the unimodal data-dependent hashing models described in Section 2.6.3.

the hyperplanes are trained to partition data-points with opposing bits with maximum margin. Different to these previously reviewed models, PDH does not induce a dependence between bits using boosting but instead explicitly attempts to enforce property E_4 of an effective hashcode, namely that the bits should be pairwise independent. I present the PDH objective function in Equation 2.51

$$\begin{aligned}
\text{argmin}_{\mathbf{W}\in\mathbb{R}^{D_x\times K},\mathbf{U}\in\mathbb{R}^{D_z\times K}} \quad & \|\mathbf{B}^x\mathbf{B}^x-\mathbf{I}\|_2^2+\|\mathbf{B}^z\mathbf{B}^z-\mathbf{I}\|_2^2+\sum_{k=1}^{K}\|\mathbf{w}_k\|^2+\sum_{k=1}^{K}\|\mathbf{u}_k\|^2 \\
& +C_x\sum_{i,k=1}^{N_{trd}}\xi_{ik}^x+C_z\sum_{i,k=1}^{N_{trd}}\xi_{ik}^z \\
\text{subject to } & \mathbf{B}^x=sgn(\mathbf{XW}) \\
& \mathbf{B}^z=sgn(\mathbf{ZU}) \\
& b_{ik}^z(\mathbf{w}_k^\intercal\mathbf{x}_i)\geq 1-\xi_{ik}^x \\
& b_{ik}^x(\mathbf{u}_k^\intercal\mathbf{z}_i)\geq 1-\xi_{ik}^z
\end{aligned} \tag{2.51}$$

where $\xi_{ik}^x\in\mathbb{R}$, $\xi_{ik}^z\in\mathbb{R}$ are slack variables that allow some points $\mathbf{x}_i,\mathbf{z}_i$ to fall on the wrong side of hyperplanes with normal vectors $\mathbf{w}_k,\mathbf{u}_k$ and $C_x\in\mathbb{R}_+$, $C_z\in\mathbb{R}_+$ are parameters that permit a trade off between the size of the margins $\frac{1}{||\mathbf{w}_k||},\frac{1}{||\mathbf{u}_k||}$ against the number of points misclassified by $\mathbf{w}_k,\mathbf{u}_k$. The first two terms of the objective function enforce the constraint that the bits should be pairwise independent (property E_3). The last two constraints are reminiscent of the standard SVM max-margin objective with the hashcode bits b_{ik}^x,b_{ik}^z in this case acting as the requisite target labels. Note the subtle but important feature of these constraints where the hashcode bits (b_{ik}^x,b_{ik}^z) for one feature space are used as the targets for hyperplanes existing in the other feature space. This means that over multiple iterations the hyperplanes in both feature spaces should become more consistent in their projections for similar and dissimilar data-points.

Rastegari et al. (2013) solve Equation 2.51 by dividing it into multiple steps as highlighted in Algorithm 5. Firstly, using the bits of the hashcodes in $\mathbf{B}^x\in\{-1,1\}^{N_{trd}\times K}$, $\mathbf{B}^z\in\{-1,1\}^{N_{trd}\times K}$ are used as the labels to train $2K$ SVM classifiers (Lines 6, 10 in Algorithm 5). This step computes an initial estimate of hashing hyperplanes $\mathbf{W}\in\mathbb{R}^{D_x\times K}$, $\mathbf{U}\in\mathbb{R}^{D_z\times K}$. The bits in $\mathbf{B}^x,\mathbf{B}^z$ are then re-labelled with the learnt SVMs (Lines 9, 13 in Algorithm 5), which flips the sign of those data-points that happened to fall on the wrong side of the respective hyperplanes. The pairwise independence property between the bits is approximately enforced by solving the familiar graph Laplacian eigenvalue problem in Equation 2.52

Algorithm 5: PREDICTABLE DUAL VIEW HASHING (PDH) (RASTEGARI ET AL. (2013))

Input: Data-points $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D_x}$, $\mathbf{Z} \in \mathbb{R}^{N_{trd} \times D_z}$, Iterations M

Output: Hyperplanes $\mathbf{W} \in \mathbb{R}^{D_x \times K}$, $\mathbf{U} \in \mathbb{R}^{D_z \times K}$

1 Initialise $\mathbf{W}$, $\mathbf{U}$ via CCA from $\mathbf{X}$, $\mathbf{Z}$
2 **for** $m \leftarrow 1$ ***to*** M **do**
3 **for** $k \leftarrow 1$ ***to*** K **do**
4 $\mathbf{b}_k^x = \mathbf{B}_{\bullet k}^x$
5 $\mathbf{b}_k^z = \mathbf{B}_{\bullet k}^z$
6 Train SVM_k^x with $\mathbf{b}_k^z$ as labels, training dataset $\mathbf{X}$
7 Obtain hyperplane $\mathbf{h}_k^x$
8 $\mathbf{W}_{\bullet k} = \mathbf{w}_k$
9 $\mathbf{B}_{\bullet k}^x = sgn(\mathbf{X}\mathbf{w}_k)$
10 Train SVM_k^z with $\mathbf{b}_k^x$ as labels, training dataset $\mathbf{Z}$
11 Obtain hyperplane $\mathbf{h}_k^z$
12 $\mathbf{U}_{\bullet k} = \mathbf{u}_k$
13 $\mathbf{B}_{\bullet k}^z = sgn(\mathbf{Z}\mathbf{u}_k)$
14 **end**
15 Update $\mathbf{B}^x$, $\mathbf{B}^z$ by solving eigenvalue problem in Equation 2.52.
16 **end**
17 **return** $\mathbf{W}, \mathbf{U}$

$$\begin{aligned} \text{argmin}_{\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}} \quad & tr(\mathbf{Y}_l^\intercal (\mathbf{D}_l - \mathbf{S}_l)\mathbf{Y}_l) \\ \text{subject to } & \mathbf{Y}_l \in \mathbb{R}^{N_{trd} \times K} \\ & \mathbf{Y}_l^\intercal \mathbf{1} = 0 \\ & \mathbf{Y}_l^\intercal \mathbf{Y}_l = N_{trd} \mathbf{I}^{K \times K} \end{aligned} \tag{2.52}$$

where $\mathbf{Y}_l \in \mathbb{R}^{N_{trd} \times K}$ for $l \in \{x, z\}$ are the real-valued (unbinarised) projections for modalities $\mathcal{X}, \mathcal{Z}$ and $\mathbf{S}_l = \mathbf{Y}_l^\intercal \mathbf{Y}_l$, with $\mathbf{D}_{ii} = \sum_j S_{ij}$. As we saw in Section 2.6.3, the solution to this problem is the top K eigenvectors with minimal eigenvalues (Line 15). These K eigenvectors are subsequently binarised to form the updated hashcodes. Intuitively this eigenvalue problem is attempting to lower the pairwise correlation between the data-point projection vectors along the rows of $\mathbf{Y}_l$ for $l \in \{x, z\}$ while maintaining the relative distances between the projection vectors as defined by the inner prod-

uct similarity. These steps are repeated M times until the algorithm has reached a suitable convergence point, at which point the learnt hyperplanes can be used in the standard linear hash function (Equation 2.47) to hash novel cross-modal data-points into hashtable buckets. The training time complexity is dominated by the $O(MN_{trd}^2K)$ operations to solve the eigenvalue problems across M iterations using the Lanczos algorithm (Golub and Van Loan (1996)).

2.6.5.5 Inter-Media Hashing (IMH)

Inter-Media Hashing (IMH) (Song et al. (2013)) can be thought of as a semi-supervised cross-modal hashing model which not only utilises the pairwise supervisory information in the adjacency matrix $\mathbf{S}^{xz} \in \{0,1\}^{N_{xz} \times N_{xz}}$, but also from unsupervised information originating from all N_{trd} data-points *within* each modality, that is, including those data-points that do not feature in $\mathbf{S}^{xz}$. The relationship between these data-points is computed through construction of a Euclidean k-NN graph within each modality. Denote as $\mathbf{S}^{x} \in \{0,1\}^{N_{trd} \times N_{trd}}$ the k-NN graph between data-points in modality $\mathcal{X}$, and similarly for modality $\mathcal{Z}$ where $\mathbf{S}^{z} \in \{0,1\}^{N_{trd} \times N_{trd}}$. Using modality $\mathcal{X}$ as an example, the k-NN graph is constructed as in Equation 2.53

$$S_{ij}^{x} = \begin{cases} 1 & \text{if } \mathbf{x}_i \in NN_k(\mathbf{x}_j) \text{ or } \mathbf{x}_j \in NN_k(\mathbf{x}_i) \\ 0 & otherwise \end{cases} \tag{2.53}$$

where $NN_k(\mathbf{x}_i)$ is a function that returns the set of k-nearest neighbours for data-point $\mathbf{x}_i$ as measured under, for example the Euclidean distance metric.

The IMH semi-supervised approach is to be contrasted with CVH, CMSSH and PDH which learn entirely from the supervisory information in the adjacency matrix $\mathbf{S}$ that pairs related data-points across the two modalities. In this sense IMH, in its full form, bears most resemblance to CRH (Section 2.6.5.2) which also proposes unsupervised terms in the objective function that act as a form of regularisation during the training procedure. When the learning is entirely confined to the labelled data-points in $\mathbf{S}^{xz}$ and further $\mathbf{S}^{xz} = \mathbf{I}^{N_{xy} \times N_{xy}}$, Song et al. (2013) show that IMH is in fact equivalent to the CCA-based CVH model (Section 2.6.5.1). The IMH objective function is presented in Equation 2.54

$$\begin{aligned}\text{argmin}_{\mathbf{W},\mathbf{U},\mathbf{Y}^x,\mathbf{Y}^z} \quad & \lambda \sum_{i,j=1}^{N_{trd}} S_{ij}^x \|\mathbf{y}_i^x - \mathbf{y}_j^x\|_2^2 + \lambda \sum_{i,j=1}^{N_{trd}} S_{ij}^z \|\mathbf{y}_i^z - \mathbf{y}_j^z\|_2^2 + \sum_{i,j=1}^{N_{xz}} S_{ij}^{xz} \|\mathbf{y}_i^x - \mathbf{y}_j^z\|_2^2 \\ & + \sum_{i=1}^{N_{trd}} \|\mathbf{W}^\intercal \mathbf{x}_i - \mathbf{y}_i^x\|_2^2 + \beta \|\mathbf{W}\|_F^2 + \sum_{j=1}^{N_{trd}} \|\mathbf{U}^\intercal \mathbf{z}_j - \mathbf{y}_j^z\|_2^2 + \beta \|\mathbf{U}\|_F^2 \\ & \text{subject to } (\mathbf{Y}^x)^\intercal \mathbf{Y}^x = \mathbf{I}^{D_x \times D_x} \\ & (\mathbf{Y}^x)^\intercal \mathbf{1} = \mathbf{0}\end{aligned} \tag{2.54}$$

where $\mathbf{S}^{xz} = \mathbf{I}^{N_{xz} \times N_{xz}}$ and $\beta \in \mathbb{R}, \lambda \in \mathbb{R}$ are user-specified scalar parameters affecting the importance of the different terms in the objective function.

The IMH objective function is quite intuitive and builds on previous research such as Weiss et al. (2008); Kumar and Udupa (2011); Zhen and Yeung (2012). The first two terms encourage similar data-points within both modalities to have similar projected values and therefore similar hashcodes upon binarisation. It is exactly the graph Laplacian eigenvalue problem (Equation 2.28) we first saw in the context of Spectral Hashing (SH) in Section 2.6.3.2 and extended first to the dual-modality case by CVH (Section 2.6.5.1). The third term encourages the projections of similar data-points across modalities to be the same, which is akin to the inter-modal consistency term used in the CRH model (Section 2.6.5.2) without the Adaboost per-pair weights. The last two terms are out-of-sample extension terms yielding the desired hashing hyperplanes $\mathbf{W} \in \mathbb{R}^{D_x \times K}, \mathbf{U} \in \mathbb{R}^{D_z \times K}$ using the standard L_2 regularised linear regression formulation with the learnt projections for the N_{trd} training data-points as the regression targets. Song et al. (2013) minimise Equation 2.54 by transforming the objective into a trace minimisation problem involving the modality $\mathcal{X}$ projections $\mathbf{Y}^x$. As is customary in the learning to hash literature (Section 2.6.3), the trace minimisation is solved as an eigenvalue problem taking the K eigenvectors with the smallest eigenvalues as the columns of $\mathbf{Y}^x$. Note that solving this eigenvalue problem will achieve the orthogonality constraint in the objective function. Given the solution for the projections in modality $\mathcal{X}$, Song et al. (2013) show that the optimal $\mathbf{Y}^z$ and $\mathbf{W}$, $\mathbf{U}$ can be obtained using closed form formulae based upon the learnt $\mathbf{Y}^x$.

As for all hashing models relying on a matrix factorisation, solving the eigenvalue problem dominates the computational time complexity of IMH requiring $O(N_{trd}^3)$ operations[24]. IMH maintains properties E_1, E_2 of an effective hashcode in both modalities

[24]For the datasets we consider in this thesis, $N_{trd} = 2{,}000$-$10{,}000$ to constrain computation time. In a real-world application N_{trd} be significantly higher.

$\mathcal{X}$, $\mathcal{Y}$, while only maintaining properties E_3, E_4 in modality $\mathcal{X}$ as a result of solving the eigenvalue problem.

2.6.5.6 A Brief Summary

In this section I described five prominent hashing algorithms that are capable of indexing similar data-points that exist in two incommensurable feature spaces into the same hashtable buckets. Specifically, I reviewed Cross-View Hashing (CVH) (Section 2.6.5.1), Co-Regularised Hashing (Section 2.6.5.2), Cross-Modal Semi-Supervised Hashing (Section 2.6.5.3), Predictable Dual-View Hashing (Section 2.6.5.4) and Inter-Media Hashing (Section 2.6.5.5). The essential link between all of these algorithms was the learning of two sets of K hyperplanes, one set of K hyperplanes for each feature space, in a way that encourages the hyperplanes in both spaces to assign similar projected values to similar cross-modal data-points. In all cases this is achieved by framing an optimising an objective function with a cross-modal consistency term that penalises a mismatch between the projected values of similar cross-modal data-points. Some of the more recently proposed cross-modal hashing algorithms (CRH, IMH, PDH) augmented this inter-modal consistency term with additional intra-modal terms that regularise the learning of the hyperplanes by, for example, ensuring that similar within-modality data-points receive similar projected values. The disadvantage of all of these algorithms are their reliance on either an expensive matrix factorisation or non-convex optimisation.

2.7 Conclusion

The purpose of this chapter was to introduce the background information relevant to the topic of this thesis, all of which is important for understanding the novel contributions that will be introduced in later chapters. I began this chapter in Section 2.3 by motivating the need for more efficient algorithms for nearest neighbour (NN) search that do not require an exhaustive brute-force scan of the dataset. This led us to the field of approximate nearest neighbour search which I argued is dominated by the seminal method of Locality Sensitive Hashing (LSH). We saw in Section 2.4 how LSH is in fact a family of different algorithms for generating similarity preserving hashcodes for a wide range of similarity functions of interest, from the inner product similarity to the Euclidean distance. I discussed how the LSH hash function family for the inner prod-

uct similarity forms the focus of this thesis. In this case LSH will generate hashcodes with a low Hamming distance to each other for those data-points that are similar under the inner product similarity. This property enables the hashcodes to be used as indices into the buckets of a set of hashtables to retrieve nearest neighbours in a constant time per query, a much improved query-time versus a brute-force linear scan.

In Sections 2.5-2.6, I then discussed how the contributions in this thesis address the effectiveness of two critical components of the LSH algorithm: projection (hyperplane) learning and binary quantisation. Retrieval effectiveness is highly dependent on how well these two steps preserve the original neighbourhood structure between the data-points in the hashcode Hamming space. Unfortunately, we saw how LSH generates its hyperplanes and quantisation thresholds randomly in the input space relying on asymptotic guarantees that as the number of hyperplanes increases, the desired similarity will be well reflected by the Hamming distance between the binary hashcodes. A random partitioning may lead to the separation of many related data-points into different hashtable buckets, contrary to the central premise of hashing-based ANN search. I described how relaxing this data-independence assumption could mitigate this effect and potentially lead to improved retrieval effectiveness while simultaneously generating more compact hashcodes compared to LSH. This dissertation is not the only research to address this important downside to LSH and so I conducted a review of recently proposed and closely related work within the quantisation and projection branches of the field in Section 2.5 and Section 2.6, respectively. My specific focus was on data-dependent hashing algorithms that learn the hashing hyperplanes and quantisation thresholds in a way that is informed by the distribution of the data, using either an unsupervised (Section 2.6.3) or supervised (Section 2.6.4) signal to avoid placing related unimodal (Sections 2.6.3-2.6.4) or cross-modal (Section 2.6.5) data-points into different hashtable buckets. The reviewed algorithms will form a broad and strong set of baselines in our experimental evaluation presented in later chapters.

Four questions unaddressed by the current literature have been identified by conducting this review, each of which are investigated in subsequent chapters of this thesis. Firstly, the multi-threshold quantisation models that I described in Section 2.5 employ unsupervised learning to position the thresholds. Given this, I explore in Chapter 4 how retrieval effectiveness can be improved by formulating a suitable *semi-supervised* objective function for multiple threshold learning. Secondly, I discovered how most existing quantisation models for hashing are restricted to a *uniform* number of thresholds per projected dimension. To the best of my knowledge there is no existing work

that explores the effect of varying the number of assigned thresholds per projected dimension on retrieval effectiveness. I fill this gap in the current literature in Chapter 5 by proposing a new quantisation model that learns an appropriate allocation of thresholds across projected dimensions based on the neighbourhood preserving quality of the associated hyperplanes. Thirdly, the current breed of supervised projection functions identified in Sections 2.6.4-2.6.5 employ computationally expensive eigen-decomposition and kernel-based optimisation strategies. In Chapter 6, I explore the extent to which a simpler and computationally less expensive optimisation algorithm can compete with these state-of-the-art supervised projection functions, both within the unimodal and cross-modal problem domains. Finally, there is no previous work that learns both the hashing hypersurfaces and *multiple* quantisation thresholds in the same model. I address this knowledge gap in Chapter 7 by combining my quantisation models from Chapters 4-5 with my supervised projection function from Chapter 6.

Having now firmly placed the research described in this thesis in the context of previous related work I will introduce in the next chapter the datasets, evaluation paradigms and evaluation metrics that will be used to judge the quality of nearest neighbour search in my experimental evaluation.