Journal of Artificial Intelligence Research 1 (1993) 1-15 Submitted 6/91; published 9/91

A Primer on Machine Learning Models for Hashing-Based Approximate Nearest Neighbour Search

Sean Moran SEAN.J.MORAN@GMAIL.COM
Edinburgh, Scotland

Abstract

Nearest neighbour search is the problem of finding the most similar data-points to a query in a large database, and is a fundamental operation that has found wide applicability in many fields, from Bioinformatics, through to Natural Language Processing (NLP) and Computer Vision. This survey reviews a host of important approximate nearest neighbour search algorithms that permit constant-time retrieval of nearest neighbours, independent of the dataset size. Typically these algorithms generate similar binary hashcodes for similar data-points, with these hashcodes then used as the indices into the buckets of hashtables, yielding a query time that is substantially improved over an exhaustive comparison. In this survey we explore a host of recently proposed approximate nearest neighbour search algorithms that improve retrieval effectiveness by learning *task specific binary hashcodes*. We categorise the field according to the standard two-step pipeline employed by many existing hashing models, namely projection and quantisation. The generation of a binary hashcode comprises two main steps carried out sequentially: *projection* of the data feature vector onto the normal vectors of a set of hyperplanes that fracture the input feature space followed by a *quantisation* operation that thresholds the projections to generate the binary hashcodes. The degree to which these two operations preserve the relative distances between the data-points in the input feature space has a direct influence on the effectiveness of using the resulting hashcodes for the task of nearest neighbour search. This review departs from the broad and shallow approach of recent surveys by going into considerable depth on the details of a few carefully handpicked hashing models, forming a useful primer of the main concepts for those entering the field and a reference for more established researchers and practitioners alike. An extendible living literature review accompanies this survey and be found at `http://learning2hash.github.io`.

1. Introduction

The strong growth of the World Wide Web (WWW) over the past two decades has brought with it a phenomenal increase in the amount of image, video and text based data being collected, stored and shared across the world. This phenomenon has been fuelled by the popularity of social media networks, cheap disk storage and the wide availability of Internet-enabled smartphones. For example it has been estimated that Facebook has in the order of 300 million images uploaded per day[1], YouTube receives 10 years worth of content per day[2] and there are now estimated to be well over 1 trillion web pages[3] in existence (Murphy, 2012). Figure 1 illustrates the explosive growth of images being uploaded onto popular social media websites and applications during the period 2005-2014. The trend towards real-time

1. Velocity 2012: Jay Parikh, "Building for a Billion Users"
2. `http://www.youtube.com/yt/press/en-GB/statistics.html`
3. `http://googleblog.blogspot.co.uk/2008/07/we-knew-web-was-big.html`

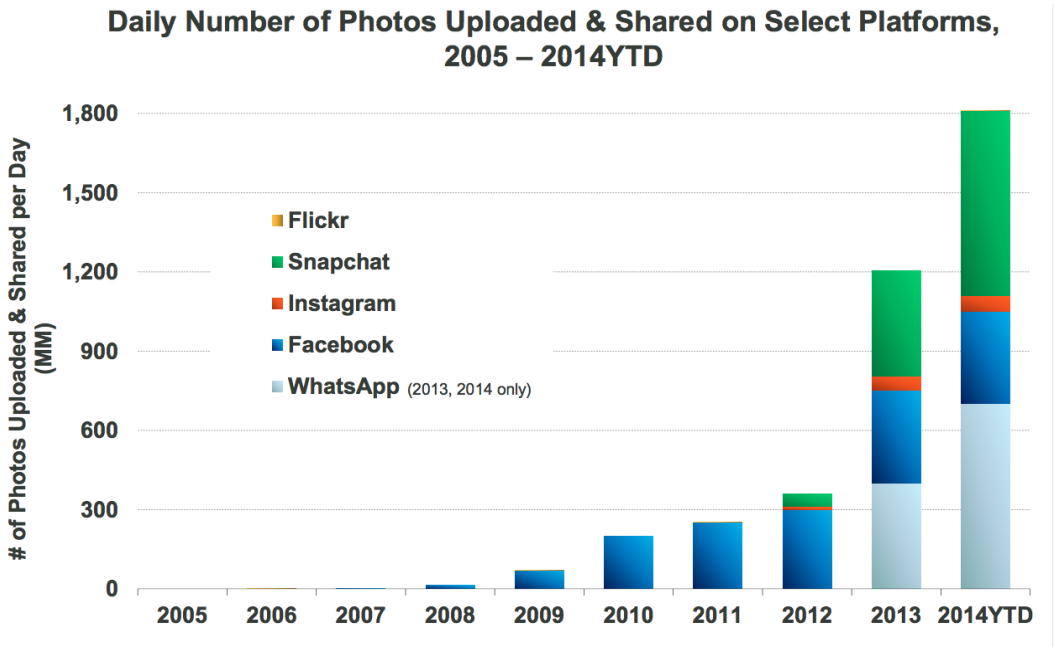


Figure 1: The amount of images being uploaded to popular social media websites (Facebook, Flickr) and mobile applications (Instagram, SnapChat, WhatsApp) has undergone a dramatic growth since 2005. Efficient algorithms for searching through such large image datasets are needed now more than ever. This chart has been copied directly from slide 62 of the talk *"Internet Trends 2014 - Code Conference"* given by the venture capitalist Mary Meeker of Kleiner Perkins Caufield Byers (KPCB): `http://www.kpcb.com/blog/2014-internet-trends`.

video sharing over the Internet with applications such as Periscope, involving a medium that is many times the size of individual images or documents, will severely exacerbate this torrent of data. In the near-term future the emergence of the Internet-of-Things (IoT) and Smart Cities promise to add further fuel to this fire, hinting at a connected society in which Internet linked sensors embedded in everyday objects, such as CCTV cameras and thermostats, produce an abundance of data that is automatically captured, stored and analysed so as to produce actionable insights for interested citizens and government stakeholders (Albakour, Macdonald, & Ounis, 2015). The sheer scale of the data being produced around the world brings with it a need for computationally efficient algorithms that ensure the storage requirements and processing overhead do not grow with the quantity of the data being produced.

In this survey we review the latest methods for performing fast nearest neighbour (NN) search, in which the goal is to find the most similar data-point(s) to a query in a large database without exhaustively comparing the query to all data-points. Similarity is typically judged by representing the data-points as fixed dimensional vectors in a vector space and computing a distance metric such as the Euclidean or cosine distance. In this case data-points with a sufficiently low distance are judged to be nearest neighbours. Due to its generality and usefulness nearest-neighbour search finds application in many areas of Sci-

ence, ranging from the field of Information Retrieval (IR) where we wish to find documents relevant to a query, to the problem of genomic assembly in the field of Bioinformatics. The näive way to solve the nearest-neighbour search problem would be to compare the query to every data-point in the database, a method known as *brute-force search*. Brute-force search is only feasible in relatively small databases where performing the number of required comparisons between the data-points remains computationally tractable. Given the linear scaling of the query time with respect to the dataset size it is impossible to exhaustively search large-scale datasets consisting of millions to billions of data-points for nearest neighbours in a reasonable amount of time. This problem is compounded in the streaming data scenario where data-points need to be processed sequentially in real-time with potentially no end to the amount of incoming data. To efficiently find nearest-neighbours in large-scale datasets, algorithms are required that offer a query time that is independent of the dataset size.

Hashing-based approximate nearest neighbour (ANN) search methods are a popular class of algorithms that permit the nearest neighbours to a query data-point to be retrieved in constant time, independent of the dataset size. Hashing has proved to be an extremely useful method for ANN search over high-dimensional, large-scale datasets that are prevalent in the modern data-rich world. Hashing permits constant time search per query by condensing both the database and the query into fixed-length compact binary hashcodes or fingerprints. The hashcodes exhibit the neighbourhood preserving property that similar data-points will be assigned similar (low Hamming distance) hashcodes. Crucially, unlike cryptographic hash functions such as MD5 or SHA-1, the data-points need not be identical to receive matching hashcodes. Rather the degree of similarity between the hashcodes is a direct function of the similarity between the feature representation of the data-points. This property is particularly ideal for the task of image retrieval where we rarely wish to find only those images that are identical down to the pixel level. Most people would deem two images to be related even if the semantically equivalent objects (e.g. a tiger) depicted in both images are in widely different poses, and therefore the images have a completely different pixel consistency.

This similarity preserving property enables the hashcodes to be used as the keys into the buckets of hashtables so that similar, but not necessarily identical, images will collide in the same buckets (Figure 2). This is a rather different use-case to the typical application of hashtables in Computer Science in which it is imperative to avoid collisions between non-identical data-points. In hashing-based ANN search we are actively encouraging collisions between similar data-points. The bucketing of the data-points drastically reduces the computational overhead of nearest neighbour search by reducing the number of comparisons that are required between the data-points: at query time we need only compare our query to those data-points colliding in the same buckets. There is no free lunch however as we pay for the reduced query time with a non-zero probability of failing to retrieve the closest nearest neighbours in the case where they happen to fall in different buckets. Nevertheless this quantifiable non-zero false negative probability turns out to be an acceptable trade-off in many application areas in which sub-optimal nearest neighbours can be almost as good as finding the exact nearest neighbour (Dean, Ruzon, Segal, Shlens, Vijayanarasimhan, & Yagnik, 2013; Petrović, Osborne, & Lavrenko, 2010).

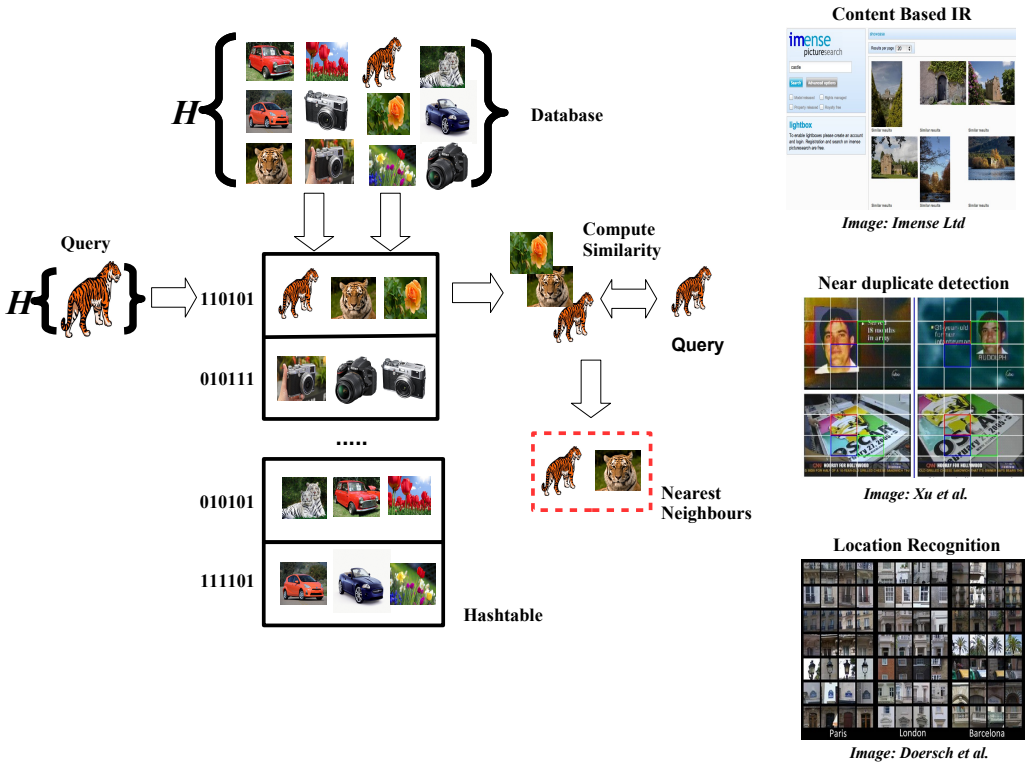


Figure 2: Nearest neighbour search with hashcodes. Similarity preserving binary codes generated by a hash function $\mathcal{H}$ can be used as the indices into the buckets of a hashtable for constant time search. Only those images that are in the same bucket as the query need be compared thereby reducing the size of the search space. The focus of the learning-to-has field is learning the hash function $\mathcal{H}$ to maximise the similarity of hashcodes for similar data-points. On the right-hand side we present examples of tasks for which nearest neighbour search has proved to be fundamental: from content-based information retrieval (IR) to near duplicate detection and location recognition. The three images on the right have been taken from Imense Ltd (`http://www.imense.com`) and (Doersch et al., 2012), (Xu et al., 2010), (Grauman & Fergus, 2013).

Hashing-based ANN has also shown great promise in terms of efficient query processing and data storage reduction across a wide range of interesting application areas within IR and Computer Vision. For example Petrović et al. (Petrović et al., 2010) present an efficient method for event detection in Twitter that scales to unbounded streams through a novel application of Locality Sensitive Hashing (LSH), a seminal randomised approach for ANN search (Indyk & Motwani, 1998). In the streaming data scenario of (Petrović et al., 2010) the $\mathcal{O}(N)$ worst case complexity of inverted indexing is undesirable, motivating the use of LSH to maintain a constant $\mathcal{O}(1)$ query time[4]. Hashing-based ANN has also proved

4. This is only true if we ignore the hashing cost (cost of generating the hashcode) and assume that each database data-point goes into its own hashtable bucket. In practice the LSH computational cost for a single hashtable and a single data-point is a sum of the hashing cost ($\mathcal{O}(KD)$), lookup cost ($\mathcal{O}(1)$) and the candidate test cost ($\mathcal{O}(ND/2^K)$), where K is the hashcode length and assuming a uniform

particularly useful for search over dense and much lower dimensional (compared to text) feature vectors, such as GIST (Oliva & Torralba, 2001), that are commonly employed in the field Computer Vision. In one recent application of LSH within Computer Vision, similarity preserving hashcodes have been successfully used for fast and accurate detection of 100,000 object classes on just a single machine (Dean et al., 2013).

The ANN search hashing models we survey all partition the input feature space into disjoint regions with a set of hypersurfaces, either linear (hyperplanes) or non-linear. In the case of linear hypersurfaces the polytope-shaped regions formed by the intersecting hyperplanes constitute the hashtable buckets (Figure 3). The hashtable key for a data-point is generated by simply determining which side of the hyperplanes the data-point lies. Depending on which side it falls a '0' or a '1' is appended to the hashcode for that data-point. By repeating this procedure for each hyperplane we can build up a hashcode for each data-point that is the same length as the number of hyperplanes partitioning the space. Intuitively, the hashcode can be thought of as an identifier that captures the geometric position of the data-points within the input feature space with each bit encoding the position of the data-point with respect to a given hyperplane. Algorithmically this hashcode generation procedure can be accomplished in two separate steps performed in a pipeline: *projection* followed by *quantisation*. This procedure is illustrated with a toy example in Figure 3. Projection involves a dot product of the feature vector representation of a data-point onto the hyperplane normal vectors positioned either randomly or in data-aware positions in the feature space. The hyperplanes should ideally partition the space in a manner that gives a higher likelihood that similar data points will fall within the same region, and therefore assigned the same hashcode. In the second step the real-valued projections are quantised into binary ('0' or '1') by thresholding the corresponding projected dimensions[5] typically with a single threshold placed at zero for mean centered data.

Despite the success and wide application of algorithms for hashing-based ANN search there still remains considerable downsides to the manner in which the projection and quantisation steps are typically performed. Locality Sensitive Hashing (LSH), one of the arguably most well-known and widely applied methods for hashing-based ANN search, sets the hashing hyperplanes and the quantisation thresholds in a manner that is *independent* of the distribution of the data. For example, in the variant of LSH for preserving the cosine similarity, the normal vectors of the hashing hyperplanes are randomly sampled from a zero mean unit variance multidimensional Gaussian distribution. This data-oblivious mechanism for generating the hashing hypersurfaces runs a high risk of separating dense areas of the feature space and therefore partitioning related data-points into different hashtable buckets (e.g. points a and b in Figure 3). To ameliorate this low recall problem a typical LSH deployment involves partitioning the data with multiple independent hashtables and presenting the union of all the data-points in the colliding hashtable buckets as candidate nearest neighbours. Unfortunately, the greater the number of hashtables the higher the

distribution of data-points to buckets. Observe that there is a trade-off between the hashing cost and the candidate test cost, both of which are dependent on K. For example, in the situation where the data-points are evenly distributed into their own hashtable bucket ($N = 2^K$), the total computational cost for LSH is actually sub-linear ($\mathcal{O}(D \log N)$).

5. We define a projected dimension as the collection of the real-valued projections (dot products) of all data-points onto the normal vector to a hyperplane.

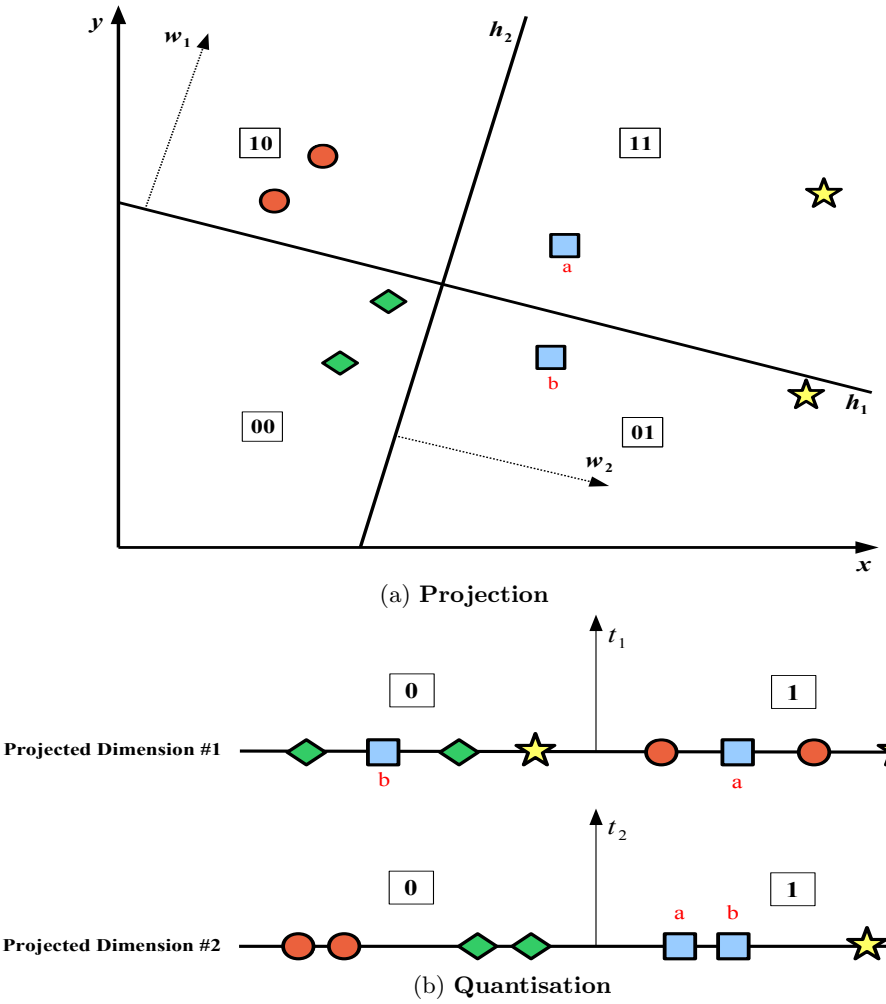


Figure 3: The projection and quantisation operations. In Figure (a) a 2D space is partitioned with two hyperplanes $\mathbf{h}_1$ and $\mathbf{h}_2$ with normal vectors $\mathbf{w}_1, \mathbf{w}_2$ creating four buckets. Data-points are shown as coloured shapes, with similar data-points having the same colour and shape. The hashcode for each data-point is found by taking the dot-product of the feature representation onto the normal vectors ($\mathbf{w}_1$, $\mathbf{w}_2$) of each hyperplane. The resulting projected dimensions are binarised by thresholding at zero (Figure (b)) with two thresholds t_1, t_2. Concatenating the resulting bits yields a 2-bit hashcode for each data-point (indicated by the unfilled squares). For example the projection of data-point a is greater than threshold t_1 and so a '1' is appended to its hashcode. Data-point a's projection onto normal vector $\mathbf{w}_2$ is also greater than t_2 and so a '1' is further appended to its hashcode. The hashcode for data-point a is therefore '11' which is also the label for the top-right region of the feature space in Figure (a).

memory requirements needed for an LSH deployment. The quantisation thresholds are also set in a data-independent manner, typically by thresholding at zero along a projected dimension. In this context a projected dimension is formed from collating the projections from

all data-points onto the normal vector to a hyperplane. Unfortunately, the region around zero on a projected dimension is usually the area of highest point density which means that there is a high chance of related data-points falling on opposite sides of the threshold and therefore being assigned different bits. There is clearly a wide scope for improving the retrieval effectiveness of LSH and many other influential but data-oblivious algorithms for hashing-based approximate nearest neighbour search by tackling both of these issues. This survey is dedicated to recent hashing models that overcome these shortcomings, with a particular focus on Computer Vision applications.

This literature survey is structured as follows: the chapter begins in Section 2 with an overview of the preliminaries and notation definition. In Section 3 we provide 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 3 as to why a relaxed version of the problem is required, known commonly as approximate nearest neighbour (ANN) search. In Section 4, we describe a seminal method, Locality Sensitive Hashing (LSH), for solving the ANN search problem in a time constant in the number of data-points. This is a self-contained introduction to LSH sufficient for researchers to understand the important aspects of the algorithm and how it can be applied in practice. The limitations of LSH are discussed and we 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. We divide this latter part of the review into methods for binary quantisation (Section 5) and projection function learning (Section 6), mirroring the two common stages of hashcode generation.

2. Preliminaries and Notation Definition

This survey 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 $\mathbf{X} \in \mathbb{R}^{N \times D}$. We 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 $d(.,.)$. We summarise the notation used throughout this review in Table 1.

Notation	Definition
N	Number of data-points in dataset
D	Dimensionality of data-point feature representation
K	Number of hashcode bits
L	Number of hashtables
Q	Number of query data-points
B	Number bits per projected dimension
T	Number of thresholds per projected dimension
M	Iterations
C	Randomly sampled data-points *or* cluster centroids

$\mathbf{X} \in \mathbb{R}^{N \times D}$	Dataset of N data-points, dimensionality D
$\mathbf{x}_r \in \mathbb{R}^D : \mathbf{x}_r = \mathbf{X}_{r\bullet}$	r^{th} row of matrix $\mathbf{X}$
$\mathbf{x}^c \in \mathbb{R}^N : \mathbf{x}^c = \mathbf{X}_{\bullet c}$	c^{th} column of matrix $\mathbf{X}$
X_{ij}	Element of matrix $\mathbf{X}$ in row i column j
$\mathbf{q} \in \mathbb{R}^D$	Query data-point
$\mathbf{p} \in \mathbb{R}^D$	Arbitrary database data-point
$\mathbf{Y} \in \mathbb{R}^{N \times K}$	Projection matrix of N data-points, dimensionality K
$\mathbf{y}_r \in \mathbb{R}^K : \mathbf{y}_r = \mathbf{Y}_{r\bullet}$	Projected values for r^{th} data-point
$\mathbf{y}^c \in \mathbb{R}^N : \mathbf{y}^c = \mathbf{Y}_{\bullet c}$	c^{th} projected dimension
$d : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$	Distance function e.g. Euclidean distance
$q_k : \mathbb{R} \to \{0,1\}^B$	Quantisation function
$\mathbf{D} \in \mathbb{R}^{N \times N}$	Matrix of data-point distances
$\mathbf{B} \in \{-1,1\}^{N \times K}$	Hashcodes of N data-points each of length K bits
$\mathbf{b}_r \in \{-1,1\}^K : \mathbf{b}_r = \mathbf{B}_{r\bullet}$	Hashcode of r^{th} data-point $\mathbf{x}_r$
$h_k : \mathbb{R}^D \to \{0,1\}$	Hash function
$g_l : \mathbb{R}^D \to \{0,1\}^K$	Hash function concatenation $[h_1(.), h_2(.), \ldots, h_K(.)]$
$\mathbf{S} \in \mathbb{R}^{N \times N}$	$S_{ij} = 1$ if $\mathbf{x}_i$ and $\mathbf{x}_j$ are nearest neighbours, 0 otherwise
$\mathbf{h}_k \in \mathbb{R}^D$	Hyperplane
$\mathbf{w}_k \in \mathbb{R}^D$	Hyperplane normal vector
$\mathbf{W} \in \mathbb{R}^{D \times K}$	Matrix of K hyperplane normal vectors
$t_k \in \mathbb{R}$	Scalar threshold
$\mathbf{T} \in \mathbb{R}^{K \times T}$	Matrix of thresholds for each projected dimension
$\mathbf{t}_r \in \mathbb{R}^T : \mathbf{t}_r = \mathbf{T}_{r\bullet}$	Set of thresholds for r^{th} projected dimension
$\kappa : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$	Kernel function
$\gamma \in \mathbb{R}$	Kernel bandwidth parameter
$\|\mathbf{X}\|_F^2 = \sum_{ij}^N \|X_{ij}\|^2$	Frobenius L_2 norm of matrix
$\|\mathbf{X}\|_F^1 = \sum_{ij}^N \|X_{ij}\|$	Frobenius L_1 norm of matrix
$\mathbf{X} = diag(\mathbf{x})$	Places elements of vector $\mathbf{x}$ on diagonal of matrix $\mathbf{X}$
$sgn(a) \in \{-1,1\}$	Sign function returning 1 for $a > 0$, and -1 otherwise
$[a]_+$	Equal to a if $a \geq 0$, and 0 otherwise

Table 1: Definition of the mathematical notation used throughout the review.

The research described in this survey 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 $\left\{h_k : \mathbb{R}^D \to \{0,1\}\right\}_{k=1}^K$ 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 $\left\{g_l : \mathbb{R}^D \to \{0,1\}^K\right\}$ 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. We dedicate the remainder of this chapter to describing how similarity preserving hash

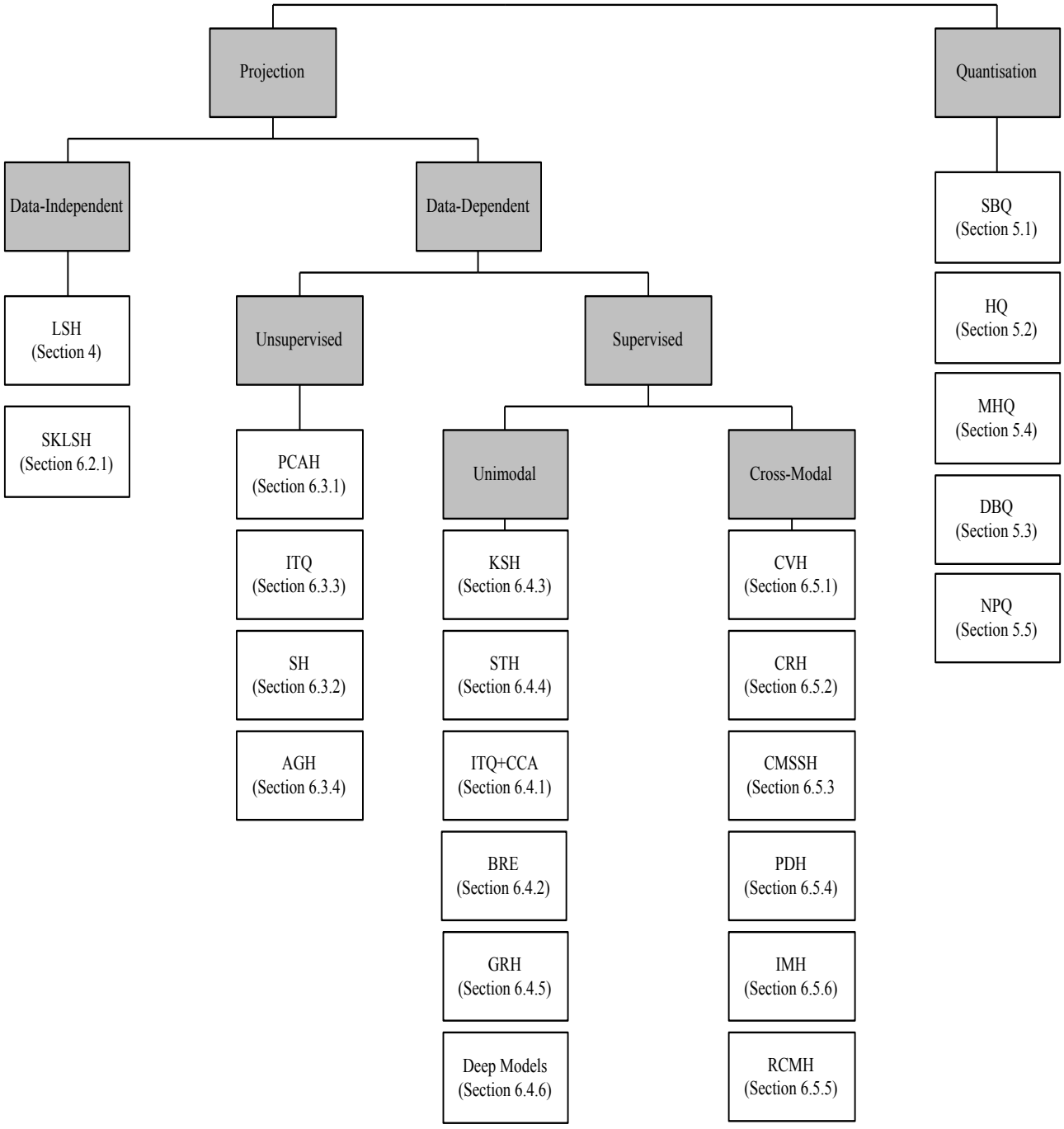


Figure 4: Overview of one possible categorisation of the field of hashing-based ANN search. The main categories are shown in the grey boxes while the actual models themselves are highlighted in white alongside their relevant section number.

functions are constructed by relevant models from the literature. A birds-eye-view of the structure of this survey is shown in Figure 4.

3. Approximate Nearest Neighbour (ANN) Search

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

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 $\big\{d(.,.) : \mathbb{R}^D \times \mathbb{R}^D \rightarrow [0,1]\big\}$. This variant of the problem is also known as 1-NN search and is specified mathematically in Equation 1

$$NN(\mathbf{q}) = argmin_{\mathbf{x}_i \in \mathbf{X}} d(\mathbf{x}_i, \mathbf{q}) \tag{1}$$

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 & Moran, 2013), (Moran & 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)

$$\begin{aligned} d_{pnorm}(\mathbf{x}_i, \mathbf{x}_j) = &\quad \|\mathbf{x}_i - \mathbf{x}_j\|_\rho \\ = &\quad \left(\sum_{k=1}^{D} |x_{ik} - x_{jk}|^\rho \right)^{\frac{1}{\rho}} \end{aligned} \tag{2}$$

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 3 is another popular distance metric for NN search that has proven particularly effective for document retrieval (Manning, Raghavan, & Schütze, 2008), (Ravichandran, Pantel, & Hovy, 2005)

$$d_{cosine}(\mathbf{x}_i, \mathbf{x}_j) = 1 - \frac{\sum_{k=1}^{D} x_{ik} x_{jk}}{\sqrt{\sum_{k=1}^{D} x_{ik}^2} \sqrt{\sum_{k=1}^{D} x_{jk}^2}} \tag{3}$$

We will also come across the Hamming distance extensively in this review as it is the de-facto metric for comparing binary strings (Equation 4)

$$d_{hamming}(\mathbf{b}_i, \mathbf{b}_j) = \sum_{k=1}^{D} \delta[b_{ik} \neq b_{jk}] \tag{4}$$

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 & 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). We pick up this thread again in Section 6 where we discuss 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 $\mathcal{O}(ND)$ 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 review in Section 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, 1975), quad-tree (Finkel & Bentley, 1974), X-tree (Berchtold, Böhm, Braunmüller, Keim, & Kriegel, 1997) and SR-tree (Katayama & 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, Schek, & Blott, 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 & 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 & Darrell, 2007), retrieving sub-optimal nearest neighbours

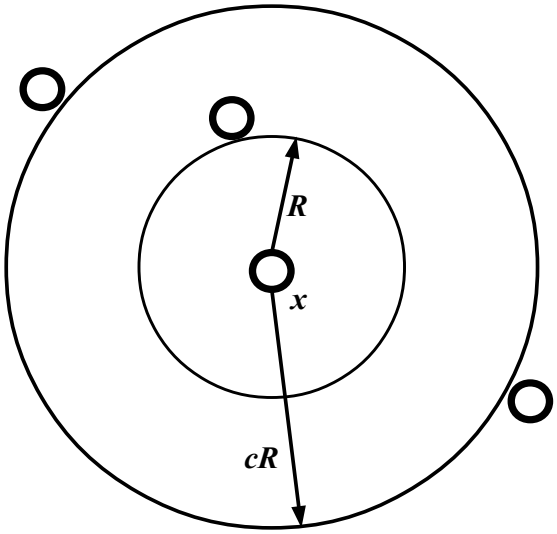


Figure 5: The (c, R)-approximate NN problem: in many applications it is acceptable to return a data-point (indicated with the circles) within a distance cR of the query point $\mathbf{x}$, where R is the distance to the exact NN and the approximation factor $c > 1$.

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 5). The (c, R)-approximate NN decision problem is formalised in (Andoni & Indyk, 2008)(Petrovic, 2012) and defined in Definition 3.1:

Definition 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 & Indyk, 2008), (Petrovic, 2012) is similar but without the approximation factor c (Definition 3.2)

Definition 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 4, we 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.

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 we introduce the core ideas behind Locality Sensitive Hashing (LSH) (Indyk & 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, Viola, & Darrell, 2003), bag-of-words indexing (Chum, Philbin, & Zisserman, 2008) and shape matching (Grauman & Darrell, 2004). The hashing models we introduce later in this survey (Sections 5-6) can all be thought of as extensions of LSH that try and overcome certain disadvantages with the original algorithm. We will begin by giving a general overview of LSH, independent of the similarity metric of interest. In Section 4.1 we will then discuss a concrete instantiation of LSH for the inner product similarity.

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 4.1)

Definition 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 & 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, Immorlica, Indyk, & Mirrokni, 2004), cosine distance (inner product similarity) (Charikar, 2002), Jaccard distance (Broder, 1997) and L_2-norm on the unit hypersphere (Terasawa & 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, Moran, McCreadie, von Lünen, Sykora, Cano, Ireson, Macdonald, Ounis, He, Jackson, Ciravegna, & O'Brien, 2014). We will expand on this particular hash function family in more detail in Section 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, Darrell, & Indyk, 2006)(Matei, Shan, Sawhney, Tan, Kumar, Huber, & Hebert, 2006). This hash function family relies on the *Johnson-Lindenstrauss* lemma (Johnson & 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 5

$$g_l(\mathbf{q}) = [h_1(\mathbf{q}), h_2(\mathbf{q}), \ldots, h_K(\mathbf{q})] \tag{5}$$

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 & 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 & 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 6-7). 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 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[6] 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, Condie, & Ganesan, 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

6. `http://www.mit.edu/~andoni/LSH/`

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
```

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
```

in Algorithm 1 while the querying process is presented in Algorithm 2. The presentation of the pre-processing and 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 3.1-3.2. The strategy presented in Algorithm 2 solves the R-near neighbour reporting problem (Definition 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 3.1) is solved by stopping the search after the first $L^{'} = 3L$ data-points have been retrieved. This strategy comes with an $\mathcal{O}(L)$ bound on the query time.

4.1 LSH with Sign Random Projections

In this survey we 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 6.

$$\begin{aligned} d(\mathbf{p}, \mathbf{q}) &= \sum_{k=1}^{D} p_k q_k \\ &= \mathbf{p}^T \mathbf{q} \end{aligned} \tag{6}$$

Equation 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 & 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 7

$$\begin{aligned} h_k(\mathbf{q}) &= \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{q})) \\ &= q_k(p_k(\mathbf{q})) \end{aligned} \tag{7}$$

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 $\mathbf{h}_k$. We 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 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 7, (Goemans & Williamson, 1995) showed that the probability of a collision is given as in Equation 8.

$$Pr_{\mathcal{H}_{cosine}}(h(\mathbf{p}) = h(\mathbf{q})) = 1 - \frac{\theta(\mathbf{p}, \mathbf{q})}{\pi} \tag{8}$$

Equation 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 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 we discussed in Section 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

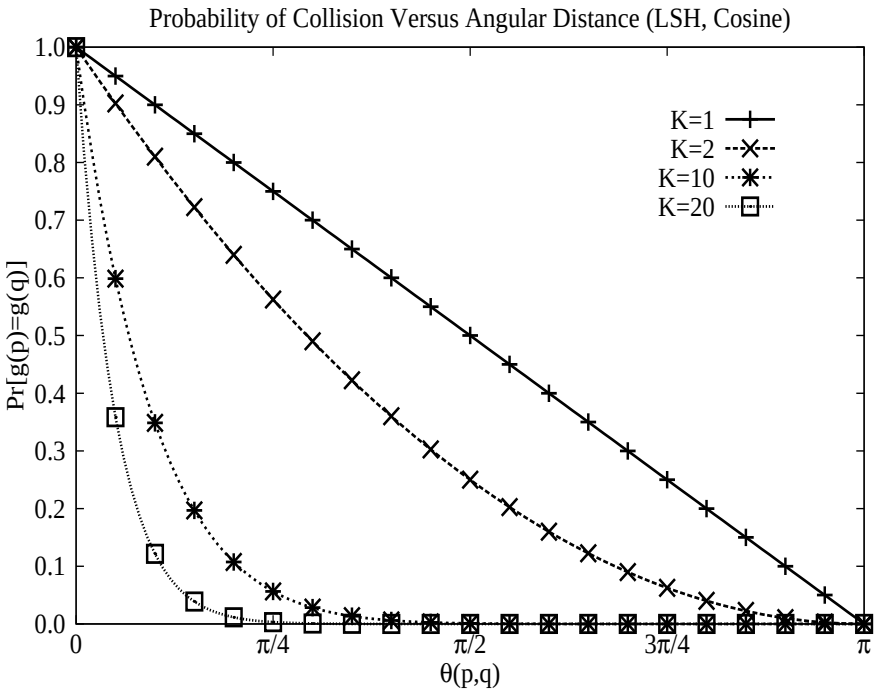


Figure 6: Visualising the probability of two hashcodes $g(\mathbf{p}), g(\mathbf{q})$ matching against the length of the hashcode key ($K = 1, 2, 10, 20$). The hash function family is $\mathcal{H}_{cosine}$. As the hashcode length becomes longer the two data-points must be close together (in terms of 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).

L is crucial for retrieval effectiveness and efficiency in an end-application. We illustrate the locality sensitive nature of $\mathcal{H}_{cosine}$ in Figures 6-7. In these figures we change the value of K and L and observe the effect on the probability of a collision occurring in the hashtables.

The query time complexity is also dependent on L and K. For a single query data-point the retrieval cost can be characterised by $\mathcal{O}(KDL) + \mathcal{O}(1) + \mathcal{O}(\frac{NDL}{2^K})$. This is made up of the *hashing time* (time spent generating the hashcodes) $\mathcal{O}(KDL)$, the *lookup time* ($\mathcal{O}(1)$ for a good hashtable implementation) and the *candidate test time* ($\mathcal{O}(\frac{NDL}{2^K})$), 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 7 provides the foundation upon which the rest of this review is formed. The models we discuss all propose novel formulations for defining the projection function $\left\{p_k : \mathbb{R}^D \rightarrow \mathbb{R}\right\}$ 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.

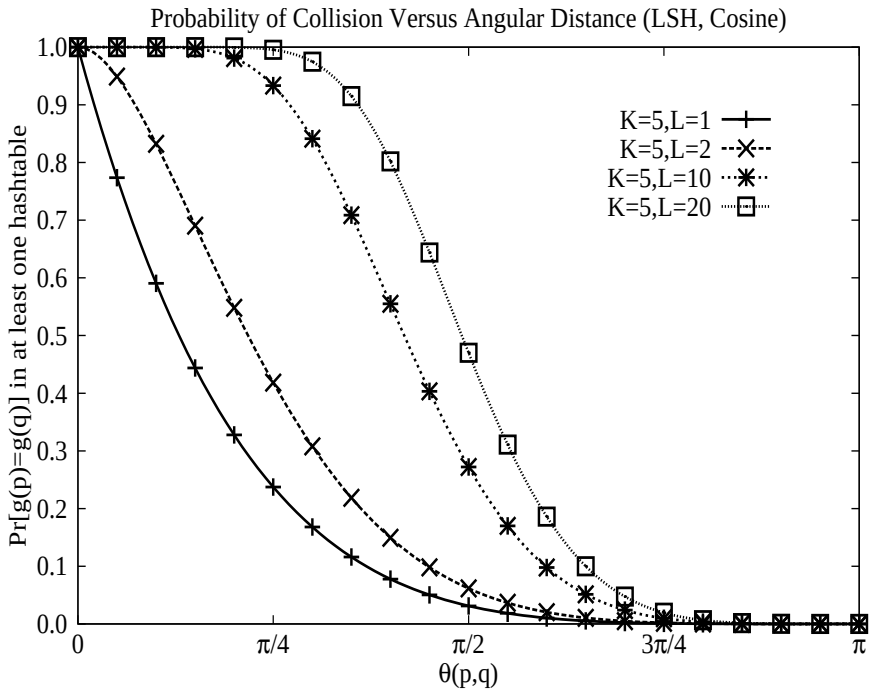
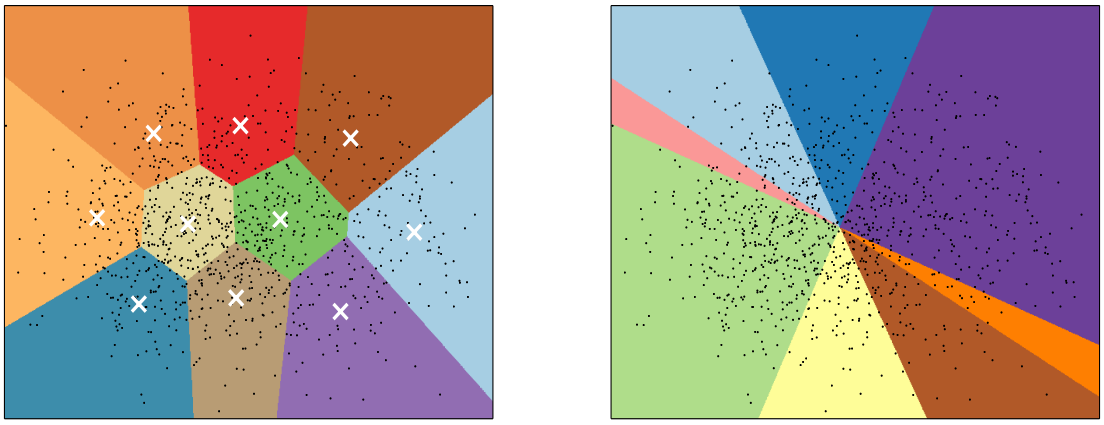


Figure 7: Probability of collision resulting from varying the number of hashtables ($L = 1, 2, 10, 20$) for fixed hashcode length $K = 5$. The hash function family is $\mathcal{H}_{cosine}$. We can see that as the number of hashtables increase there is a higher probability of two data-points being close 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).

More specifically, the literature challenges the notion that a sign function is an optimal quantisation strategy for converting the real-valued projection in Equation 7 to binary (Section 5 and secondly, that randomly sampled hyperplanes produce optimal hashcodes (Section 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, Kumar, & Chang, 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. We discuss state-of-the-art data-driven hash functions in Sections 5-6.

5. Quantisation for Nearest Neighbour Search

In this section we review previous related research in the field of binary quantisation for hashing-based ANN search. We saw in Section 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 we study in depth recently proposed algorithms for reducing the



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

Figure 8: 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.

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 7. 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 & 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 product of the feature vector of each data-point onto the normal vectors to a set of random hyperplanes partitioning the feature space (Figure 8). As we will discover in Section 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

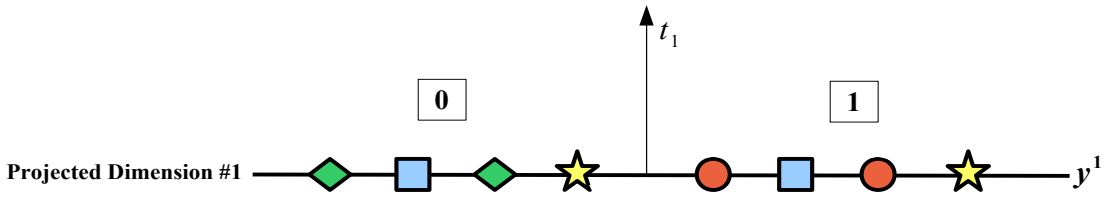


Figure 9: Single bit quantisation (SBQ) uses one threshold $t_1 \in \mathbb{R}$ to binarise a projected dimension: projected values (indicated by the coloured shapes) lower than the threshold (on the left) are assigned a '0' bit, while projected values greater than the threshold (on the right) are assigned a '1' bit.

vectors (centroids). K-means divides the space into Voronoi regions forming a more flexible data-driven partitioning. We illustrate the partitions formed by vector quantisation and a hyperplane based scalar quantisation in Figure 8. We will not cover vector quantisation any further here, but we 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 read off inter-cluster Euclidean distances using the centroid indices[7]. This computation has been found to be 10-20 times slower than the Hamming distance computation between the binary hashcodes on standard datasets (He, Wen, & Sun, 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 5.1) or multi-bit (Sections 5.3-5.4) 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 review we follow (Kong, Li, & Guo, 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 9 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 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

7. 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	$\mathcal{O}(1)$	5.1
HQ	00/01/10/11	Spectral partitioning	1 and 2	$\mathcal{O}(CN^{+}_{trd})$	5.2
DBQ	00/10/11	Squared error	2	$\mathcal{O}(N_{trd} \log N_{trd})$	5.3
MHQ	NBC	1D K-means	3+	$\mathcal{O}(2^{B} N_{trd})$	5.4
NPQ	Any	Semi-supervised	1+	$\mathcal{O}(N^{2}_{trd} TF)$	5.5

Table 2: 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 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, B is the number of bits per projected dimension, T is the number of thresholds, F is the number of objective function evaluations. Time complexity is for positioning thresholds along a single projected dimension.

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 we 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 $\left\{\mathbf{r}_i \subset \mathbf{y}^k\right\}_{i=1}^{T+1}$ 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, Lavrenko, & Osborne, 2013a)(Kong et al., 2012) (Kong & 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 we 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.

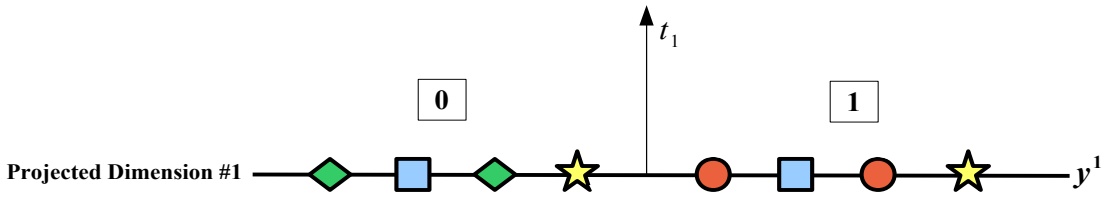


Figure 10: Single Bit Quantisation (SBQ) uses one threshold t_1 to binarise a projected dimension: projected values (indicated by the coloured shapes) lower than the threshold (on the left) are assigned a 0 bit, while projected values greater than the threshold (on the right) are assigned a 1 bit.

The previous paragraph hints at three key properties that can be used to distinguish and categorise existing methods of scalar quantisation[8]: 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 we provide an overview of existing quantisation methods as categorised along these three dimensions of variability. In the following sections we describe these existing quantisation schemes in detail. Specifically, in Section 5.1 we introduce the traditional method of Single Bit Quantisation (SBQ) and in Sections 5.2-5.4 we describe the more recent multi-threshold quantisation schemes: Hierarchical Quantisation (HQ), Double Bit Quantisation (DBQ) (Section 5.3), Manhattan Hashing Quantisation (MHQ) (Section 5.4) and Neighbourhood Preserving Quantisation (NPQ) (Section 5.5).

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 9.

$$h_k(\mathbf{x}_i) = \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k)) \tag{9}$$

In this quantisation scheme each hyperplane contributes one bit towards the hashcode 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 $\mathcal{O}(1)$ time[9] for

8. We will use the term *quantisation* to refer to scalar quantisation throughout the remainder of this survey.
9. This increases to $\mathcal{O}(N_{trd})$ time for threshold learning if the median is used as the threshold.

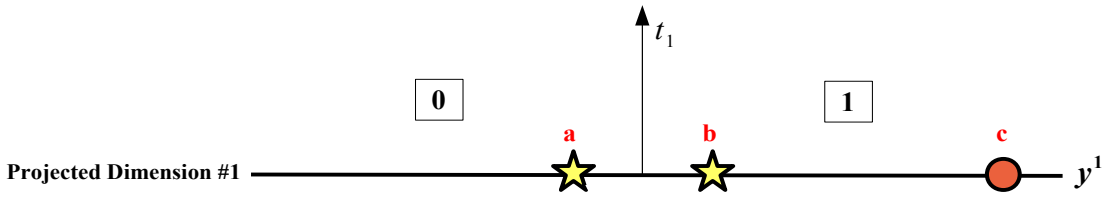


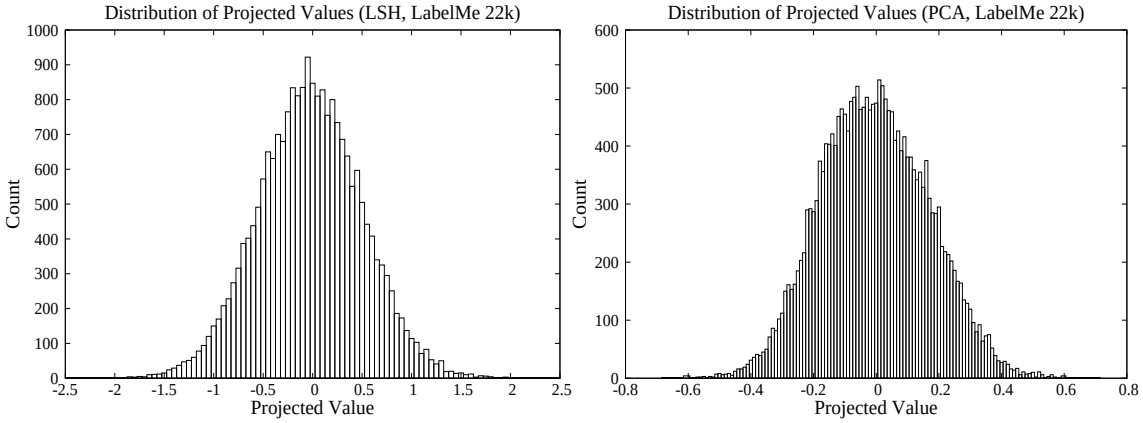
Figure 11: The problem with Single Bit Quantisation (SBQ): true nearest neighbours such points a, b are assigned different bits despite being close together along the projected dimension. Conversely points b, c located above the threshold, which are non-nearest neighbours, are assigned the same bit (1), even though they are more distantly spaced along the projected dimension.

threshold learning and $\mathcal{O}(1)$ time for encoding a novel query data-point. SBQ is further illustrated with a toy example in Figure 10.

The multi-threshold quantisation algorithms we describe in Section 5.2-5.4 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 & Li, 2012a; Moran et al., 2013a; Moran, Lavrenko, & Osborne, 2013b). 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. This problem with SBQ is easily illustrated by considering a hypothetical true nearest neighbour data-point pair in Figure 11. 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 Hamming 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[10]. 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 12). Partitioning a projected

10. 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. We cover the latter data-dependent methods in Section 6.



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

Figure 12: 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 & 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 & 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 5.3.

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 is maximal preservation of the neighbourhood structure encoded in the projected dimension through a multi-bit codebook and a threshold optimisation algorithm. We will now discuss one of the first proposed multi-threshold quantisation schemes in Section 5.2.

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. We describe the projection learning component of AGH in detail in Section 6.3.4, while in this section we 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 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 13. (Liu, Wang, Kumar, & Chang, 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 $\hat{\mathbf{S}}$[11]. 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 10):

$$Z_{ij} = \begin{cases} \dfrac{exp(-d^2(\mathbf{x}_j, \mathbf{c}_i)/\gamma)}{\sum\limits_{i' \in \langle j \rangle} exp(-d^2(\mathbf{x}_j, \mathbf{c}_{i'}))/\gamma)} & \text{if } i \in \langle j \rangle \\ 0 & otherwise \end{cases} \tag{10}$$

where γ is the kernel bandwidth, $\left\{d(.,.) : \mathbb{R}^D \times \mathbb{R}^D \rightarrow [0,1]\right\}$ 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 $\left\{\mathbf{c}_i \in \mathbb{R}^D\right\}_{i=1}^{C}$ 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 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 11).

$$h_k(\mathbf{x}_i) = \begin{cases} \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{x}_i - t_2)), & \text{if } \mathbf{w}_k^\intercal \mathbf{x}_i \geq 0 \\ \frac{1}{2}(1 + sgn(-\mathbf{w}_k^\intercal \mathbf{x}_i + t_3)), & \text{if } \mathbf{w}_k^\intercal \mathbf{x}_i < 0 \end{cases} \tag{11}$$

Binarising a graph Laplacian eigenvector has the effect of partitioning the graph encoded by $\hat{\mathbf{S}}$ into two groups, with each of the K eigenvectors forming a different bi-partitioning of the graph (Shi & 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 & 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

11. The rank of a matrix is the number of linearly independent rows or columns.

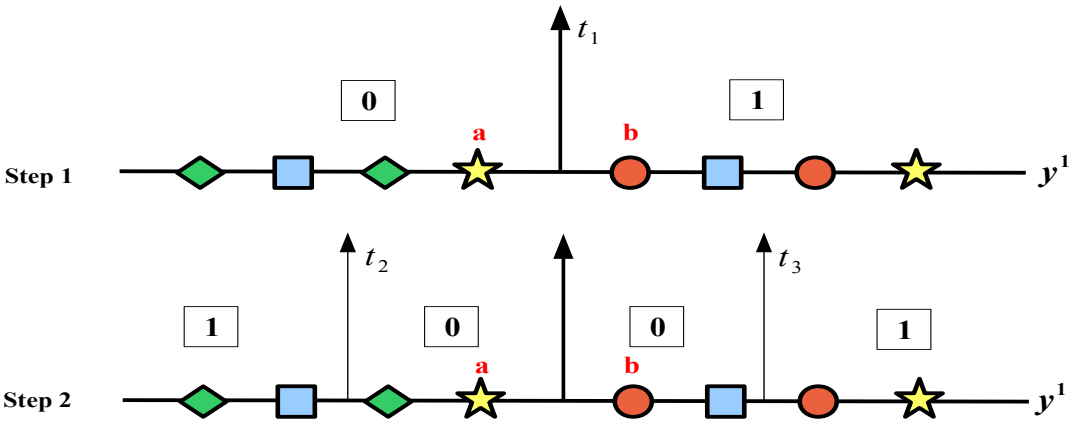


Figure 13: Illustration of the Hierarchical Quantisation (HQ) algorithm of (Liu et al., 2011). We use the data-points a (yellow star) and b (red circle) as examples. The quantisation proceeds in two steps: in the first step SBQ thresholds the projected dimension into two regions generating the first 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.

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 12

$$\begin{aligned} \text{argmin}_{t_2,t_3} \quad & \mathbf{f}^\intercal \mathbf{L} \mathbf{f} \\ \text{where } \mathbf{f} = & \begin{bmatrix} \mathbf{y}^+ - t_2 \mathbf{1}_1 \\ -\mathbf{y}^- + t_3 \mathbf{1}_2 \end{bmatrix} \\ \text{subject to } & \mathbf{1}^\intercal \mathbf{f} = 0 \end{aligned} \tag{12}$$

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, Torralba, & Fergus, 2008). Liu et al., 2011 demonstrate that by setting to zero the derivatives of Equation 12, the two thresholds t_2, t_3 minimising Equation 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 $\mathcal{O}(1)$. 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. We will examine subsequent research contributions in this area in chronological order continuing next to the Double Bit Quantisation (DBQ) algorithm of Kong & Li (Kong & Li, 2012a).

5.3 Double Bit Quantisation (DBQ)

Double-Bit Quantisation (DBQ) (Kong & Li, 2012a) allocates two thresholds per projected dimension and assigns two bits to the three resulting thresholded regions. Unlike HQ (Section 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 14 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 13)

$$\mathcal{J}_{dbq}(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) = \sum_{y_i \in \mathbf{r}_1} (y_i - \mu_1)^2 + \sum_{y_j \in \mathbf{r}_2} (y_j - \mu_2)^2 + \sum_{y_l \in \mathbf{r}_3} (y_l - \mu_3)^2 \tag{13}$$

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 12), 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 16 (Kong & Li, 2012a)

$$\begin{aligned}
\mathcal{J}_{dbq}(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) &= \sum_{y_i \in \mathbf{y}_k} y_i^2 - 2 \sum_{y_i \in \mathbf{r}_1} y\mu_1 + \sum_{y_i \in \mathbf{r}_1} \mu_1^2 - 2 \sum_{y_l \in \mathbf{r}_3} y\mu_3 + \sum_{y_l \in \mathbf{r}_3} \mu_3^2, && (14) \\
&= \sum_{y_i \in \mathbf{y}_k} y_i^2 - |\mathbf{r}_1|\mu_1^2 - |\mathbf{r}_3|\mu_3^2, && (15) \\
&= \sum_{y_i \in \mathbf{y}_k} y_i^2 - \frac{(\sum_{y_i \in \mathbf{r}_1} y_i)^2}{|\mathbf{r}_1|} - \frac{(\sum_{y_i \in \mathbf{r}_3} y_i)^2}{|\mathbf{r}_3|} && (16)
\end{aligned}$$

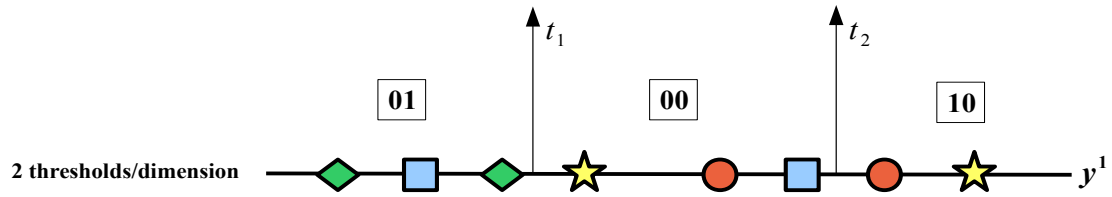


Figure 14: Double Bit Quantisation allocates two thresholds t_1, t_2 to binarise a projected dimension. The resulting three thresholded regions are assigned the two-bit encoding shown. This encoding ensures that adjacent regions are only separated by a Hamming distance of 1.

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 17).

$$\mathcal{J}'_{dbq}(\mathbf{r}_1, \mathbf{r}_3) = \frac{(\sum_{y_i \in \mathbf{r}_1} y_i)^2}{|\mathbf{r}_1|} + \frac{(\sum_{y_j \in \mathbf{r}_3} y_j)^2}{|\mathbf{r}_3|} \tag{17}$$

The objective function $\mathcal{J}'_{dbq}$ 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 $\mathcal{J}'_{dbq}$ 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 $\mathcal{J}'_{dbq}$ is then computed (Line 15) on the new regions $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3$. If there is an increase in $\mathcal{J}'_{dbq}$ 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 $\mathcal{J}'_{dbq}$.

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 assigned similar hashcodes (given that they will end up in the same thresholded region) versus an entirely random threshold setting. While Kong & Li, 2012a demonstrate that this is a reasonable assumption in practice, Moran et al. (Moran et al., 2013a) demonstrated 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

Algorithm 3: Double Bit Quantisation (Kong & 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 $t_1 \in \mathbb{R}$, $t_2 \in \mathbb{R}$

1 $\mathbf{r}_1 \leftarrow \{y_i | y_i \leq 0, y_i \in \mathbf{y}^k\}$
2 $\mathbf{r}_2 \leftarrow \emptyset$
3 $\mathbf{r}_3 \leftarrow \{y_i | y_i > 0, y_i \in \mathbf{y}^k\}$
4 Sort (ascending) projected-values in $\mathbf{r}_1$
5 Sort (ascending) projected-values in $\mathbf{r}_3$
6 $J_{max} \leftarrow 0$
7 $i \leftarrow 1$
8 **while** $\mathbf{r}_1 \neq \emptyset$ *or* $\mathbf{r}_3 \neq \emptyset$ **do**
9 **if** *(sum($\mathbf{r}_2$) ≤ 0)* **then**
10 $\mathbf{r}_2 \leftarrow \mathbf{r}_2 \cup min(\mathbf{r}_3)$ `// Remove minimum value in` $\mathbf{r}_3$
11 **else**
12 $\mathbf{r}_2 \leftarrow \mathbf{r}_2 \cup max(\mathbf{r}_1)$ `// Remove maximum value in` $\mathbf{r}_1$
13 **end**
14 $i \leftarrow i + 1$
15 $J \leftarrow \mathcal{J}'_{dbq}(\mathbf{r}_1, \mathbf{r}_3)$ `// Equation 17`
16 **if** $(J > J_{max})$ **then**
17 $t_1 \leftarrow max(\mathbf{r}_1)$
18 $t_2 \leftarrow max(\mathbf{r}_2)$
19 $J_{max} \leftarrow J$
20 **end**
21 **end**
22 **return** t_1, t_2

produces the projections (Section 5.5). The threshold learning time complexity of DBQ is $\mathcal{O}(N_{trd} \log N_{trd})$ which arises from the pre-processing step that sorts the projected values in regions $\mathbf{r}_1$ and $\mathbf{r}_3$ (Lines 4 and 5). DBQ has $\mathcal{O}(1)$ time complexity when using the learnt thresholds to generate a bit for a novel query data-point.

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. 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. We describe both contributions in this section.

The binary encoding scheme advocated by MHQ is illustrated in Figure 15. 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 15 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 15). 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 effectiveness. 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[12]. To illustrate how this method works we 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 18

$$\begin{aligned} d_{MHQ}(000100, 110000) &= d_M(00, 11) + d_M(01, 00) + d_M(00, 00) \\ &= 3 + 1 + 0 \\ &= 4 \end{aligned} \tag{18}$$

If the number of bits per dimension $B = 3$ then the computation proceeds as in Equation 19.

$$\begin{aligned} d_{MHQ}(000100, 110000) &= d_M(000, 110) + d_M(100, 000) \\ &= 6 + 4 \\ &= 10 \end{aligned} \tag{19}$$

12. We 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.

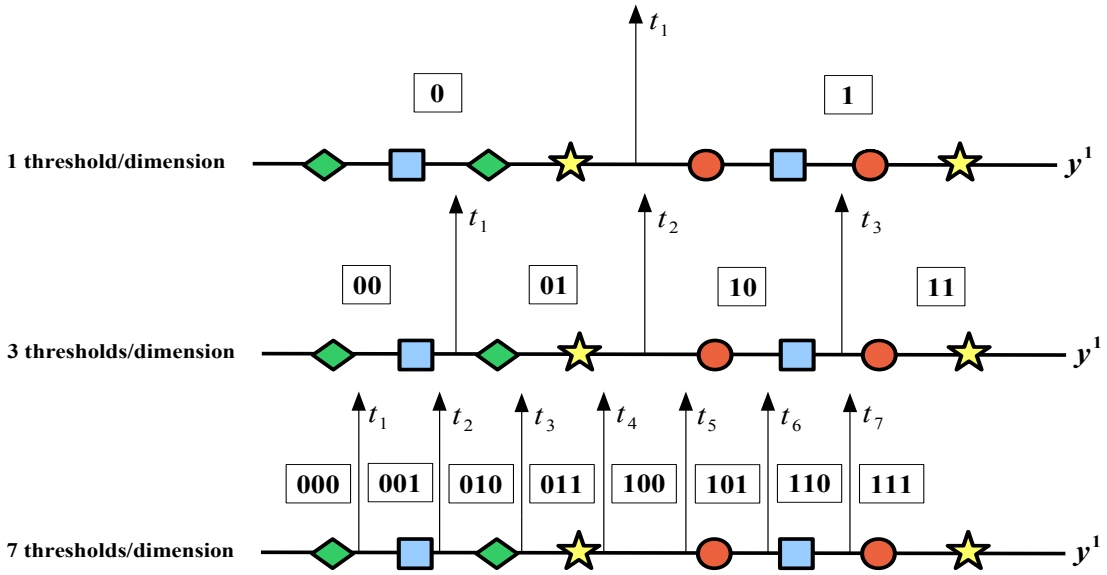


Figure 15: Manhattan Hashing Quantisation (MHQ) assigns $T = 2^B - 1$ thresholds per dimension, where B is the number of bits allocated per dimension. The encoding scheme for the thresholded regions is natural binary code (NBC). Each region from the left to the right is assigned an integer starting at 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.

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, Duan, Lin, Wang, Huang, & Gao, 2015b) by showing that the Manhattan distance requires substantially more atomic operations on the CPU than the Hamming distance.

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 20).

$$t_i = \frac{(c_i + c_{i+1})}{2} \tag{20}$$

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

5.5 Neighbourhood Preserving Quantisation (NPQ)

In contrast to previously discussed quantisation models such as AGH (Liu et al., 2011), DBQ (Kong et al., 2012) and MHQ (Kong & Li, 2012a), Moran et al. (Moran et al., 2013a) propose a quantisation algorithm, *Neighbourhood Preserving Quantisation (NPQ)*, that leverages a binary adjacency matrix $\boldsymbol{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$, where N_{trd} is the number of training data-points ($N_{trd} \ll N$), to guide the threshold positioning. Their unique hypothesis is that the neighbourhood structure between the data-points in the input feature space is a valuable signal for guiding the quantisation thresholds within the lower-dimensional projected space. The adjacency matrix $\mathbf{S}$ therefore encodes the neighbourhood structure of the data-points in the original feature space, where $S_{ij} = 1$ if points $\mathbf{x}_i$ and $\mathbf{x}_j$ are considered neighbours (a *positive* pair), and $S_{ij} = 0$ otherwise (a *negative* pair). $\mathbf{S}$ can be generated, for example, by computing Euclidean distance between N_{trd} data-points and setting any data-points within an ϵ-ball of each other as true nearest neighbours[13]. The pairwise affinity matrix $\mathbf{S}$ specifies the pairs of points that should fall within the same thresholded regions and therefore be assigned identical hashcodes from the codebook.

Moran et al. (Moran et al., 2013a) define an objective function for threshold positioning that directly leverages the neighbourhood structure encoded in $\mathbf{S}$. For a fixed set of thresholds $\mathbf{t}_k = [t_{k1} \dots t_{kT}]$ they define a per-projected dimension indicator matrix $\mathbf{P}^k \in \{0,1\}^{N_{trd} \times N_{trd}}$ with the property given in Equation 21:

$$P^k_{ij} = \begin{cases} 1, & \text{if} \quad \exists_\gamma \quad s.t. \quad t_{k\gamma} \leq (y^k_i, y^k_j) < t_{k(\gamma+1)} \\ 0, & \text{otherwise.} \end{cases} \tag{21}$$

The index $\gamma \in \mathbb{Z}$ spans the range: $0 \leq \gamma \leq T$, where the scalar quantity T denotes the total number of thresholds partitioning a given projected dimension. Intuitively, matrix $\mathbf{P}^k$ indicates whether or not the projections (y^k_i, y^k_j) of any pair of data-points $(\mathbf{x}_i, \mathbf{x}_j)$ fall within the same thresholded region of the one-dimensional projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$. Given a particular instantiation of the thresholds $[t_{k1} \dots t_{kT}]$, the algorithm counts the number of *true positives* (TP), *false negatives* (FN) and *false positives* (FP) across all regions. The requisite TP, FP and FN counts can then be stated as in Equations 22-24

$$TP = \frac{1}{2} \sum_{ij} P_{ij} S_{ij} = \frac{1}{2} \|\mathbf{P} \circ \mathbf{S}\|_1 \tag{22}$$

$$FN = \frac{1}{2} \sum_{ij} S_{ij} - TP = \frac{1}{2} \|\mathbf{S}\|_1 - TP \tag{23}$$

13. A fuller definition of ϵ-NNs can be found in Section 7.3.1

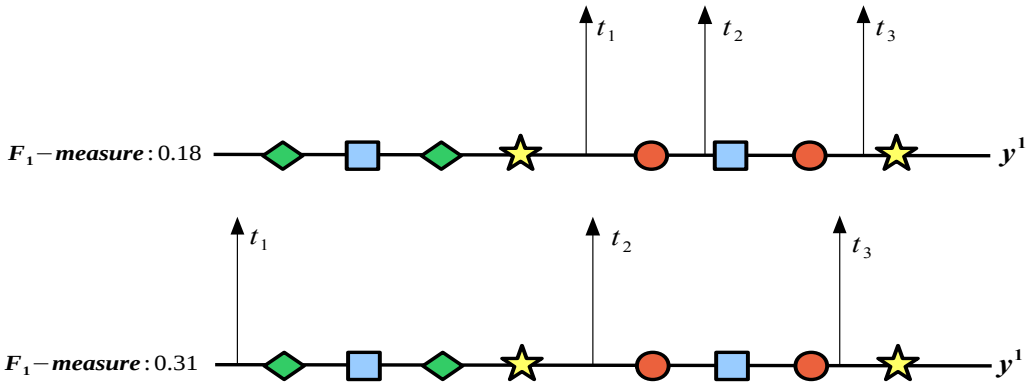


Figure 16: Maximisation of the F_1-measure can lead to an effective setting of the quantisation thresholds. In both diagrams we seek to position three thresholds along the same projected dimension. In the top diagram the threshold positioning leads to an F_1-measure of 0.18. This is a rather low score which results from separating many of the true NNs (indicated with the same colour and shape) in different regions. The threshold positions in the lower diagram lead to a higher F_1-measure, approximately twice as high, which arises from capturing more true nearest neighbours in the same thresholded regions.

$$FP = \frac{1}{2}\sum_{ij} P_{ij} - TP = \frac{1}{2}\|\mathbf{P}\|_1 - TP \tag{24}$$

where $\circ$ denotes the Hadamard (elementwise) product and $\|.\|_1$ is the L_1 matrix norm defined as $\|\mathbf{X}\|_1 = \sum_{ij} |X_{ij}|$. Intuitively TP is the number of positive pairs that are found within the same thresholded region, FP is the proportion of negative pairs found within the same region, and FN are the proportions of positive pairs found in different regions. The factor of $1/2$ appears in Equations 22-24 as both $\mathbf{P}$ and $\mathbf{S}$ are symmetric matrices under the ϵ-NN groundtruth paradigm and so each pairwise relationship between two points is counted twice: for example if $\mathbf{x}_i$ and $\mathbf{x}_j$ are true nearest neighbours then $S_{ij} = 1$ and $S_{ji} = 1$. The TP, FP and FN counts are combined using the familiar set-based F_1-measure from Information Retrieval (Equation 25):

$$F_1(\mathbf{t}_k) = \frac{2\|\mathbf{P} \circ \mathbf{S}\|_1}{\|\mathbf{S}\|_1 + \|\mathbf{P}\|_1} \tag{25}$$

The application of an F_1-measure based objective function is motivated by the highly unbalanced nature of the adjacency matrix $\mathbf{S}$: this matrix is usually very sparse, with approximately 1% of the elements being positive pairs. The F_1-measure is well known to be much less affected by this imbalanced distribution between positive and negatives (as we are not affected by true negatives) than, for instance, the classification accuracy (Chawla, 2005). Figure 16 illustrates the computation of the F_1-measure on a toy projected dimension. The overall objective function optimised is given in Equation 26.

$$\mathcal{J}_{npq}(\mathbf{t}_k) = \alpha F_1(\mathbf{t}_k) + (1-\alpha)(1-\Omega(\mathbf{t}_k)) \tag{26}$$

where $\alpha \in [0, 1]$ and the unsupervised term $\Omega(\mathbf{t}_k)$ is defined as given in Equation 27:

$$\Omega(\mathbf{t}_k) = \frac{1}{\sigma_k} \sum_{j=1}^{T+1} \sum_{i:y_i^k \in \mathbf{r}_j} \left\{ y_i^k - \mu_j \right\}^2 \tag{27}$$

where $\mathbf{r}_j = \left\{ y_i | t_{j-1} \leq y_i < t_j, y_i \in \mathbf{y}^k \right\}$ denotes the projections within thresholded region $\mathbf{r}_j$ with $t_0 = -\infty, t_{T+1} = +\infty$, $\sigma_k = \sum_{i=1}^{N_{trd}} \left\{ y_i^k - \mu_k \right\}^2$, $\mu_k \in \mathbb{R}$ denotes the mean of projected dimension $\mathbf{y}^k \in \mathbb{R}^N$ and $\mu_j \in \mathbb{R}$ denotes the mean of the projections located in thresholded region $\mathbf{r}_j$. Intuitively maximisation of Equation 26 encourages a clustering of the projected dimension so that as many of the must-link (i.e. $S_{ij} = 1$) and cannot-link (i.e. $S_{ij} = 0$) constraints encoded in the adjacency matrix $\mathbf{S}$ are respected while also minimising the cluster dispersion of the projections within each thresholded region. Equation 26 therefore fuses two valuable signals in a complementary manner: the neighbourhood structure encoded in the adjacency matrix which provides information on the pairwise relationships between the data-points in the input feature space; and the neighbourhood information captured by the projection function that was responsible for generating the projected dimensions in the first place. This semi-supervised objective function is the main point of conceptual departure from the previously discussed quantisation algorithms such as AGH (Section 5.2), MHQ (Section 5.4) and DBQ (Section 5.3) which only leverage the structure in the projected space.

The F_1-measure term in Equation 26 is non-differentiable due to the discontinuous form of Equation 21 at the threshold points $t_{k\gamma}, t_{k(\gamma+1)}$. Continuous optimisation via gradient ascent is therefore difficult. Moran et al. (Moran et al., 2013a) apply evolutionary algorithms and simulated annealing to directly optimise this objective function without appealing to a continuous relaxation. In total, for $K^{'} = \lfloor K/B \rfloor$ projected dimensions, the time complexity of this optimisation is of $O(K^{'} N_{trd}^2 TF)$, where B denotes the number of bits per projected dimension, T is the number of thresholds ($T \in [1, 2, \ldots, 15]$) and F is the number of objective function evaluations (i.e. evaluations of Equation 26). The dependence on the square of the number of training datapoints is of concern for large training datasets and dense pairwise connectivity relationships between the points. Moran (Moran, 2016) mitigates this concern somewhat by showing that, for the commonly used ϵ-NN groundtruth definition (Section 7.3.1), the adjacency matrix for learning is highly sparse thereby allowing NPQ to scale to large datasets and achieve a runtime commesurate with the k-means driven algorithm of MHQ. In a follow-up publication Moran et al. (Moran et al., 2013b) further extend their algorithm by exploring how allocating *variable thresholds* per projected dimension can positively influence the retrieval effectiveness.

5.6 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, Kohavi, & Sahami, 1995)(Garcia, Luengo, Saez, Lopez, & Herrera, 2013). Several well-known machine learning models such as Naïve Bayes (Bishop, 2006)(Yang & 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 we 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 & Irani, 1993) and multivariate (each attribute is discretised jointly) (Mehta, Parthasarathy, & Yang, 2005), (Kerber, 1992). Given the maturity of the discretisation research field we believe that there is significant potential for these already established ideas to inform the design of future scalar quantisation algorithms for hashing.

5.7 A Brief Summary

In this section we introduced five 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 hashing models (Section 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 we argued, it can lead to high quantisation errors (related data-points being assigned different bits). The multi-threshold quantisation algorithms, Hierarchical Quantisation (HQ) (Section 5.2), Manhattan Hashing Quantisation (MHQ) (Section 5.4), Double Bit Quantisation (DBQ) (Section 5.3) and Neighbourhood Preserving Quantisation (NPQ) (Section 5.5) 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, MHQ and DBQ optimise objections related to squared error and variance minimisation, and NPQ introduces a semi-supervised objective. The encoding schemes also differ significantly between three of the multi-threshold algorithms (MHQ, DBQ, HQ, NPQ), but all are designed so that the relative distance between the data-points is maximally preserved in the resulting hashcodes. We now turn our attention to a family of methods in Section 6 that learn the projections that we have just quantised.

6. Projection for Nearest Neighbour Search

In Section 4.1, we identified two main steps - *projection* and *quantisation* - that are used to generate similarity preserving hashcodes in the context of Locality Sensitive Hashing (LSH). We 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 5 we reviewed prior art that focused solely on improving the quantisation step by threshold learning. The quantisation algorithms we 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 28

$$h_k(\mathbf{x}_i) = \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k)) \tag{28}$$

where $\mathbf{w}_k \in \mathbb{R}^D$ is the hyperplane normal vector and $t_k \in \mathbb{R}$ is the quantisation threshold. Equation 28 is the popular linear hash function adopted in most hashing research. We discussed in Section 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 we review the equally important step of *projection* and focus 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 17, we show geometrically the effect of the dot product and how a projected dimension is formed using this operation.

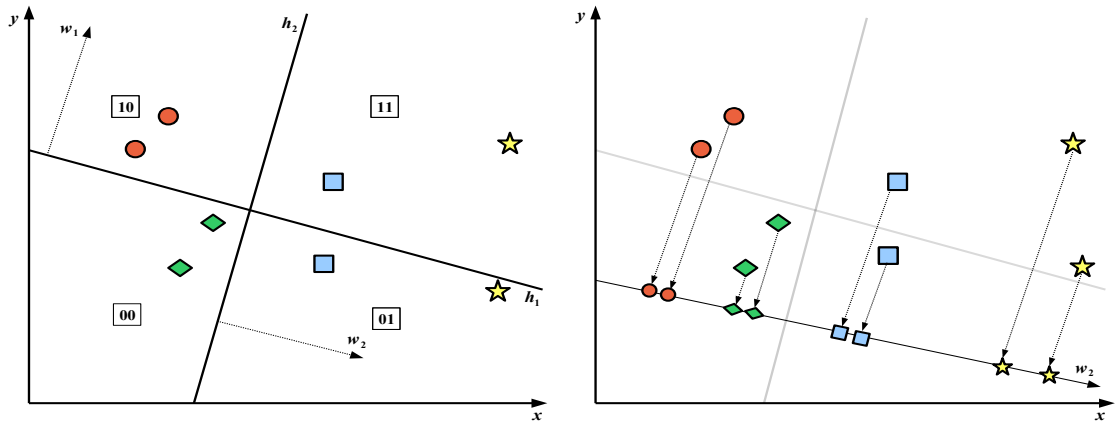
In Section 4, we introduced Locality Sensitive Hashing (LSH) a seminal early method for solving the ANN search decision problems given in Definitions 3.1-3.2. As we discussed in Section 4.1, LSH for the inner product similarity samples hyperplanes uniformly from the unit sphere, relying on an asymptotic guarantee that as the number of hyperplanes increases the Hamming distance between the hashcodes will reflect the cosine similarity between any two data-points[14]. Nevertheless, as we pointed out in Section 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; Liu, Wang, Ji, Jiang, & Chang, 2012; Weiss et al., 2008; Gong & Lazebnik, 2011; Raginsky & Lazebnik, 2009; Kulis & Darrell, 2009; Zhang, Wang, Cai, & Lu, 2010). 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 6.2), *data-dependent*

14. Goemans and Williamson (Goemans & 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.

Method	Type	Learning	Hash Function	Training Complexity	Properties	Section
LSH	I	U	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k)$	$\mathcal{O}(KD)$	E_2	4.1
SKLSH	I	U	$sgn(cos(\mathbf{w}_k^\intercal \mathbf{x} + t_k) + t_{k'})$	$\mathcal{O}(KD)$	E_2	6.2.1
PCAH	D	U	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k)$	$\mathcal{O}(min(N_{trd}^2 D, N_{trd} D^2))$	E_2, E_3, E_4	6.3.1
AGH	D	U	$sgn(\mathbf{w}_k^\intercal \mathbf{z} + t_k)$	$\mathcal{O}(N_{trd} CK)$	E_1, E_2, E_3, E_4	6.3.4
ITQ	D	U	$sgn(\mathbf{R}\mathbf{W}^\intercal \mathbf{x})$	$\mathcal{O}(K^3)$	E_1, E_2, E_3, E_4	6.3.3
SH	D	U	$sgn(sin(\frac{\pi}{2} + j\pi(\mathbf{w}_k^\intercal \mathbf{x})))$	$\mathcal{O}(min(N_{trd}^2 D, N_{trd} D^2))$	E_1, E_2, E_3, E_4	6.3.2
STH	D	S	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k)$	$\mathcal{O}(N_{trd} DK + M N_{trd}^2 K)$	E_1, E_2, E_3, E_4	6.4.4
KSH	D	S	$sgn(\mathbf{w}_k^\intercal \kappa(\mathbf{x}) + t_k)$	$\mathcal{O}(N_{trd} CK + N_{trd}^2 CK + N_{trd} C^2 K + C^3 K)$	E_1, E_2	6.4.3
BRE	D	S	$sgn(\mathbf{w}_k^\intercal \kappa(\mathbf{x}) + t_k)$	$\mathcal{O}(KN_{trd}^2 + KN_{trd} \log N_{trd})$	E_1, E_2	6.4.2
GRH	D	S	$sgn(\mathbf{w}_k^\intercal \kappa(\mathbf{x}) + t_k)$	$\mathcal{O}(N_{trd}^2 K)$+O($\mathrm{N}_{trd} DK$)	E_1, E_2	6.4.5
CVH	D	S	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k^x)$, $sgn(\mathbf{u}_k^\intercal \mathbf{z} + t_k^z)$	$\mathcal{O}(N_{trd} D^2 + D^3)$, $D{=}max(D_x, D_z)$	E_2, E_3, E_4	6.5.1
CMSSH	D	S	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k^x)$, $sgn(\mathbf{u}_k^\intercal \mathbf{z} + t_k^z)$	$\mathcal{O}(KMN_{trd} D)$, $D{=}max(D_x, D_z)$	E_1, E_2	6.5.3
CRH	D	S	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k^x)$, $sgn(\mathbf{u}_k^\intercal \mathbf{z} + t_k^z)$	$\mathcal{O}(D_x^2 D_z + D_x D_z^2 + D_z^3)$	E_1, E_2	6.5.2
PDH	D	S	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k^x)$, $sgn(\mathbf{u}_k^\intercal \mathbf{z} + t_k^z)$	$\mathcal{O}(MN_{trd}^2 K)$	E_1, E_2, E_4	6.5.4
IMH	D	S	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k^x)$, $sgn(\mathbf{u}_k^\intercal \mathbf{z} + t_k^z)$	$\mathcal{O}(N_{trd}^3)$	E_1, E_2, E_3, E_4	6.5.6
RCMH	D	S	$sgn(\mathbf{w}_k^\intercal \mathbf{x} + t_k^x)$, $sgn(\mathbf{u}_k^\intercal \mathbf{z} + t_k^z)$	$\mathcal{O}(MN_{trd} D_x K + MN_{trd} D_z K + MSK)$	E_1, E_2	6.5.5

Table 3: Categorisation of existing projection learning algorithms. This table is inspired in part by the categorisation given in (Wang et al., 2010a). Type is the data dependency of the algorithm: data independent (I) or data dependent (D). Learning is the learning paraidgm used: unsupervised (U) or supervised (S). See Section 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.



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

Figure 17: Illustration of the projection operation. Methods for projection partition the feature space with a set of hyperplanes. In Figure (a) we 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) we 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 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.

but unsupervised (Section 6.3) and *data-dependent and supervised* (Sections 6.4-6.5). The projection methods we examine in this section are categorised in Table 3. We segment the field into these three areas and review related work under each category in Sections 6.2-6.5. 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. We 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 we focus in detail on the more well-known models across each category whose authors have made the codebase freely available to the research community. We point the interested reader to two recently published review articles of Wang et al. (Wang, Shen, Song, & Ji, 2014) and Grauman and Fergus (Grauman & Fergus, 2013) for an additional overview of this part of the field. Note further that all of the hashing models we 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 scope of this review. The reader is encouraged to see Xu et al. (Xu, Wang, Li, Zeng, Li, & Yu, 2011) and Liu et al. (Liu, He, & Lang, 2013) for research in this direction.

6.1 The Four Properties of an Effective Hashcode

Before we discuss individual models for projection, we 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. (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 we have previously discussed the importance of the first property (E_1) in the context of LSH (Section 4) and binary quantisation (Section 5), we 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 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 & 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[15]. At query time we therefore avoid the degenerate case of having to examine an unnecessarily large number of nearest neighbours 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. We will study the extent to which these properties can be simultaneously preserved during the optimisation of the hashing hyperplanes in Sections 6.2-6.4.

15. Wang et al. (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.

6.2 Data-Independent Projection Methods

Aside from Locality Sensitive Hashing (LSH) which we reviewed in detail in Section 4, we will discuss one other influencial data-independent hashing method, Locality Sensitive Hashing from Shift Invariant Kernels (SKLSH) (Raginsky & Lazebnik, 2009) that extends LSH to the preservation of kernel similarity (Section 6.2.1).

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 (Raginsky & 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 (Rahimi & Recht, 2007) that provide a guarantee that the inner product between the two transformed data-points approximates the output of a *shift invariant* kernel[16] $\Psi_k(\mathbf{x}_i) \cdot \Psi_k(\mathbf{x}_j) \approx \hat{\kappa}(\mathbf{x}_i - \mathbf{x}_j)$. The random Fourier features mapping is given in Equation 29

$$\Psi_k(\mathbf{x}_i) = \sqrt{2}cos(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k) \tag{29}$$

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 (Raginsky & Lazebnik, 2009) is to use this embedding as the centerpiece of a novel hash function (Equation 30)

$$h_k(\mathbf{x}_i) = \frac{1}{2}[1 + sgn(cos(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k) + t_{k'})] \tag{30}$$

where sgn denotes the sign function adjusted so that $sgn(0) = -1$ and $t_{k'} \sim Unif[-1, 1]$. Raginsky and Lazebnik (Raginsky & 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 $\mathcal{O}(DK)$. SKLSH satisfies property E_2 of an effective hashcode, namely efficient computation of hashcodes.

6.3 Data-Dependent (Unsupervised) Projection Methods

In this section we 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

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

information in the form of user provided pairwise constraints on data-point similarity or class labels. All of the unsupervised data-dependent hashing models we 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 we do not review here, we give a brief introduction to this important solution strategy before we review the individual hashing algorithms themselves in Section 6.3.1-6.3.4[17].

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 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 31)

$$\begin{aligned} \text{argmax}_{\mathbf{V} \in \mathbb{R}^{N \times K}} \quad & tr(\mathbf{V}^\intercal \mathbf{A} \mathbf{V}) \\ \text{subject to } & \mathbf{V}^\intercal \mathbf{1} = 0 \\ & \mathbf{V}^\intercal \mathbf{B} \mathbf{V} = \mathbf{I}^{K \times K} \end{aligned} \tag{31}$$

where $\mathbf{A}$ is a symmetric matrix, $\mathbf{B}$ is positive definite matrix, $\mathbf{V}$ is an orthonormal[18] matrix and $tr(A) = \sum_i A_{ii}$. The exact specification of these matrices is projection function dependent. We will concretely define $\mathbf{A}$, $\mathbf{B}$, $\mathbf{V}$ including their dimensionalities in Sections 6.3.1-6.3.4. 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 31) 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 31 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 31) 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:

17. For more detail on trace optimisation and eigenproblems for dimensionality reduction the reader is pointed to the excellent article of Kokiopoulou et al. (Kokiopoulou, Chen, & Saad, 2011).
18. 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.

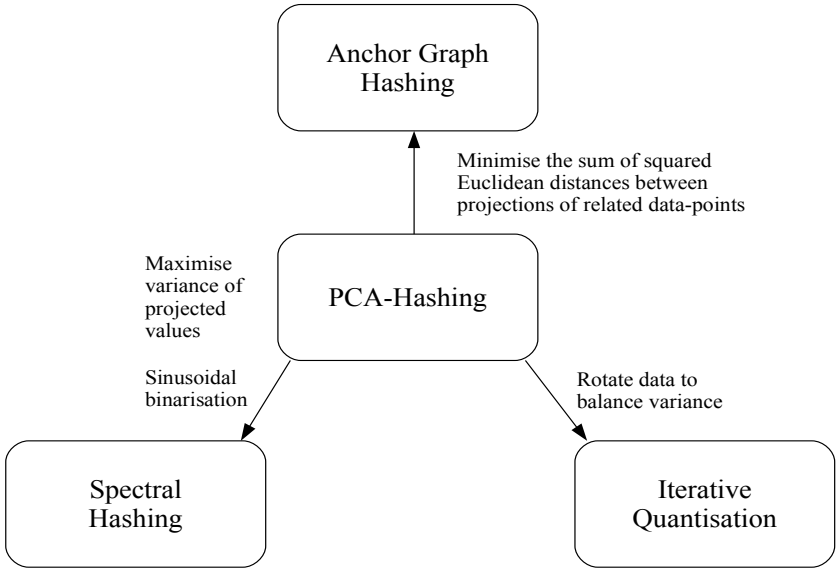


Figure 18: The data-dependent (unsupervised) hashing models are intimately related. This diagram illustrates one interpretation of that relationship where PCA is very much the centerpiece. The directed arcs are labelled with the operation necessary to transform one model into another model pointed to by the arc. See Section 6.3 for a full description of the four models.

1. Manipulating the problem into a matrix trace minimisation/maximisation (Equation 31).

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.

These four design principles are apparent in all four data-dependent hashing methods we review in this section. Specifically we survey PCA hashing (PCAH) (Section 6.3.1), Spectral Hashing (SH) (Section 6.3.2), Iterative Quantisation (ITQ) (Section 6.3.3) and Anchor Graph Hashing (AGH) (Section 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 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.

We present a diagram summarising one interpretation of the relationship between these hashing algorithms in Figure 18.

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 & Li, 2012b; Gong & Lazebnik, 2011; Weiss et al., 2008; Wang et al., 2012). We 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, Kumar, & Chang, 2010b).

We 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 32

$$\begin{aligned} \text{argmax}_{\{\mathbf{w}_k \in \mathbb{R}^D\}_{k=1}^K} \quad & \frac{1}{N_{trd}} \sum_k \mathbf{w}_k^\intercal \mathbf{X}^\intercal \mathbf{X} \mathbf{w}_k \\ = \quad & \frac{1}{N_{trd}} tr(\mathbf{W}^\intercal \mathbf{X}^\intercal \mathbf{X} \mathbf{W}) \\ \text{subject to } & \mathbf{W}^\intercal \mathbf{W} = \mathbf{I} \end{aligned} \tag{32}$$

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 6.1). Equation 32 is identical to Equation 31 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 32 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 $\mathcal{O}(min(N_{trd}^2 D, N_{trd} D^2))$ 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. 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 33.

$$h_k(\mathbf{x}_i) = \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{x}_i)) \tag{33}$$

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 6.1.

While the use of PCA is popular within the learning to hash literature, we mention here a number of disadvantages with using this matrix factorisation for generating hashcodes.

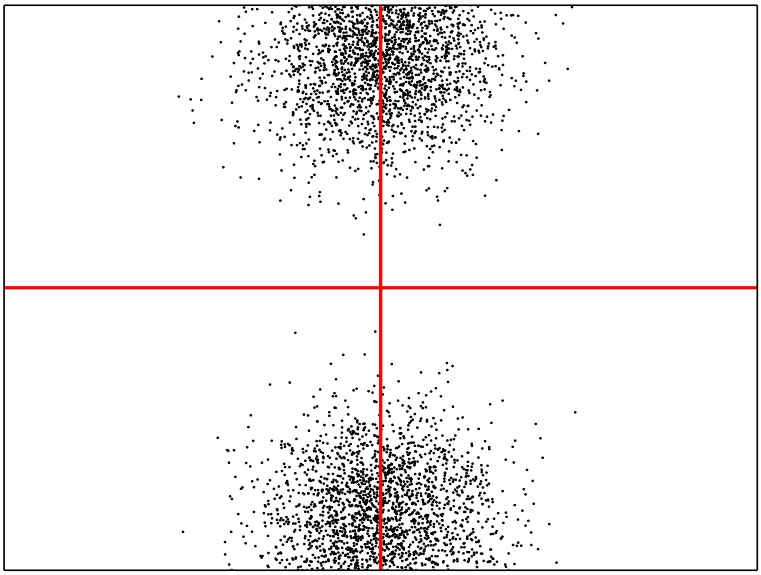


Figure 19: Plot displaying the PCA principal components $\mathbf{w}_1 \in \mathbb{R}^D$, $\mathbf{w}_2 \in \mathbb{R}^D$ (shown as perpendicular lines) for the data-points indicated by the black dots. Data has been aligned to the principal axes. The two principal components point in the directions of greatest variance of the data. These components are used as the vectors normal to the hashing hyperplanes in the PCA hashing (PCAH) algorithm.

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 33 is likely to result in poor quality hashcodes that do not discriminate well between data-points. This latter issue, which we 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 6.3.2, 6.3.4) or that transform the original data-space so that the learnt hyperplanes capture an equal amount of the variance (Section 6.3.3).

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. As we discussed in Section 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 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 34

$$\begin{aligned} \text{argmin}_{\mathbf{Y}\in\mathbb{R}^{N_{trd}\times K}} \quad & \sum_{ij} S_{ij}\|\mathbf{y}_i - \mathbf{y}_j\|^2 \\ = \quad & tr(\mathbf{Y}^\intercal(\mathbf{D}-\mathbf{S})\mathbf{Y}) \\ \text{subject to } & \mathbf{Y} \in \{-1,1\}^{N_{trd}\times K} \\ & \mathbf{Y}^\intercal\mathbf{1} = \mathbf{0} \\ & \mathbf{Y}^\intercal\mathbf{Y} = N_{trd}\mathbf{I}^{K\times K} \end{aligned} \tag{34}$$

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. (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[19]. In order to make the optimisation problem tractable Weiss et al. (Weiss et al., 2008) use the spectral relaxation trick (Shi & 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 35).

$$\begin{aligned} \text{argmin}_{\mathbf{Y}\in\mathbb{R}^{N_{trd}\times K}} \quad & tr(\mathbf{Y}^\intercal(\mathbf{D}-\mathbf{S})\mathbf{Y}) \\ \text{subject to } & \mathbf{Y} \in \mathbb{R}^{N_{trd}\times K} \\ & \mathbf{Y}^\intercal\mathbf{1} = \mathbf{0} \\ & \mathbf{Y}^\intercal\mathbf{Y} = N_{trd}\mathbf{I}^{K\times K} \end{aligned} \tag{35}$$

Equation 35 is identical to Equation 31 with $\mathbf{A} = \mathbf{D} - \mathbf{S}$ and $\mathbf{V} = \mathbf{Y}$. The solutions of Equation 35 are therefore the K eigenvectors with minimal eigenvalue of the graph Laplacian $\mathbf{D} - \mathbf{S}$[20]. 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 35

19. The interested reader is pointed to Weiss et al. (Weiss et al., 2008) for a proof.
20. The trivial eigenvector $\mathbf{1}$ with eigenvalue 0 is ignored.

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 35 is approximately $\mathcal{O}(N_{trd}^2 K)$. 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, Paiement, Vincent, Delalleau, Roux, & Ouimet, 2004)(Williams & Seeger, 2001). However without making a suitable approximation (see Section 6.3.4) this procedure is just as costly ($\mathcal{O}(N_{trd}K)$) 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. (Weiss et al., 2008) make a simple approximation by 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 36.

$$\Psi_{kj}(y_i^k) = sin(\frac{\pi}{2} + \frac{f\pi}{b_k - a_k} y_i^k) \tag{36}$$

with eigenvalues given by Equation 37:

$$\lambda_{kf} = 1 - e^{-\frac{\gamma^2}{2}|\frac{f\pi}{b_k - a_k}|^2} \tag{37}$$

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 we 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 hashcodes 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 $\mathbf{Y} = \mathbf{XW}$ 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 $\{\Psi_{kf}\}_{f=1}^K$ and their associated eigenvalues $\{\lambda_{kf} \in \mathbb{R}\}_{f=1}^K$ given by Equations 36-37.

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 $\left\{\bar{a}_k, \bar{b}_k\right\}_{k=1}^{K}$. 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 38 using $\left\{\bar{\Psi}_k, \bar{\mathbf{w}}_k, \bar{a}_k, \bar{b}_k\right\}_{k=1}^{K}$ retained in Step 5 of the pre-processing stage. Using Equation 36 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 6.2.1).

$$h_k(\mathbf{q}) = \frac{1}{2}(1 + sgn(\bar{\Psi}_k(\bar{\mathbf{w}}_k^{\intercal}\mathbf{q})) \tag{38}$$

The computational complexity of this algorithm is dominated by the $\mathcal{O}(min(N_{trd}^2 D, N_{trd}\ D^{\ 2}))$ 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 37. 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 6.3.1) and usually leads to a higher retrieval effectiveness (Liu et al., 2011; Moran et al., 2013b). 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. (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. (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. We examine AGH in Section 6.3.4.

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

Algorithm 4: Iterative Quantisation (ITQ) (Gong & 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 $\mathbf{R} \in \mathbb{R}^{K \times K}$

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

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

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

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

  $\mathbf{R} \leftarrow \hat{\mathbf{S}}\mathbf{S}^\intercal$ `// Rotation minimising Eq 39 for fixed` $\mathbf{B}$

end

return R

variance problem, Iterative Quantisation (ITQ) seeks to balance the variance across PCA hyperplanes through a learnt rotation of the feature space[21]. 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 39

$$\begin{aligned} \text{argmin}_{\mathbf{B} \in \mathbb{R}^{N_{trd} \times K}, \mathbf{R} \in \mathbb{R}^{K \times K}} \quad & \|\mathbf{B} - \mathbf{YR}\|_F^2 \\ \text{where } & \mathbf{B} \in \{-1, 1\}^{N_{trd} \times K} \\ \text{subject to } & \mathbf{R}^\intercal\mathbf{R} = K\mathbf{I}^{K \times K} \end{aligned} \tag{39}$$

Equation 39 is similar to the orthogonal Procrustes[22] 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 39 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 $\mathbf{R}$.

21. We comment briefly on why ITQ is considered a method of projection in this survey, rather than quantisation. In Section 5, we 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 as it does not directly convert real-valued projections to binary relying instead on SBQ (Section 5.1) for quantisation. We 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.

22. 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.

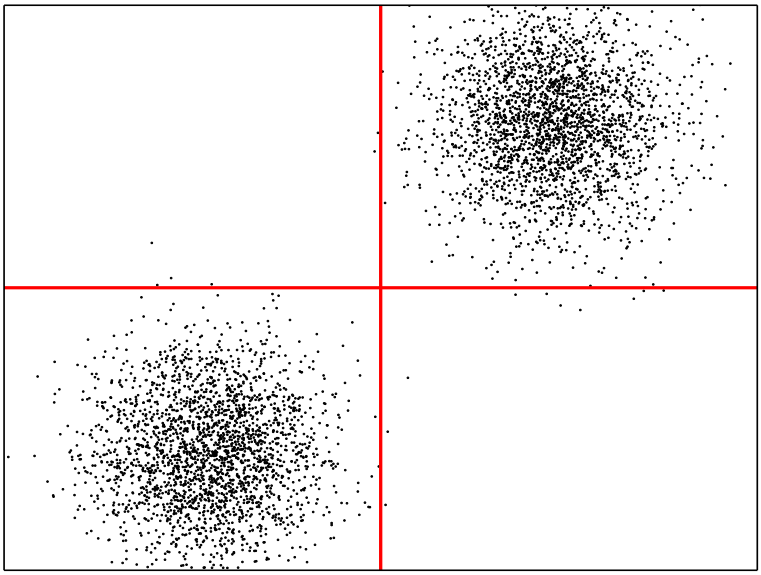


Figure 20: The effect of an ITQ rotation of the feature space. Here we show the same data as in Figure 19 but rotated by $\mathbf{R} \in \mathbb{R}^{K \times K}$ as found by ITQ over 100 iterations. The variance is more evenly distributed between the two hyperplanes (indicated as perpendicular lines) 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 & Lazebnik, 2011).

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 are learnt individually with the other fixed (Gong & 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 (Hanson & Norris, 1981) and Arun et al. (Arun, Huang, & Blostein, 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 (Gong & Lazebnik, 2011) show that the optimal $\mathbf{B}$ minimising Equation 39 can be obtained simply by using single bit quantisation (Section 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 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. (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 $\mathcal{O}(K^3)$ 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 40

$$g_l(\mathbf{q}) = \frac{1}{2}(1 + sgn(\mathbf{R}\mathbf{W}^\intercal\mathbf{q})) \tag{40}$$

where we assume the data has been mean-centered so that the quantisation threshold $t_k = 0$.

We conclude with two 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 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 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. 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. The reader is encouraged to see Section 6.4.2 and Liu et al. (Liu, Mu, Kumar, & Chang, 2014) for an overview.

6.3.4 Anchor Graph Hashing (AGH)

We previously described the Hierarchical Quantisation (HQ) algorithm employed by Anchor Graph Hashing (AGH) in Section 5.2. In this section we focus exclusively on the AGH component that learns the projection function. AGH examines the same relaxed objective function as SH which we repeat in Equation 41 for reading convenience

$$\begin{aligned} \text{argmin}_{\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}} \quad & tr(\mathbf{Y}^\intercal(\mathbf{D} - \mathbf{S})\mathbf{Y}) \\ \text{subject to } & \mathbf{Y} \in \mathbb{R}^{N_{trd} \times K} \\ & \mathbf{Y}^\intercal \mathbf{1} = \mathbf{0} \\ & \mathbf{Y}^\intercal \mathbf{Y} = N_{trd} \mathbf{I}^{K \times K} \end{aligned} \tag{41}$$

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 $\mathcal{O}(N_{trd}^2 D)$ 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 41 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 6.3.2 this out-of-sample extension can be derived using the Nyström method (Bengio et al., 2004)(Williams & Seeger, 2001) requiring $\mathcal{O}(N_{trd}K)$ 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 (Section 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. We first presented the anchor graph formulation in Equation 10 in the context of the Hierarchical Quantisation (HQ) method which for convenience we repeat in Equation 42

$$Z_{ij} = \begin{cases} \dfrac{exp(-d^2(\mathbf{x}_j, \mathbf{c}_i)/\gamma)}{\sum\limits_{i' \in \langle j \rangle} exp(-d^2(\mathbf{x}_j, \mathbf{c}_{i'}))/\gamma)} & \text{if } i \in \langle j \rangle \\ 0 & otherwise \end{cases} \tag{42}$$

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 $\mathcal{O}(N_{trd}CD)$ rather than $\mathcal{O}(N_{trd}^2 D)$ for $\mathbf{S}$. Liu et al. (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 $\mathbf{\Sigma}^{1/2}\mathbf{Z}^\intercal\mathbf{Z}\mathbf{\Sigma}^{-1/2}$. Given a bit budget of K in the hierarchical variant of their algorithm, Liu et al. (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 43.

$$\mathbf{Y} = \sqrt{N_{trd}}\mathbf{Z}\Sigma^{-1/2}\mathbf{V}\Lambda^{-1/2} = \mathbf{ZW} \tag{43}$$

The training time complexity of computing $\mathbf{Y}$ is $\mathcal{O}(N_{trd}CK^{'})$. 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 42. Liu et al. (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 42. 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 44)

$$h_k(\mathbf{q}) = \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{z})) \tag{44}$$

where we 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 $\mathcal{O}(CD + CK^{'})$ operations, a testing time complexity that is a marked improvement over the $\mathcal{O}(N_{trd}K)$ time complexity of the Nyström method[23]. Rather than generate one bit per hash function as suggested by Equation 44, 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 previously discussed this Hierarchical Quantisation (HQ) algorithm in detail in Section 5.2.

6.3.5 A Brief Summary

In this section we surveyed a selection of the more well-known unsupervised algorithms that position the hashing hyperplanes based on the distribution of the data. We reviewed Principal Components Analysis Hashing (PCAH) (Section 6.3.1), Iterative Quantisation (ITQ) (Section 6.3.3), Spectral Hashing (SH) (Section 6.3.2) and Anchor Graph Hashing (AGH) (Section 6.3.4). We 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 one bit per hyperplane or simply uses the PCA hyperplanes as is.

We 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

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

AGH is its computationally efficient method, based on the Nyström method of (Williams & Seeger, 2001), for out-of-sample extension to unseen data-points.

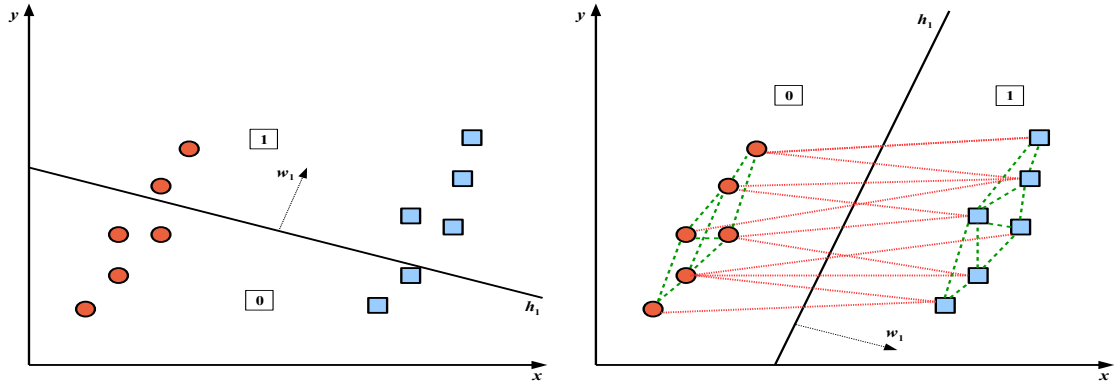
Aside from AGH which makes an honest attempt at reducing the computational complexity at training time, the downside with most of these hashing algorithms is the severe computational penalty $\mathcal{O}(min(N^2D, ND^2))$ required for solving the SVD or eigenvalue problem making their application intractable 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.

6.4 Data-Dependent (Supervised) Projection Methods

In Section 4 and Sections 6.2-6.3 we 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, Worring, Santini, Gupta, & Jain, 2000). A robust way of linking these two domains is one of the grand challenges in the fields of object recognition, image retrieval and image annotation (Moran & Lavrenko, 2015a).

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[24]. In the visual search domain, Grauman et al. (Grauman & 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 we review in this section. In general, we define a supervised hashing model as a model that leverages the same type of information (e.g. class labels, metric distances) in the hash function learning algorithm that was also used to compute the groundtruth information for evaluation purposes. In Figure 21, we 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.

24. Listwise and triplet-based supervision has also been used to learn supervised hashing models, but we do not consider this type of learning algorithm in this review. The reader should consult Wang et al. (Wang, Liu, Kumar, & Chang, 2015a) for more information on these type of models.



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

Figure 21: 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. We 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 a similar manner to the data-dependent (unsupervised) models, we restrict our 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. We review ITQ with a Canonical Correlation Analysis (CCA) embedding (ITQ + CCA) (Gong & Lazebnik, 2011), Supervised Hashing with Kernels (KSH) (Liu et al., 2012), Binary Reconstructive Embedding (BRE) (Kulis & Darrell, 2009), Self-Taught Hashing (STH) (Zhang et al., 2010) and Graph Regularised Hashing (GRH)(Moran & Lavrenko, 2015a). These five models 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, GRH uses a random walk over a neighbourhood graph to enforce consistency betweent the bits of similar data-points, while ITQ+CCA frames an objective that maximises the correlation of the labels and data-point projections. The relationship between these five supervised hashing models is summarised in Figure 22.

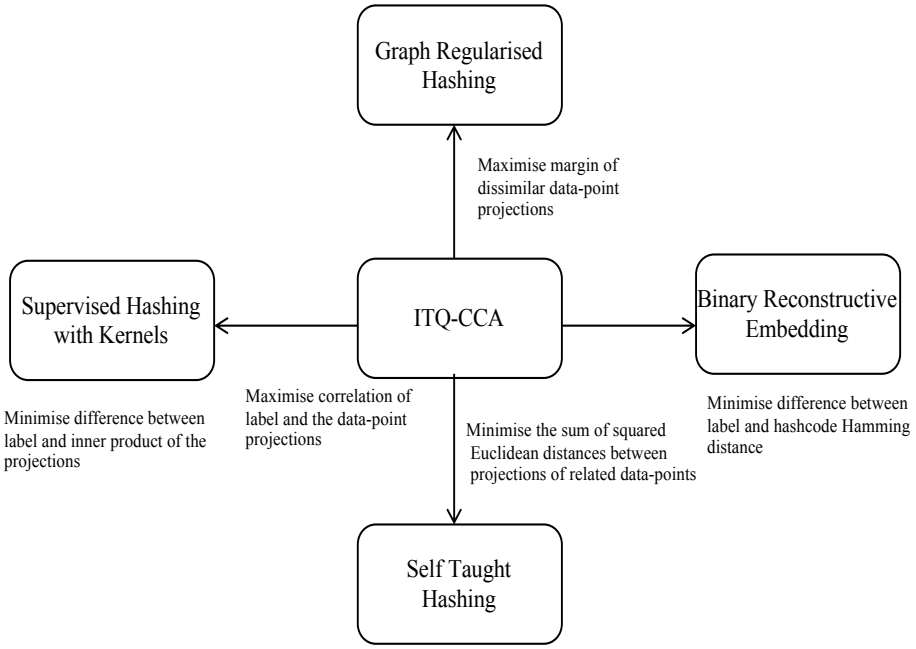


Figure 22: Relationship between the five supervised hashing models reviewed in this section. The labels on the arrows indicate the transformation necessary to convert between the different models. The fundamental difference between the models arises in how the available labels are related to the projections/hashcodes so as to compute an error signal to adjust the hashing hyperplanes.

6.4.1 ITQ + Canonical Correlation Analysis (CCA)

We reviewed the unsupervised variant of Iterative Quantisation (ITQ) in Section 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 original 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 & Lazebnik, 2011) replace PCA with Canonical Correlation Analysis (CCA) (Hardoon, Szedmak, & Shawe-Taylor, 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 45).

$$\begin{aligned} \text{argmax}_{\mathbf{w}_k \in \mathbb{R}^D, \mathbf{u}_k \in \mathbb{R}^D} \quad & \frac{\mathbf{w}_k \mathbf{X}\mathbf{Z}\mathbf{u}_k}{\sqrt{\mathbf{w}_k^\intercal \mathbf{X}^\intercal \mathbf{X}\mathbf{w}_k \mathbf{u}_k^\intercal \mathbf{Z}^\intercal \mathbf{Z}\mathbf{u}_k}} \\ \text{subject to } & \mathbf{w}_k^\intercal \mathbf{X}^\intercal \mathbf{X}\mathbf{w}_k = 1 \\ & \mathbf{u}_k^\intercal \mathbf{Z}^\intercal \mathbf{Z}\mathbf{u}_k = 1 \end{aligned} \tag{45}$$

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

$$\mathbf{X}^\intercal \mathbf{Z}(\mathbf{Z}^\intercal \mathbf{Z} + \rho \mathbf{I})^{-1} \mathbf{Z}^\intercal \mathbf{X}^\intercal \mathbf{w}_k = \lambda_k^2 (\mathbf{X}^\intercal \mathbf{X} + \rho \mathbf{I}) \mathbf{w}_k \tag{46}$$

where λ_k is the eigenvalue and ρ is a regularisation constant set to 0.0001 in (Gong & Lazebnik, 2011). Repeatedly solving Equation 46 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 6.3.3). If we denote $D = max(D_x, D_z)$, then the computational complexity of ITQ+CCA is bounded by $\mathcal{O}(N_{trd}D^2 + D^3)$. This is made up of the $\mathcal{O}(N_{trd}D^2)$ operations required to compute the covariance matrices and the $\mathcal{O}(D^3)$ operations arising from the matrix multiplications, inversion and solving the eigenvalue problem (Rasiwasia, Mahajan, Mahadevan, & Aggarwal, 2014). Following a similar line of argument to ITQ, ITQ+CCA approximately preserves properties E_1-E_4 of an effective hashcode.

6.4.2 Binary Reconstructive Embedding (BRE)

Binary Reconstructive Embedding (BRE) is the only projection method we 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 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. We present the *supervised* variant of the BRE objective function in Equation 47

$$\begin{aligned}
&\text{argmin}_{\mathbf{W}\in\mathbb{R}^{D\times K}} \sum_{ij\in\mathbf{S}\in\{0,1\}^{N_{trd}\times N_{trd}}} \left\{(1-S_{ij}) - \frac{1}{K}\|g(\mathbf{x}_i) - g(\mathbf{x}_j)\|_2^2\right\}^2 \\
&\text{subject to } g(\mathbf{x}_i) = [h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \ldots, h_k(\mathbf{x}_i)]^\intercal \\
&\text{where } h_k(\mathbf{x}_i) = \frac{1}{2} sgn(1 + \sum_{j=1}^{C} W_{jk}\kappa(\mathbf{x}_j, \mathbf{x}_i))
\end{aligned} \tag{47}$$

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 $\left\{\kappa : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}\right\}$. Kulis and Darrell (Kulis & Darrell, 2009) set κ to be the linear kernel in the original publication.

The objective function in Equation 47 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 $\left\{\kappa : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}\right\}$ 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 44) 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 47 with the sign function intact. To optimise the non-differentiable objective function, Kulis and Darrell (Kulis & Darrell, 2009) formulate a coordinate descent algorithm that cycles through each hash function one by one and finds the value minimising Equation 47 of a randomly chosen element W_{jk} of each hyperplane $\mathbf{W}_{\bullet k}$, while holding the remaining hyperplanes constant. Kulis and Darrell (Kulis & Darrell, 2009) provide a closed form solution for computing the optimal W_{jk} in $\mathcal{O}(N_{trd}^2)$ time. This procedure is repeated for the remaining hash functions. In total one iteration through all K hash functions takes $\mathcal{O}(KN_{trd}^2 + KN_{trd}\log N_{trd})$ operations[25]. 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. (Liu et al., 2014) and Shen et al. (Shen, Shen, Liu, & Tao Shen, 2015).

25. For ease of presentation we 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.

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 6.3.4) and BRE (Section 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 we 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 review. The familiar kernelised hash function is presented in Equation 48

$$h_k(\mathbf{q}) = sgn(\sum_{j=1}^{C} W_{jk}\kappa(\mathbf{x}_j, \mathbf{q}) + t_k) \tag{48}$$

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 49) 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.

$$\begin{aligned} \text{argmin}_{\mathbf{W} \in \mathbb{R}^{C \times K}} \quad & \sum_{ij \in \mathbf{S} \in \mathbb{R}^{N_{trd} \times N_{trd}}} \left\{ S_{ij} - \frac{1}{K} g^\intercal(\mathbf{x}_i) g(\mathbf{x}_j) \right\}^2 \\ \text{subject to } & g(\mathbf{x}_i) = [h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \ldots, h_K(\mathbf{x}_i)]^\intercal \\ & h_k(\mathbf{x}_i) = sgn(\sum_{j=1}^{C} W_{jk}\kappa(\mathbf{x}_j, \mathbf{x}_i)) \end{aligned} \tag{49}$$

Recall from Section 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, we drop the sign function and rewrite Equation 49 to iterate over the K hash functions rather than data-point pairs (Equation 50)

$$\begin{aligned} \text{argmin}_{\mathbf{W}\in\mathbb{R}^{C\times K}} \quad & \sum_{k=1}^{K} \|K\mathbf{S} - \mathbf{y}^k(\mathbf{y}^k)^\intercal\|_F^2 \\ \text{where } y_i^k = & \sum_{j=1}^{C} W_{jk}\kappa(\mathbf{x}_j, \mathbf{x}_i) \end{aligned} \tag{50}$$

where we have assumed that the data is mean centered, and therefore $t_k = 0$. Recall from Table 1 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 50 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 51

$$\mathbf{R}^{k-1} = K\mathbf{S} - \sum_{l=1}^{k-1} \mathbf{y}^l(\mathbf{y}^l)^\intercal \tag{51}$$

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. (Liu et al., 2012) show that the objective function in Equation 50 can be reduced to Equation 52

$$\begin{aligned} \text{argmax}_{\mathbf{w}_k\in\mathbb{R}^C} \quad & (\mathbf{K}\mathbf{w}_k)^\intercal\mathbf{R}^{k-1}(\mathbf{K}\mathbf{w}_k) \\ \text{where } K_{ij} = & \kappa(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to } & (\mathbf{K}\mathbf{w}_k)^\intercal(\mathbf{K}\mathbf{w}_k) = L \end{aligned} \tag{52}$$

where $\mathbf{K} \in \mathbb{R}^{N_{trd}\times C}$ is the kernel matrix. Comparing the form of Equation 52 to the standard eigenvalue problem template presented in Equation 31 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 52. 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 optimisation algorithm with time complexity $\mathcal{O}(NCK + N_{trd}^2CK + N_{trd}C^2K + C^3K)$ by limiting the number C, N_{trd}[26] of sampled data-points used to construct the hash functions and by

26. Typically $N_{trd} = 1000$ and $C = 300$.

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.

6.4.4 Self-Taught Hashing (STH)

Self-taught hashing (STH) (Zhang et al., 2010) 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 6.3.4) and Spectral Hashing (SH) (Section 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 $\mathbf{L} = \mathbf{D} - \mathbf{S}$. We present the now familiar graph Laplacian optimisation objective in Equation 53

$$\begin{aligned} \text{argmin}_{\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}} \quad & tr(\mathbf{Y}^\intercal(\mathbf{D} - \mathbf{S})\mathbf{Y}) \\ \text{subject to} \quad & \mathbf{Y} \in \mathbb{R}^{N_{trd} \times K} \\ & \mathbf{Y}^\intercal \mathbf{D}\mathbf{1} = \mathbf{0} \\ & \mathbf{Y}^\intercal \mathbf{D}\mathbf{Y} = N_{trd}\mathbf{I}^{K \times K} \end{aligned} \tag{53}$$

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 53 and the objective of SH (Equation 35). The diagonal degree matrix $\mathbf{D}$ makes an appearance in the constraints of Equation 53, which gives a normalised cut of $\mathbf{S}$ rather than a ratio-cut (Aggarwal & Reddy, 2014) when the graph Laplacian eigenvectors are binarised. Equation 53 is therefore equivalent to the Laplacian Eigenmap embedding (Belkin & Niyogi, 2003). The solutions of Equation 53 are the eigenvectors corresponding to the lowest eigenvalues of the generalised eigenvalue problem $L\mathbf{y}^k = \lambda \mathbf{D}\mathbf{y}^k$. The eigenvalue problem can be solved in $\mathcal{O}(MN_{trd}^2 K)$ operations using M iterations of the Lanczos algorithm (Golub & Van Loan, 1996), (Zhang et al., 2010). Note that, as for BRE (Section 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 53 approximately preserves properties E_3 and E_4 of an effective hashcode (Section 6.1).

Equation 53 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 we 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 & Seeger, 2001) as in AGH (Section 6.3.4) or making a separable uniform distribution approximation as for SH (Section 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 vec-

tors $\left\{\mathbf{w}_k \in \mathbb{R}^D\right\}_{k=1}^K$ necessary for out-of-sample extension to unseen data-points. Training K linear SVMs takes $\mathcal{O}(N_{trd}DK)$ time (Joachims, 2006) while out-of-sample extension (test time) is $\mathcal{O}(DK)$ for a single test data-point.

6.4.5 Graph Regularised Hashing (GRH)

Graph Regularised Hashing (GRH) (Moran & Lavrenko, 2015a) is a simple and scalable iterative hashing model that borrows inspiration from the core innovations of STH and ITQ. At the beginning the hashcodes $\{\mathbf{b}_1 \ldots \mathbf{b}_{N_{trd}}\}$ are initialised by running the points $\{\mathbf{x}_1 \ldots \mathbf{x}_{N_{trd}}\}$ through any existing unsupervised or supervised projection function, such as LSH (Indyk & Motwani, 1998) or ITQ+CCA (Gong & Lazebnik, 2011). Having initialised the hashcodes GRH iteratively performs the following three key steps to learn a set of hashing hyperplanes:

1. *Regularisation*: GRH uses a graph-based approach to regularise the hashcodes. The nodes of the graph correspond to the points $\{\mathbf{x}_1 \ldots \mathbf{x}_{N_{trd}}\}$ and $\mathbf{S}$ plays the role of an adjacency matrix: an undirected edge is inserted between nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ if and only if $S_{ij} = 1$. Each node $\mathbf{x}_i$ is annotated with K binary labels, corresponding to the K bits of the hashcode $\mathbf{b}_i$. The aim is to increase the similarity of the label sets at the opposite ends of each edge in the graph. This is achieved by averaging the label set of each node with the label sets of its immediate neighbours, and in doing so the N_{trd} hashcodes $\{\mathbf{b}_1 \ldots \mathbf{b}_{N_{trd}}\}$ are made more consistent with the adjacency matrix $\mathbf{S} \in \mathbb{R}^{N_{trd} \times N_{trd}}$ (Equation 54). This technique is similar to the *score regularisation* method of Diaz (Diaz, 2007) and the *label propagation* algorithm of Zhu and Ghahramani (Zhu & Ghahramani, 2002).

$$\mathbf{B}_m \leftarrow \text{sgn}\left(\alpha\ \mathbf{S}\mathbf{D}^{-1}\mathbf{B}_{m-1} + (1-\alpha)\mathbf{B}_0\right) \tag{54}$$

   Here $m \in [1, \ldots, M]$, where M is the maximum number of iterations, $\mathbf{S}$ is the adjacency matrix and $\mathbf{D}$ is a diagonal matrix containing the degree of each node in the graph[27]. $\mathbf{B} \in \{-1, +1\}^{N_{trd} \times K}$ represents the labels assigned to every node at the previous step of the algorithm, $\mathbf{B}_0$ indicates the labels at iteration 0, namely as initialised by LSH or ITQ+CCA, $\alpha \in [0, 1]$ is a scalar smoothing parameter and sgn represents the sign function, modified so that $\text{sgn}(0) = -1$. The hashcodes at the current iteration $\mathbf{B}_m$ are set to be a convex combination of the hashcodes at the previous iteration $\mathbf{B}_{m-1}$ and the initialised hashcodes ($\mathbf{B}_0$).

2. *Partitioning*: at the end of the regularisation step, each point $\mathbf{x}_i$ has K binary labels $\{-1, +1\}$. These labels are used to learn a set of hypersurfaces $\{\mathbf{h}_1 \ldots \mathbf{h}_K\}$. Each surface $\mathbf{h}_k \in \mathbb{R}^D$ partitions the space $\mathbb{R}^D$ into two disjoint regions: *positive* and *negative*. The positive region of $\mathbf{h}_k$ should envelop all points $\mathbf{x}_i$ for which the k'th label was $+1$; while the negative region should contain all the $\mathbf{x}_i$ for which $B_{ik} = -1$. These hyperplanes are needed to efficiently compute the hashcodes for testing points $\mathbf{x} \in \mathbb{R}^D$, where we have no affinity information available (out-of-sample extension). The partitioning of the feature space is achieved using using `liblinear` (Fan, Chang, Hsieh, Wang, & Lin, 2008).

27. $\mathbf{D}^{-1}$ has the effect of L_1-normalising the rows of $\mathbf{S}$.

3. *Prediction*: the estimated hyperplane normal vectors $\{\mathbf{w}_1 \ldots \mathbf{w}_K\}$ are used to re-label the data-points:

$$B_{ik} = \text{sgn}(\mathbf{w}_k{}^\intercal \mathbf{x}_i + t_k) \;\; \text{for } i{=}\{1 \ldots N_{trd}\} \text{ and } k{=}\{1 \ldots K\} \tag{55}$$

The effect of this step is that points which could not be classified correctly will now be relabelled to make them consistent with all hyperplanes. After the last iteration, the hyperplane normal vectors $\{\mathbf{w}_1 \ldots \mathbf{w}_K\}$ are used to predict hashcodes for new instances $\mathbf{x}$: the k'th bit in the code is set to 1 if $\mathbf{w}_k^\intercal \mathbf{x} + t_k > 0$, otherwise it is zero.

The above three steps are repeated in a manner reminiscent of the *EM algorithm* (Dempster, Laird, & Rubin, 1977): the regularised hashcodes from step 1 adjust the hyperplanes in step 2, and these in turn generate new hashcodes in step 3 which are then passed into step 1. Figure 23 illustrates the operation of GRH on a toy example dataset. If we let N_{trd} denote the number of training data-points then the graph regularisation step is of $\mathcal{O}(N_{trd}^2 K)$. Training a linear SVM takes $\mathcal{O}(N_{trd} DK)$ time (Joachims, 2006) while prediction (test time) is $\mathcal{O}(N_{trd} DK)$. Therefore linear GRH is $\mathcal{O}(M N_{trd}^2 K)$ for M iterations. Typically the adjacency matrix $\mathbf{S}$ is sparse[28], $N_{trd} \ll N$ and K is small (≤ 128 bits) thereby ensuring that the *linear* variant of GRH is readily scalable to large datasets. GRH meets criteria E_1 and E_2 of an effective hashcode, due to firstly, the effective use of supervision (GRH is shown to be more effective than the other models discussed in this review on standard datasets (Moran & Lavrenko, 2015a)(Moran, 2016) and secondly, the use of efficient dot products to compute hashcodes, respectively.

6.4.6 Deep Neural Network-based Hashing

Given the recent revolution brought about by high capacity deep neural networks within the field of computer vision, and in particular within object detection (Girshick, Donahue, Darrell, & Malik, 2014) and image classification (Krizhevsky, Sutskever, & Hinton, 2012) it would be amiss in this review not to touch upon the effect these models have had within the learning-to-hash field. In contrast to the previously discussed shallow hashing models, these deep models can be thought of as learning multiple projection matrices $\mathbf{W}$ instead of just one. We describe a mix of older (Semantic Hashing) and more recent deep hashing models in this last section.

Arguably the first instance of a neural network-based hashing model was the early work of Salakhutdinov and Hinton (Salakhutdinov & Hinton, 2009) with a proposal for *Semantic Hashing*. Their focus was on learning binary hashcodes for efficient search over textual documents represented as word count vectors. The central idea behind this approach is to represent a query document as an address in memory in a way that documents in nearby memory addresses (obtained by bit flipping the hashcode for the query document) will be semantically similar to the query document[29]. To tackle this problem, Salakhutdinov and Hinton propose a deep generative hash function. Figure 24 shows the network architecture

28. For example, around 10% of the entries in $\mathbf{S}$ are non-zero for the CIFAR-10 dataset.
29. Contrast this with LSH and the other hashing models we have discussed that seek to map similar data-points to the same hashtable buckets, avoiding the need for continguous memory or a Hamming ball search strategy.

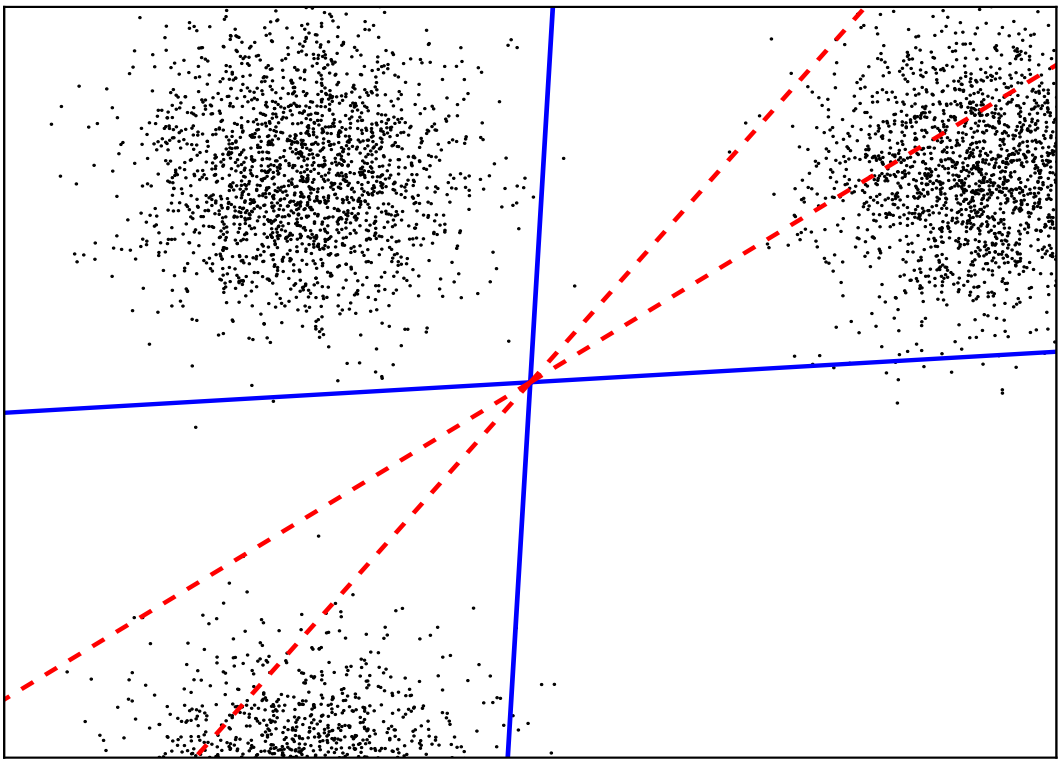


Figure 23: Synthetic toy example illustrating the Graph Regularised Hashing (GRH) model (Moran & Lavrenko, 2015a). The toy dataset consists of 6,000 data-points clustered into three distinct clusters. The data-points in each cluster are of the same class as each other (out of three possible classes), and therefore each cluster should ideally end up in its own bucket (region). The dashed lines indicate the two hyperplane normal vectors produced by Locality Sensitive Hashing (LSH). In this case many data-points from different classes (clusters) end up in the same bucket (mAP=0.6803). GRH refines the two LSH hyperplanes to produce the hyperplane normal vectors shown with the solid lines. In this case, the data-points from the three different classes are almost all in their own bucket (mAP=0.9931).

of the model alongside the two-step training strategy. For effective learning, the deep generative model is trained in two distinct stages: firstly there is a *pretraining stage* in which the network is greedily trained layer-by-layer as a stack of Restricted Boltzmann Machines (RBMs). This has the effect of initialising the network to a good region in weight space. To fully capitalise on the multiple layers, it is then necessary to fine tune the model by unrolling the RBMs to form an undercomplete deep autoencoder[30] in which the code dimension is less than the input dimension (Figure 24). The deep autoencoder is trained via backpropagation so that the discrete word counts can be optimally reconstructed from the latent space representation at the code (middle) layer. This fine-tuning has the effect of bringing the network to a local optimum in parameter space and is critical for good retrieval performance. To prevent the logistic units from saturating in their middle range,

30. An autoencoder is a feedforward neural network in which the input and output are identical.

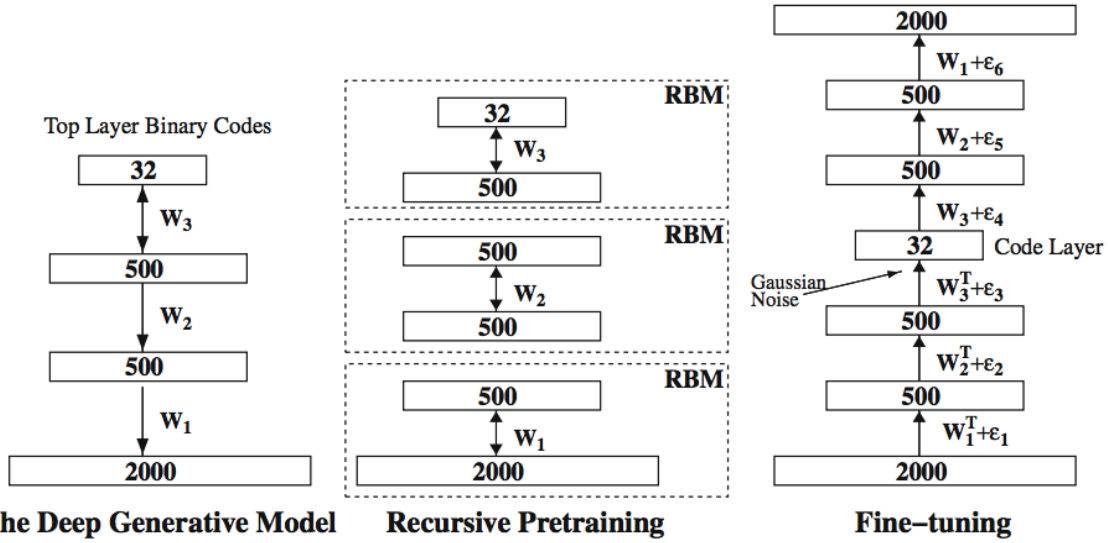


Figure 24: The Semantic Hashing model of Salakhutdinov and Hinton (Salakhutdinov & Hinton, 2009) is formalised as a deep generative hashing model. In this example, it is assumed a document is a 2000 dimensional bag of words. The neural network consists of 2000 units in the input layer, 500 units in the second and third layers, and 32 logistic units in the output layer. The model is trained in two phases: firstly, a *recursive pretraining* step in which the network is segemented into a stack of Restricted Boltzmann Machines (RBMs). The feature activations of an RBM in an earlier layer is used as data for the RBM in the adjacent layer. Secondly, there follows a *fine-tuning* stage in which the RBMs are unrolled to form a deep autoencoder with logistic units at the bottleneck (code) layer. The autoencoder is effectively fine-tuned to be good at re-constructing document word counts from binary features. Gaussian noise is added to the bottleneck layer to encourage the activations to be close to binary. A test time, to obtain the hashcode for a novel document we perform a single forward pass through the deep generative model. This is a matrix multiplication followed by a component-wise non-linearity for each hidden layer. Image reproduced with permission of Salakhutdinov and Hinton (Salakhutdinov & Hinton, 2009).

Gaussian noise is added to the inputs of the code layer units during fine-tuning. The Gaussian noise has the effect of encouraging the activities to be approximately bimodal (i.e. close to 0 or 1). Semantic hashing has since been extended to images by Torralba et al. (Torralba et al., 2008) who mapped from data-points to hashcodes using a stack of RBMs trained to minimise a Neighbourhood Components Analysis (NCA) objective on image labels (Goldberger, Roweis, Hinton, & Salakhutdinov, 2004). Petrović (Petrovic, 2012) notes two weaknesses with the original semantic hashing approach to learning highly non-linear hash functions: firstly, the training of a deep generative model in this manner is significantly slower than indexing the collection using LSH or a data dependent linear hash function such as those described in Section 6.4. Secondly, Salakhutdinov and Hinton (Salakhutdinov & Hinton, 2009) evaluate the retrieval effectiveness using cosine similarity

but perform the retrieval in Euclidean space with E^2LSH[31]. This mismatch should perhaps be kept in mind when interpreting the retrieval results. Indeed Kulis and Darrell (Kulis & Darrell, 2009) found the unsupervised variant of the deep generative model of Salakhutdinov and Hinton (Salakhutdinov & Hinton, 2009) to have a *lower* retrieval effectiveness than unsupervised BRE (Section 6.4.2).

Since the advent of Semantic Hashing, and particularly after the 2012 revival of deep neural networks for computer vision tasks (Krizhevsky et al., 2012), there has been an upsurge in research activity on models for learning deep highly non-linear hash functions for image retrieval applications. The commonality between most of these more recent models is the clever integration of the four properties of effective hashcodes (Section 6.1) into the deep neural network learning framework. Prime examples include the Deep Hashing (DH) and Supervised Deep Hashing (SDH) models of Liong et al. (Liong, Lu, Wang, Moulin, & Zhou, 2015), the Convolutional Neural Network hashing (CNNH) model of Xia et al. (Xia, Pan, Lai, Liu, & Yan, 2014), and the fully end-to-end Deep Neural Network Hashing (DNNH) model of Lai et al. (Lai, Pan, Liu, & Yan, 2015). Liong et al. (Liong et al., 2015) propose an unsupervised (DH) and a supervised deep neural network (SDH) model for learning binary hashcodes. They formulate an objective function that aims to minimise the loss between the binary hashcode of an image and its feature representation, while maintaining the bit balance and independence properties of effective hashcodes. In the DH variant, after initialising the first layer weights using PCA, the weights of all layers are refined by optimising the objective function via stochastic gradient descent and backpropagation. Binary hashcodes are obtained for an image by performing a forward pass and thresholding the outputs of the last layer using the sign function. The supervised variant of their model is formed by integrating pairwise must and cannot-link constraints into the objective function. In a similar manner to Semantic Hashing, these models have been found to either equally or less effective than much shallower learning schemes such as CCA+ITQ (Section 6.4.1) and KSH (Section 6.4.3). Wang et al. (Wang et al., 2015a) suggest that this might in part be due to the lack of a pre-training phase in the learning of DH and SDH.

All of the hash functions so far reviewed in this literature survey assume that a suitable image feature extractor has been applied to the image to obtain hand-crafted image features such as GIST or SIFT descriptors. These image features are then fed into the learning stage to obtain the projection matrices necessary for generating the hashcodes for unseen data-points. It would be particularly attractive if we could discard these hand-crafted image features and perform the learning directly on the raw image pixels, letting the learning procedure itself determine the most effective image representation for generating binary hashcodes. This tight coupling of the feature representation learning and hash function learning might be expected to yield superior retrieval effectiveness to models that treat each as disjoint and separate. It is exactly this approach that is taken by the Convolutional Neural Network hashing (CNNH) model of Xia et al. (Xia et al., 2014) and the Deep Neural Network hashing model (DNNH) of Lai et al. (Lai et al., 2015). Xia et al. (Xia et al., 2014) propose a two-step hashcode learning methodology for obtaining deep hash functions, consisting of a hashcode learning stage and a hash function learning stage. This type of two-step decomposition commonly avoids a complex and non-convex optimisation procedure

31. `http://www.mit.edu/~andoni/LSH/`

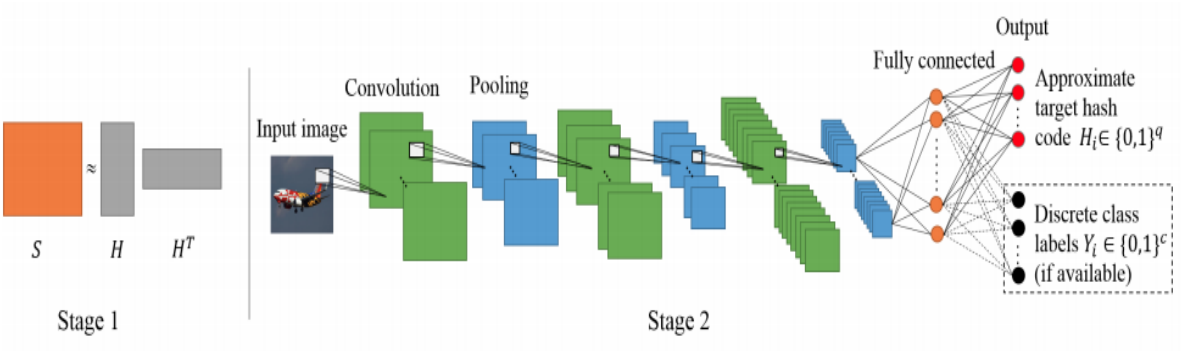


Figure 25: The Convolutional Neural Network hashing (CNNH) model of Xia et al. In the first step the similarity matrix **S** is decomposed into the product of two matrices $\mathbf{HH}^T$, where H is the matrix of approximate target hashcodes for the data-points in the training dataset (Xia et al., 2014). The convolutional neural netowrk is then trained to predict the approximate target hashcodes using backpropagation. See text for a description. Image courtesy of Xia et al. (Xia et al., 2014).

and has also been successfully applied in shallow models such as Self Taught Hashing (STH) (Section 6.4.4) and Graph Regularised Hashing (Section 6.4.5). Their method is dubbed Convolutional Neural Network hashing (CNNH). In the first step the similarity matrix **S** that denotes the must and cannot-link constraints between training images is decomposed using a coordinate descent procedure into the product of two matrices. These matrices can be thought of as representing the approximate hashcodes of the training images. In the second hash function learning stage a convolutional neural network is trained directly on image pixels to predict the target hashcodes for each training image as obtained by the decomposition procedure in the first step. The model is shown to outperform shallow models such as KSH (Section 6.4.3) in the experimental evaluation.

Lai et al. (Lai et al., 2015) avoid the need for this two-step procedure in their Deep Neural Network hashing model (DNNH), which is also shown in Figure 26. DNNH offers several innovations to tightly couple representation and hash function learning. In contrast to the usual pairwise constraints used all the models we have so far reviewed in this survey, Lai et al. (Lai et al., 2015) leverage triplet-based supervision to train the network. A tripet consists of three images I_1, I_2, I_3, in which I_1 is more similar to I_2 than it is to I_3. DNNH employs a shared sub-network with stacked convolution layers that extracts intermediate image representations for each image in the triplet. This sub-network follows the Network-in-Network architecture of Lin et al. (Lin, Chen, & Yan, 2014). The intermediate image features are subsequently fed into a divide-and-encode module that maps the features to binary hashcodes. This mapping is performed by dividing the features into K slices of equal length with each slice being mapped by a fully connected layer, a sigmoid activation function and a piecewise thresholding function to obtain one binary haschode bit. The intuition behind the divide-and-conquer layer is to reduce the possible redundancy in the bits that might result from a more standard fully-connected layer. Given this network topology, a triplet ranking loss is proposed to learn the parameters of the neural network using stochastic

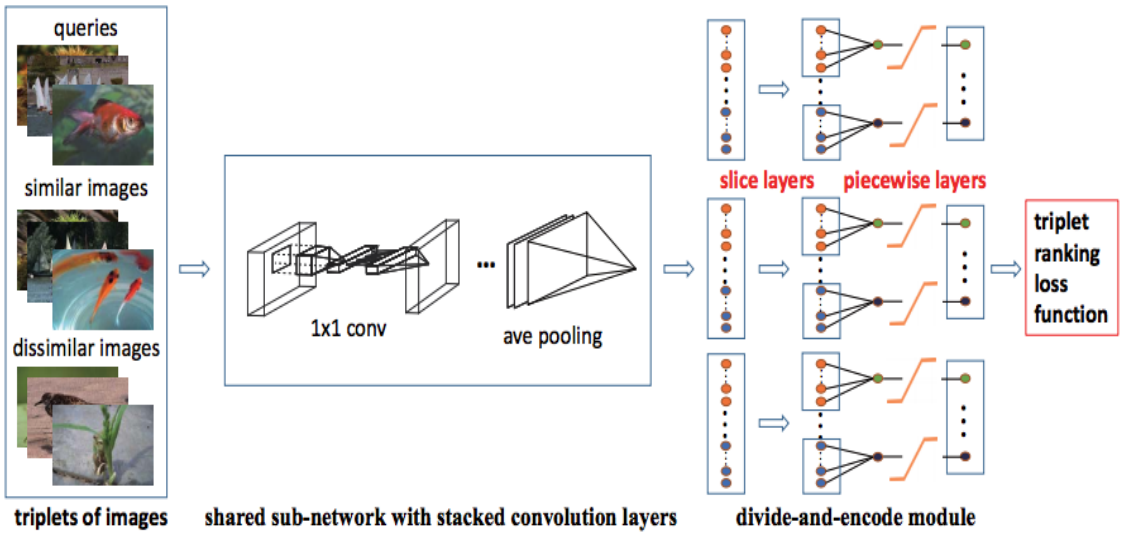


Figure 26: The Deep Neural Network Hashing (DNNH) model learns binary hash functions directly from raw image pixels in an end-to-end framework using a sub-network with stacked convolution layers and a divide and conquer module all optimised with a triplet ranking loss. See text for a description. Image courtesy of Lai et al (Lai et al., 2015).

gradient descent so that the relative similarities between images are preserved in the binary hashcodes. Importantly, this end-to-end learning scheme is shown to yield superior retrieval effectiveness to models that rely on hand-crafted image features and two-step hash function learning procedures, an encouraging finding that bodes well for the future applicability of these models in the field.

6.4.7 A Brief Summary

We have reviewed a sample of five models that inject a supervised signal into the learning of the hashing hyperplanes for unimodal ANN search. The methods reviewed included ITQ+CCA (Section 6.4.1), Binary Reconstructive Embedding (BRE) (Section 6.4.2), Supervised Hashing with Kernels (KSH) (Section 6.4.3), Self Taught Hashing (STH) (Section 6.4.4) and Graph Regularised Hashing (Section 6.4.5). 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). De-

spite 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. We concluded the section by examining a selection of recent deep learning-based hash functions (Section 6.4.6) that are beginning to gain traction in the field. Despite initial concerns with early models, these highly non-linear models have a promising future in the field for those willing to accept a much longer training time in return for superior retrieval effectiveness.

6.5 Cross-Modality Projection Methods

Locality sensitive hashing (LSH) and its kernelised variant SKLSH which were both described in Section 4 and Section 6.2.1 and the data-dependent hashing models presented in Sections 6.3-6.4 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[32]. For example, consider an image from the popular photo sharing website Flickr[33] 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 27), or give the retrieval system geographical coordinates and receive images related to that locality.

The data-dependent hashing models we 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 we assume that both datasets have the same number of training data-points N_{trd}, and we further denote as N_{xz} the number of *paired* data-points across the modalities ($N_{xz} \leq N_{trd}$). The logical relationship 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 six 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

32. We use the term 'modality' and 'feature space' interchangeably in this survey.
33. `http://www.flickr.com`

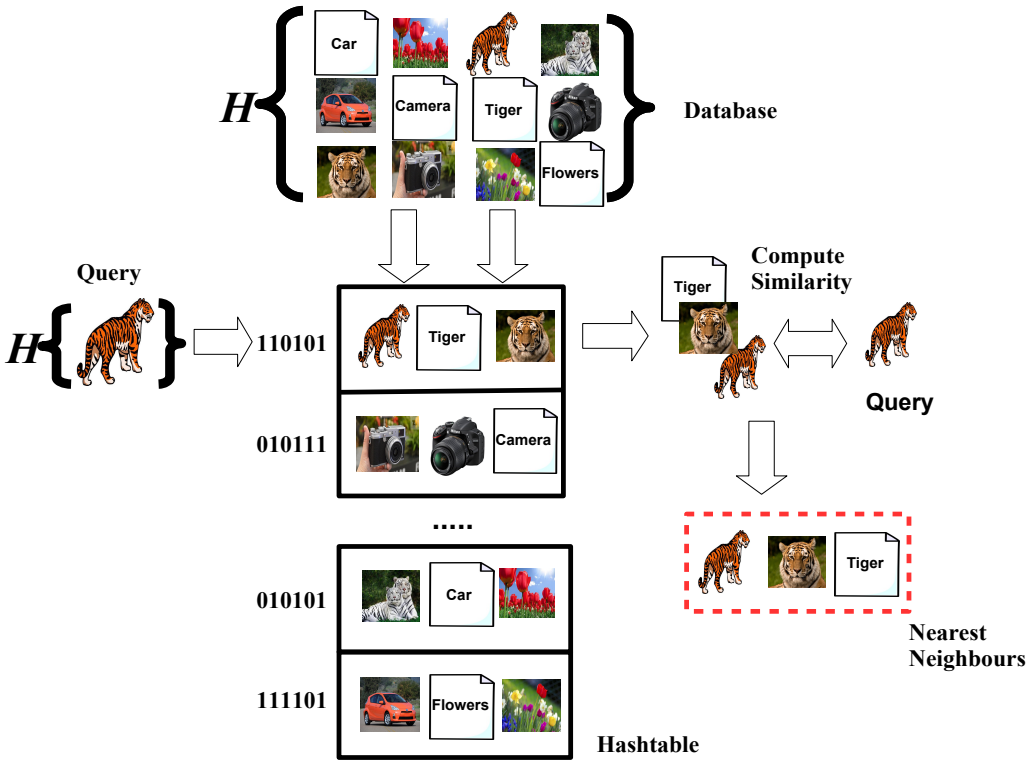
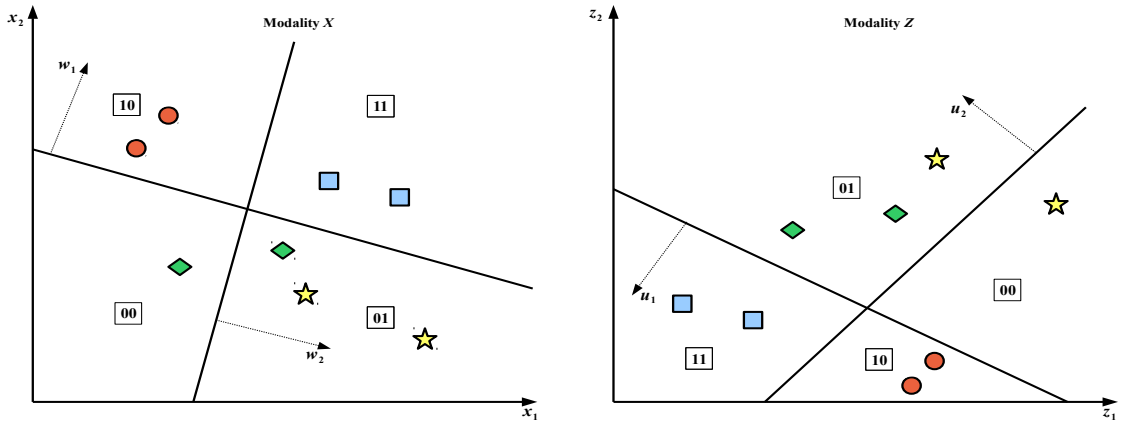


Figure 27: Cross-modal hashing-based ANN search. In the cross-modal variant of hashing-based ANN search we wish to partition the input-space such that similar data-points across modalities fall into the same hashtable buckets. In this diagram we show how cross-modal hash functions can be used to retrieve 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.

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 cat and a text snippet describing a cat). In Section 4 and Sections 6.3-6.4, we discussed how to learn K hyperplanes that assign similar data-points similar hashcodes in the *same* modality. We 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. We provide an intuitive high-level overview of this fundamental concept in Figure 28. The cross-modal hashing models analysed in this review include Cross View Hashing (CVH) model of (Kumar & Udupa, 2011) (Section 6.5.1), Co-Regularised Hashing (CRH) (Zhen & Yeung, 2012) (Section 6.5.2), Predicable Dual View Hashing (PDH) (Rastegari, Choi, Fakhraei, III, & Davis, 2013) (Section 6.5.4),



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

Figure 28: The essence of learning to hash across modalities. In Figure (a) we 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) we 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.

Regularised Cross-Modal Hashing (RCMH) (Section 6.5.5), Inter-Media Hashing (IMH) (Song, Yang, Yang, Huang, & Shen, 2013) (Section 6.5.6), Regularised Cross Modal Hashing (RCMH) (Moran & Lavrenko, 2015b) (Section 6.5.5) and Cross Modal Semi-Supervised Hashing (CMSSH) (Bronstein, Bronstein, Michel, & Paragios, 2010) (Section 6.5.3).

6.5.1 Cross View Hashing (CVH)

Cross View Hashing (CVH)[34] (Kumar & Udupa, 2011) is equivalent to $ITQ+CCA$ (Section 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 56 presents the hash functions for both modalities

34. As is standard in the literature we 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 & Udupa, 2011)).

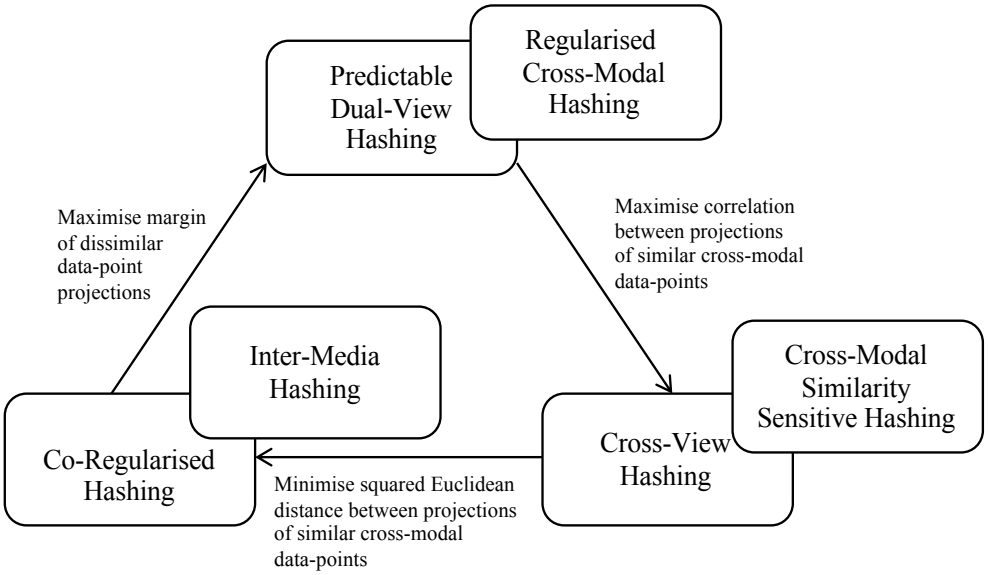


Figure 29: Relationship between the six cross-modal hashing models reviewed in this section. We only consider the inter-modal consistency term when relating the models (ignoring intra-modal and out-of-sample extension terms). The labels on the arcs denote the essential transform required to convert one model into the model(s) pointed to by the arc.

$$\begin{aligned} h_k^{\mathcal{X}}(\mathbf{x}_i) &= \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{x}_i)) \\ h_k^{\mathcal{Z}}(\mathbf{z}_i) &= \frac{1}{2}(1 + sgn(\mathbf{u}_k^\intercal \mathbf{z}_i)) \end{aligned} \tag{56}$$

Following the same argument as for ITQ+CCA, the asymptotic computational complexity of CVH is $\mathcal{O}(N_{trd}D^2 + D^3)$ 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 we will review in Sections 6.5.2-6.5.6 introduce new schemes for learning both sets of hyperplane that achieve a higher retrieval effectiveness than CVH on standard image-text datasets.

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 & Yeung, 2012). Boosting (Freund & 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 6.4.3). CRH uses the standard linear hash function (Equation 56) as for CVH (Section 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 57

$$\begin{aligned}\text{argmin}_{\mathbf{w}_k \in \mathbb{R}^{D_x}, \mathbf{u}_k \in \mathbb{R}^{D_z}} \quad & \frac{1}{N_x}\sum_{i=1}^{N_x}[1 - |\mathbf{w}_k^\intercal \mathbf{x}_i|]_+ + \frac{1}{N_z}\sum_{j=1}^{N_z}[1 - |\mathbf{u}_k^\intercal \mathbf{z}_j|]_+ \\ & \gamma \sum_{i,j=1}^{N_{xz}} \alpha_{ij} S_{ij} (\mathbf{w}_k^T \mathbf{x}_i - \mathbf{u}_k^\intercal \mathbf{z}_j)^2 + \frac{\lambda_x}{2}\|\mathbf{w}_k\|^2 + \frac{\lambda_z}{2}\|\mathbf{u}_k\|^2\end{aligned} \tag{57}$$

where $\{\alpha_{ij} \in \mathbb{R}_+\}_{i,j=1}^{N_{trd}}$ are weights updated using Adaboost (Freund & 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[35].

Equation 57 is non-convex and so Zhen & Yeung (Zhen & 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 & 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 & 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 57 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 $\mathcal{O}(KMND)$ where $D = max(D_x, D_z)$ and M is the number of iterations required for convergence of the Pegasos solver (Shalev-Shwartz, Singer, & Srebro, 2007). Properties E_1, E_2 of an effective hashcode (Section 6.1) are preserved by CRH but not properties E_3, E_4.

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 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 58 that effectively coerces the k^{th} pair of hash functions to correct mistakes committed by the k-1 previous hash functions

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

$$\text{argmax}_{\mathbf{w}_k \in \mathbb{R}^{D_x}, \mathbf{u}_k \in \mathbb{R}^{D_z}} \quad \sum_{i,j=1}^{N_{xz}} \alpha_{ij} S_{ij} sgn(\mathbf{w}_k^\intercal \mathbf{x}_i) sgn(\mathbf{u}_k^\intercal \mathbf{z}_j) \tag{58}$$

where $\{\alpha_{ij} \in \mathbb{R}_+\}_{i,j=1}^{N_{xz}}$ are per data-point pair weights adjusted using Adaboost (Freund & 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 58 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 6.4.3) and SH (Section 6.3.2). This relaxation simply involves dropping the sign function altogether giving Equation 59.

$$\begin{aligned} \text{argmax}_{\mathbf{w}_k \in \mathbb{R}^{D_x}, \mathbf{u}_k \in \mathbb{R}^{D_z}} \quad & \sum_{i,j=1}^{N_{xz}} \alpha_{ij} S_{ij} (\mathbf{w}_k^\intercal \mathbf{x}_i)(\mathbf{u}_k^\intercal \mathbf{z}_j) \\ = \quad & \mathbf{w}_k^\intercal \left\{ \sum_{ij=1}^{N_{xz}} \alpha_{ij} S_{ij} \mathbf{x}_i \mathbf{z}_j^\intercal \right\} \mathbf{u}_k \\ = \quad & \mathbf{w}_k^\intercal \mathbf{C} \mathbf{u}_k \end{aligned} \tag{59}$$

Equation 59 can be solved in closed form by performing an SVD on the matrix $\mathbf{C} \in \mathbb{R}^{D_x \times D_z}$ taking $\mathcal{O}(D_x^2 D_z + D_x D_z^2 + D_z^3)$ operations[36]. 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 56) 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.

6.5.4 Predictable Dual-View Hashing (PDH)

Predictable Dual-View Hashing (PDH) (Rastegari et al., 2013) is fully specified in Algorithm 5. 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 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. The PDH objective function is presented in Equation 60

36. 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 6.3.

$$\begin{aligned}\text{argmin}_{\mathbf{W}\in\mathbb{R}^{D_x\times K},\mathbf{U}\in\mathbb{R}^{D_z\times K}} \quad & \|\mathbf{B}^x\mathbf{B}^x-\mathbf{I}\|_2^2+\|\mathbf{B}^z\mathbf{B}^z-\mathbf{I}\|_2^2+\sum_{k=1}^{K}\|\mathbf{w}_k\|^2+\sum_{k=1}^{K}\|\mathbf{u}_k\|^2 \\ & +C_x\sum_{i,k=1}^{N_{trd}}\xi_{ik}^x+C_z\sum_{i,k=1}^{N_{trd}}\xi_{ik}^z \\ \text{subject to } & \mathbf{B}^x=sgn(\mathbf{XW}) \\ & \mathbf{B}^z=sgn(\mathbf{ZU}) \\ & b_{ik}^z(\mathbf{w}_k^\intercal\mathbf{x}_i)\geq 1-\xi_{ik}^x \\ & b_{ik}^x(\mathbf{u}_k^\intercal\mathbf{z}_i)\geq 1-\xi_{ik}^z\end{aligned} \tag{60}$$

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. (Rastegari et al., 2013) solve Equation 60 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 61

$$\begin{aligned}\text{argmin}_{\mathbf{Y}\in\mathbb{R}^{N_{trd}\times K}} \quad & tr(\mathbf{Y}_l^\intercal(\mathbf{D}_l-\mathbf{S}_l)\mathbf{Y}_l) \\ \text{subject to } & \mathbf{Y}_l\in\mathbb{R}^{N_{trd}\times K} \\ & \mathbf{Y}_l^\intercal\mathbf{1}=0 \\ & \mathbf{Y}_l^\intercal\mathbf{Y}_l=N_{trd}\mathbf{I}^{K\times K}\end{aligned} \tag{61}$$

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 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

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

Initialise $\mathbf{W}$, $\mathbf{U}$ via CCA from $\mathbf{X}$, $\mathbf{Z}$
for $m \leftarrow 1$ ***to*** M **do**
  for $k \leftarrow 1$ ***to*** K **do**
    $\mathbf{b}_k^x = \mathbf{B}_{\bullet k}^x$
    $\mathbf{b}_k^z = \mathbf{B}_{\bullet k}^z$
    Train SVM_k^x with $\mathbf{b}_k^z$ as labels, training dataset $\mathbf{X}$
    Obtain hyperplane $\mathbf{h}_k^x$
    $\mathbf{W}_{\bullet k} = \mathbf{w}_k$
    $\mathbf{B}_{\bullet k}^x = sgn(\mathbf{X}\mathbf{w}_k)$
    Train SVM_k^z with $\mathbf{b}_k^x$ as labels, training dataset $\mathbf{Z}$
    Obtain hyperplane $\mathbf{h}_k^z$
    $\mathbf{U}_{\bullet k} = \mathbf{u}_k$
    $\mathbf{B}_{\bullet k}^z = sgn(\mathbf{Z}\mathbf{u}_k)$
  end
  Update $\mathbf{B}^x$, $\mathbf{B}^z$ by solving eigenvalue problem in Equation 61.
end
return $\mathbf{W}, \mathbf{U}$

the projection vectors as defined by the inner product 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 56) to hash novel cross-modal data-points into hashtable buckets. The training time complexity is dominated by the $\mathcal{O}(MN_{trd}^2K)$ operations to solve the eigenvalue problems across M iterations using the Lanczos algorithm (Golub & Van Loan, 1996).

6.5.5 Regularised Cross-Modal Hashing (RCMH)

Regularised cross-modal hashing (RCMH) proposed by Moran et al. (Moran & Lavrenko, 2015b) is a cross-modal extension of the unimodal graph regularised hashing (GRH) model (Moran & Lavrenko, 2015a). The hash functions $h_k^{\mathcal{X}}, h_k^{\mathcal{Z}}$ are based on K hyperplanes each: $\{\mathbf{w}_1 \ldots \mathbf{w}_K\}$ for the space of words and $\{\mathbf{u}_1 \ldots \mathbf{u}_K\}$ for the space of visual features. The hyperplane $\mathbf{w}_j$ is used to assign the j'th bit in the annotation hashcode, while $\mathbf{u}_j$ determines the j'th bit in the visual hashcode. Moran et al. (Moran & Lavrenko, 2015b) initialise all hyperplanes randomly using unimodal LSH, and iteratively perform the following three steps summarised in Algorithm 6:

1. *R*egularisation, where the hashcodes $\{\mathbf{b}_1 \ldots \mathbf{b}_N\}$ are made more consistent with the affinity matrix $\mathbf{S}$. This step is illustrated in Figure 30. Shown are five images $a \ldots e$ with their initial hashcodes (K=2 bits for this example). The lines between images reflect the neighbourhood structure encoded in the affinity matrix $\mathbf{S}$. Image d has a

Algorithm 6: Regularised Cross Modal Hashing (RCMH)(Moran & Lavrenko, 2015b)

Input: Annotation descriptors $\mathbf{X}_{trd} \in \mathbb{R}^{N_{trd} \times D_x}$, Visual descriptors $\mathbf{Z}_{trd} \in \mathbb{R}^{N_{trd} \times D_z}$, adjacency matrix $\mathbf{S} \in \{0,1\}$, degree matrix $\mathbf{D} \in \mathbb{Z}_+$, interpolation parameter $\alpha \in [0,1]$, number of iterations $M \in \mathbb{Z}_+$

Output: Hyperplanes $\{\mathbf{w}_1 \ldots \mathbf{w}_K\}$, $\{\mathbf{u}_1 \ldots \mathbf{u}_K\}$, biases $\{t_1 \ldots t_K\}$, $\{o_1 \ldots o_K\}$

1 Initialise $\mathbf{B}_0 \in \{0,1\}^{N_{trd} \times K}$ via LSH or ITQ+CCA from $\mathbf{X}$
2 $\mathbf{B}_0 = \text{sgn}(\mathbf{B}_0 - \frac{1}{2})$
3 $\mathbf{B} = \mathbf{B}_0$
4 **for** $m \leftarrow 1$ ***to*** M **do**
5 $\quad \mathbf{B} = \text{sgn}\left(\alpha \mathbf{S}\mathbf{D}^{-1}\mathbf{B} + (1-\alpha)\mathbf{B}_0\right)$
6 $\quad$ **for** $k \leftarrow 1$ ***to*** K **do**
7 $\quad\quad \mathbf{b}_k = \mathbf{B}(:,k)$
8 $\quad\quad$ Train SVM_k^x with $\mathbf{b}_k$ as labels, training dataset $\mathbf{X}_{trd} \in \mathbb{R}^{N_{trd} \times D_x}$
9 $\quad\quad$ Train SVM_k^z with $\mathbf{b}_k$ as labels, training dataset $\mathbf{Z}_{trd} \in \mathbb{R}^{N_{trd} \times D_z}$
10 $\quad\quad$ Obtain hyperplanes $\mathbf{w}_k, \mathbf{u}_k$ and biases t_k, o_k
11 $\quad$ **end**
12 $\quad B_{ik} = \text{sgn}(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k)$ for $i = \{1 \ldots N_{trd}\}$ and $k = \{1 \ldots K\}$
13 **end**
14 **return** $\{\mathbf{w}_k\}_{k=1}^K, \{\mathbf{u}_k\}_{k=1}^K, \{t_k\}_{k=1}^K, \{o_k\}_{k=1}^K$

hashcode (-11), but its neighbours b, c, e have hashcodes (-1-1), (11) and (1-1) respectively. The arrow beside the initial hashcode (-11) of image d shows the update of its hashcode: its hashcode changes to (1-1), which is more consistent with neighbouring hashcodes (on average).

2. *P*artitioning, where the hyperplanes $\mathbf{w}_j, \mathbf{u}_j$ are adjusted to be consistent with the j'th bit of the hashcodes from step (2). Adjusting the visual hyperplanes based on the annotation bits is how the necessary cross-modal bridge is formed. The approach is illustrated in Figure 31. Five images $a \ldots e$ are shown in two sets of coordinates: the word space on the top and the visual feature-space on the bottom. Each image is associated with a 2-bit hashcode, and each bit is used to learn a maximum-margin hyperplane that bisects the corresponding space. For example, the first bit has value -1 for images a, b and value 1 for images c, d, e, giving rise to hyperplanes $\mathbf{w}_1$ and $\mathbf{u}_1$, shown as dark lines on the top and the bottom parts of Figure 31. Note that $\mathbf{w}_1$ and $\mathbf{u}_1$ look very different, because they are defined over two completely different modalities: words on the top and visual features on the bottom. Similarly, the second bit results in the hyperplanes $\mathbf{w}_2$ and $\mathbf{u}_2$.

3. *P*rediction, the hyperplanes $\{\mathbf{w}_1 \ldots \mathbf{w}_K\}$ are then used to assign hashcodes $\{\mathbf{b}_1 \ldots \mathbf{b}_N\}$ to the training images which are then fed back into step 1 ready for the next iteration. Pseudocode describing the salient parts of the model is provided in Algorithm 6. The key difference between GRH (Section 6.4.5) is shown on Lines 8-9, where the

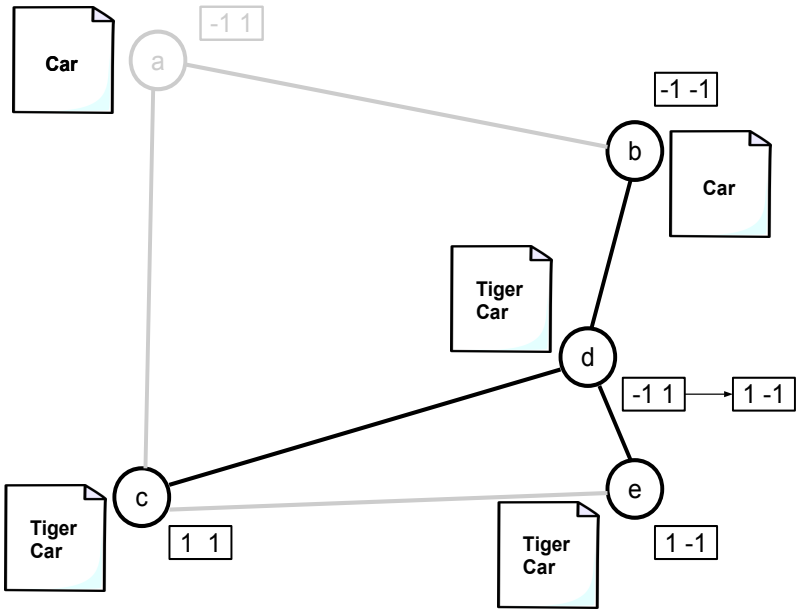


Figure 30: Regularisation step: the hashcode for annotation node d is updated to be more similar with its neighbours (c,b,e).

annotation space bits are used to learn hyperplanes in the annotation and visual feature spaces.

The model bears similarities with the Predictable Dual-View Hashing (PDH) hash function of (Rastegari et al., 2013) which we reviewed in Section 6.5.4. PDH employs an Expectation-Maximisation (EM)-like iterative learning scheme with a max-margin formulation to refine the positioning of the hashing hyperplanes within both feature spaces. In contrast to RCMH, PDH integrates supervision into the hyperplane learning by solely relying on initialising the hashcodes with Canonical Correlation Analysis (CCA). We previously described CCA in detail as part of the review of the ITQ+CCA model in Section 6.4.1. CCA finds two sets of K hyperplanes, one set of K hyperplanes for each feature space, that result in projections that are maximally correlated for related data-points across the two modalities. PDH also explicitly seeks to induce a pairwise independence between the hashcode bits through the solution of a graph Laplacian eigenvalue problem after each iteration. In a different manner to PDH, RCMH does not seek to enforce bit independence and also does not rely on an initial CCA initialisation to integrate the supervisory signal into learning algorithm. Instead RCMH eliminates the need to solve either eigenvalue system, relying on graph regularisation to enforce the data-point must-link and cannot-link constraints. The training time of RCMH can be characterised by $\mathcal{O}(MN_{trd}D_xK + MN_{trd}D_zK + MSK)$ with the testing (prediction) time given by $\mathcal{O}(N_{teq}DK)$. RCMH achieves properties E_1 and E_2 of an effective hashcode given that it generates effective cross-modal hashcodes (Moran & Lavrenko, 2015b) and is efficient as prediction time by using dot products to compute the hashcode bits.

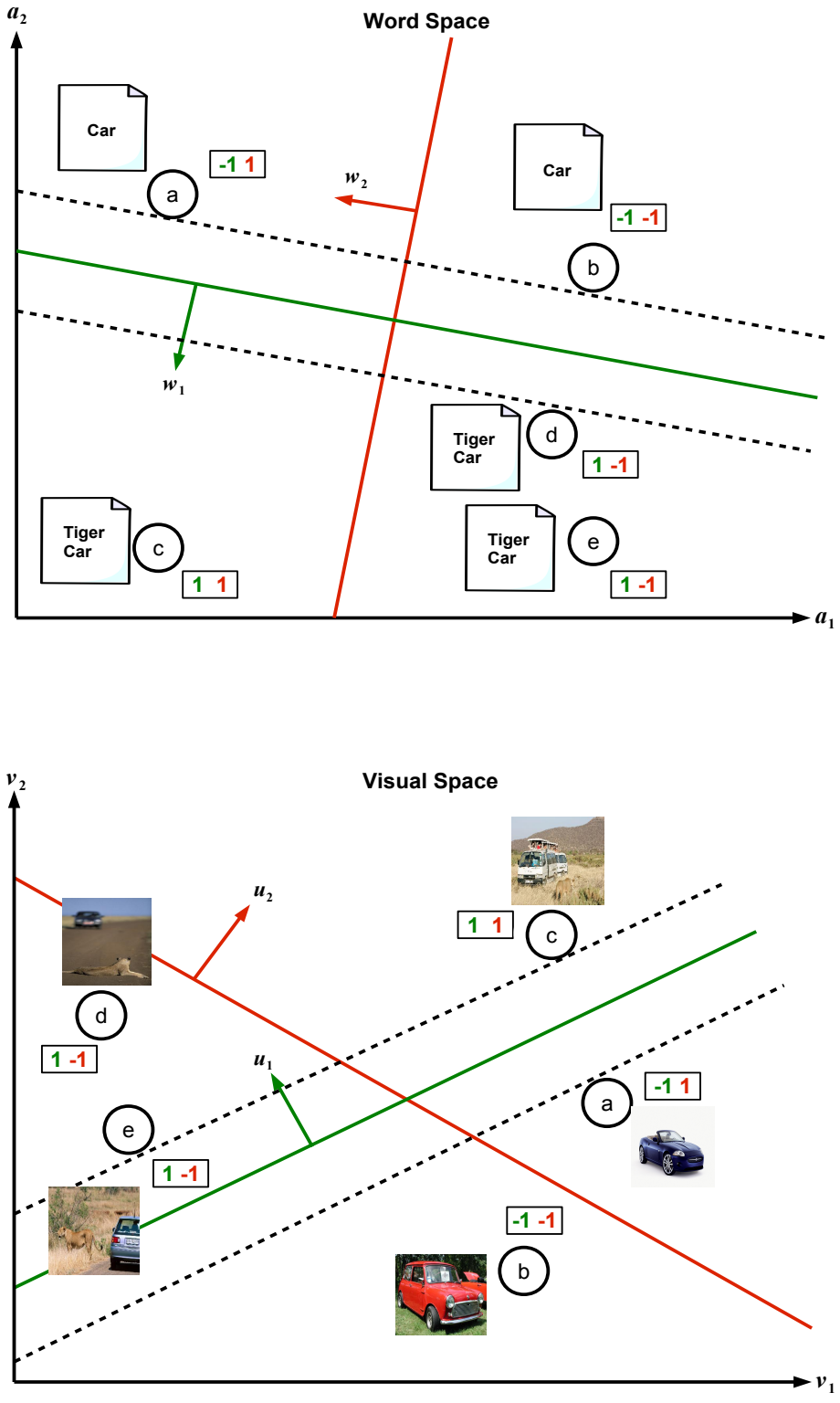


Figure 31: Partitioning step: hyperplanes are learnt in the annotation (top) and visual (bottom) space using annotation bits as labels. Hyperplanes in both feature spaces are positioned in such a way that nodes with the same letter are assigned the same bits in both feature spaces.

6.5.6 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 62

$$S^{x}_{ij} = \begin{cases} 1 & \text{if } \mathbf{x}_i \in NN_k(\mathbf{x}_j) \text{ or } \mathbf{x}_j \in NN_k(\mathbf{x}_i) \\ 0 & otherwise \end{cases} \tag{62}$$

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 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 6.5.1). The IMH objective function is presented in Equation 63

$$\begin{aligned} \text{argmin}_{\mathbf{W},\mathbf{U},\mathbf{Y}^x,\mathbf{Y}^z} \quad & \lambda \sum_{i,j=1}^{N_{trd}} S^{x}_{ij} \|\mathbf{y}^x_i - \mathbf{y}^x_j\|_2^2 + \lambda \sum_{i,j=1}^{N_{trd}} S^{z}_{ij} \|\mathbf{y}^z_i - \mathbf{y}^z_j\|_2^2 + \sum_{i,j=1}^{N_{xz}} S^{xz}_{ij} \|\mathbf{y}^x_i - \mathbf{y}^z_j\|_2^2 \\ & + \sum_{i=1}^{N_{trd}} \|\mathbf{W}^\intercal \mathbf{x}_i - \mathbf{y}^x_i\|_2^2 + \beta \|\mathbf{W}\|_F^2 + \sum_{j=1}^{N_{trd}} \|\mathbf{U}^\intercal \mathbf{z}_j - \mathbf{y}^z_j\|_2^2 + \beta \|\mathbf{U}\|_F^2 \\ & \text{subject to } (\mathbf{Y}^x)^\intercal \mathbf{Y}^x = \mathbf{I}^{D_x \times D_x} \\ & (\mathbf{Y}^x)^\intercal \mathbf{1} = \mathbf{0} \end{aligned} \tag{63}$$

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 extensively on previous research (Weiss et al., 2008; Kumar & Udupa, 2011; Zhen & 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 35) we first saw in the context of Spectral Hashing (SH) in Section 6.3.2 and extended first to the dual-modality case by CVH (Section 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 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 63 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 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 $\mathcal{O}(N_{trd}^3)$ operations[37]. IMH maintains properties E_1, E_2 of an effective hashcode in both modalities $\mathcal{X}$, $\mathcal{Y}$, while only maintaining properties E_3, E_4 in modality $\mathcal{X}$ as a result of solving the eigenvalue problem.

6.5.7 A Brief Summary

In this section we described six prominent hashing algorithms that are capable of indexing similar data-points that exist in two incommensurable feature spaces into the same hashtable buckets. Specifically, we reviewed Cross-View Hashing (CVH) (Section 6.5.1), Co-Regularised Hashing (Section 6.5.2), Cross-Modal Semi-Supervised Hashing (Section 6.5.3), Predictable Dual-View Hashing (Section 6.5.4), Regularised Cross-Modal Hashing (Section 6.5.5) and Inter-Media Hashing (Section 6.5.6). 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 most of these algorithms are their general reliance on either an expensive matrix factorisation or non-convex optimisation.

7. Evaluation Paradigms, Datasets and Performance Metrics

In this section we describe the common experimental methodology adopted throughout the learning-to-hash literature. This includes the datasets selected as the testbed for the retrieval experiments (Section 7.1), the definition of groundtruth for evaluation (Section 7.3) and the metrics adopted to ascertain retrieval effectiveness (Section 7.6).

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

7.1 Datasets

The focus of this review has been learning hash functions for the task of large-scale image retrieval. This task can be split into three sub-tasks: 1) an image is used to retrieve related images from a still image archive, 2) a text query is used to retrieve related images, 3) an image query is used to retrieve relevant annotations for that image. We therefore describe the experimental setup for unimodal and cross-modal retrieval experiments which will cover a wide range of important use-cases in image retrieval from query-by-example search to image annotation. Unimodal datasets are those where the query and the database are in the same visual modality such as bag-of-visual-word feature descriptors. Cross-modal datasets permit retrieval experiments that straddle two different modalities such as a textual query executed against an image database. The latter task mimics the familiar image search scenario offered by many modern web search engines. To align the evaluation closely with the learning to hash literature we select a subset of the most popular datasets from both categories (Sections 7.1.1-7.2). The desiderata for dataset selection is two-fold: firstly, the datasets must be publicly available to enable replication of experimental results by a third party; and secondly the datasets must be standard in the sense that they have been widely used in related publications. This ensures that the experimental results published in this literature are reproducible and directly comparable to previously published research.

7.1.1 Unimodal Retrieval Experiments

For the unimodal experiments, there are six popular and freely available image datasets: LabelMe, CIFAR-10, NUS-WIDE, MNIST, SIFT1M and ImageNet. The datasets are of widely varying size (22,019-1.3 million images), are represented by an array of different feature descriptors (from GIST, SIFT, raw pixels to bag of visual words) and cover a diverse range of different image topics from natural scenes to personal photos, logos and drawings. These properties ensure that the datasets provide a challenging test suite for evaluation. All datasets are identical to those used in recent publications (Kong & Li, 2012a), (Shen et al., 2015), (Liu et al., 2012) and are available online to the research community.

- **LABELME:** 22,019 images represented as 512 dimensional GIST descriptors (Torralba et al., 2008)(Russell, Torralba, Murphy, & Freeman, 2008)[38] The dataset is mean centred.
- **CIFAR-10:** 60,000 32×32 colour images sampled from the 80 million Tiny Images dataset (Krizhevsky & Hinton, 2009). Each image is encoded with a 512 dimensional GIST descriptor (Oliva & Torralba, 2001) and is manually assigned a label from a selection of 10 classes[39]. Each class has 6,000 associated images. The visual feature descriptors are mean centered.
- **NUS-WIDE:** 269,648 images downloaded from Flickr each annotated with multiple ground truth concept tags (e.g. nature, dog, animal, swimming, car) from an 81 concept vocabulary[40] (Chua, Tang, Hong, Li, Luo, & Zheng, 2009). The images

38. http://www.cs.toronto.edu/~norouzi/research/mlh/
39. http://www.cs.toronto.edu/~kriz/cifar.html
40. http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm

Figure 32: The NUS-WIDE dataset consists of around 270,000 images randomly sampled from Flickr. Given the diversity of images (from people, animals, landscapes to buildings and drawings) and widely varying resolution the dataset provides a challenging testbed for image retrieval.

are represented by a 500 dimensional bag-of-visual-words (BoW) feature descriptor formed by vector quantising SIFT descriptors via k-means clustering. The visual feature descriptors are L_2-normalised to unit length and mean centered. For illustrative purposes I show a random sampling of images from the NUS-WIDE dataset in Figure 32.

- **SIFT1M:** 1,000,000 images from Flickr encoded with 128-dimensional SIFT descriptors[41]. This dataset was first introduced by Jegou et al. (Jegou et al., 2011) and has since became a standard image collection for evaluating nearest neighbour search methods (Kong & Li, 2012a), (He et al., 2013), (Wang et al., 2010b). The dataset is mean centred.

- **MNIST:** 70,000 images of handwritten grayscale digits represented as 784 dimensional raw pixel values (Lecun, Bottou, Bengio, & Haffner, 1998).

- **ImageNet:** 1,331,167 images from the ILSVRC2012 challenge. These images are a sub-sample from the larger ImageNet dataset[42]. ImageNet is a large-scale image

41. `http://lear.inrialpes.fr/~jegou/data.php`
42. `http://www.image-net.org/`

Dataset	# images	# labels	Labels/image	Images/label	Descriptor
LABELME	22,019	–	–	–	512-D Gist
CIFAR-10	60,000	10	1	6,000	512-D Gist
NUS-WIDE	269,648	81	1.87	6,220	500-D BoW
SIFT1M	1,000,000	–	–	–	128-D SIFT
MNIST	70,000	–	–	–	784-D raw pixels
ImageNet	1,331,167	1,000	1	1,331	4096-D CNN

Table 4: Salient statistics of the six datasets commonly used in unimodal hashing experiments. The Labels/image and Images/label are the mean values computed on the entire dataset.

collection consisting of 10 million images organised hierarchically according to over 10,000 nouns in WordNet (Deng, Dong, Socher, Li, Li, & Li, 2009). The images in the 1.3 million image subsample are annotated with labels from a 1,000 label vocabulary and are represented by a 4096-dimensional feature descriptor extracted from the penultimate layer of a deep convolutional neural network (CNN) (Krizhevsky et al., 2012). These CNN image feature descriptors are known to produce state-of-the-art performance in many prime Computer Vision tasks including image retrieval (Razavian, Azizpour, Sullivan, & Carlsson, 2014). The visual feature descriptors are L_2-normalised to unit length and mean centered. To the best of our knowledge this is one of the largest labelled image datasets available.

7.2 Cross-Modal Retrieval Experiments

Cross-modal retrieval experiments are typically conducted on the 'Wiki' dataset and NUS-WIDE (Kumar & Udupa, 2011) (Zhen & Yeung, 2012) (Song et al., 2013) (Rastegari et al., 2013) (Bronstein et al., 2010) datasets. Both datasets come with images and associated paired textual descriptors, a key requirement for training and evaluating a cross-modal retrieval model. As for the unimodal retrieval datasets described in Section 7.1.1 these two cross-modal datasets are also freely available to the research community.

- **Wiki:** is generated from 2,866 Wikipedia articles[43] derived from Wikipedia's "feature articles" (Rasiwasia, Costa Pereira, Coviello, Doyle, Lanckriet, Levy, & Vasconcelos, 2010). The featured articles segment of Wikipedia hosts the highest quality articles on the site as judged by a panel of independent Wikipedia editors. Each feature article is a document consisting of multiple sections and annotated with at least one relevant image from the Wikimedia commons. Each article is designated with a manually labelled category out of 29 possibilities. Rasiwasia et al. (Rasiwasia et al., 2010) only keep the articles pertaining to the 10 most populated categories. Each article is further split by section and the image manually placed in that section by the author(s) is used as the corresponding visual description of the text in that section. Any section that ends up without an associated image is discarded. This leaves 2,866 short and focused

43. `http://www.svcl.ucsd.edu/projects/crossmodal/`

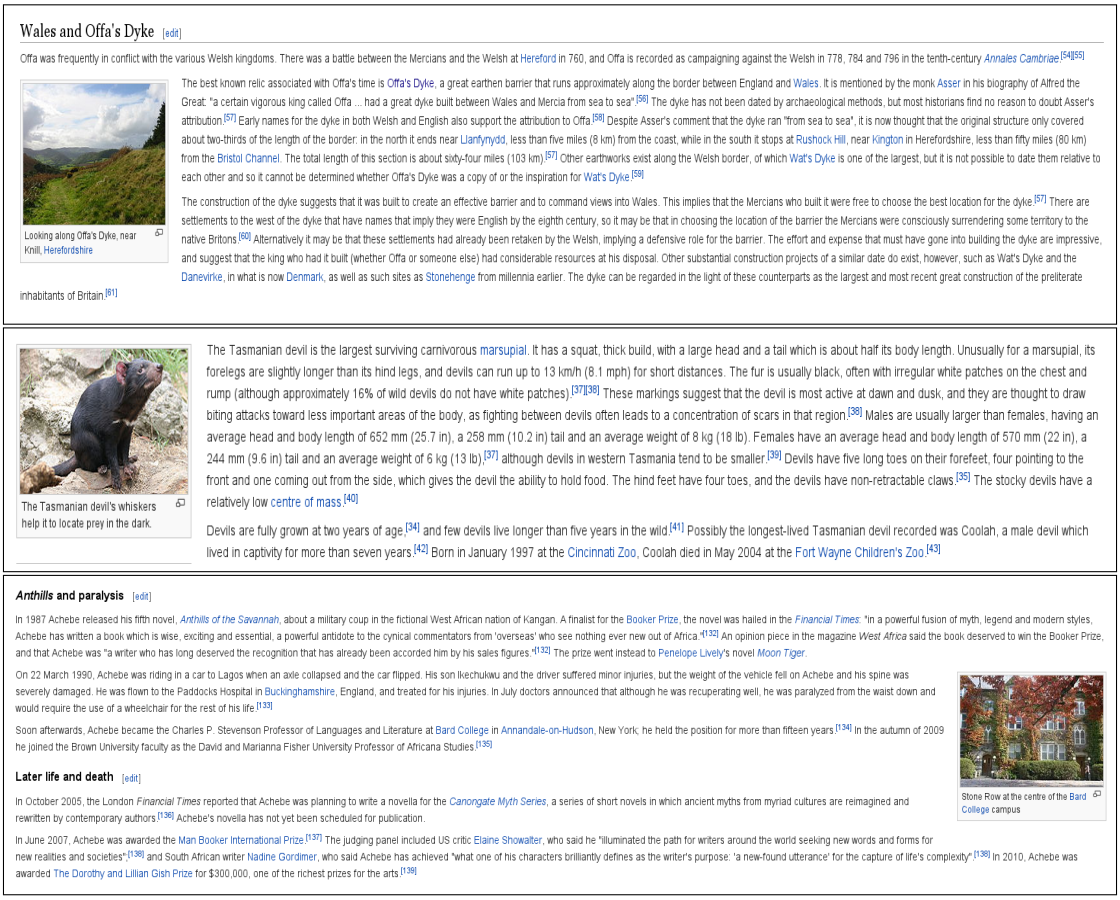


Figure 33: Three example Wikipedia article sections (Offa King of Mercia (`https://en.wikipedia.org/wiki/Offa_of_Mercia`), the Tasmanian devil (`https://en.wikipedia.org/wiki/Tasmanian_devil`) and a description from the life of Nigerian novelist Chinua Achebe (`https://en.wikipedia.org/wiki/Chinua_Achebe`) and their aligned images taken from the cross-modal Wiki dataset.

"articles" of a median length of 200 words, with each article having at least 70 words. We use the image and text feature set provided by Rasiwasia et al. (Rasiwasia et al., 2010) which is used in most related cross-modal hashing research (Zhen & Yeung, 2012). The visual modality is represented as a 128-dimensional SIFT (Lowe, 2004) bag-of-words histogram, while the textual modality is represented as 10-dimensional probability distribution over Latent Dirichlet Allocation (LDA) topics (Blei, Ng, & Jordan, 2003). Three example Wikipedia article sections and their associated images are shown in Figure 33.

- **NUS-WIDE:** is identical to the unprocessed NUS-WIDE dataset described in Section 7.1.1 (Chua et al., 2009). For cross-modal experiments this dataset is pre-processed in a different manner to the strategy described in Section 7.1.1 (Zhen & Yeung, 2012). More specifically, for cross-modal retrieval the multiple image tags associated with an

image are used to define the textual modality. The image-text pairs associated with the most frequent 10 classes are retained. Each image is associated with a subset of 5,018 tags manually assigned by Flickr users. A PCA dimensionality reduction is performed on the 269,648×5018 dimensional tag co-occurrence matrix to form a 1,000-dimensional tag feature set. This projected tag feature set is then mean-centered and used as a representation of the textual modality. This is a standard pre-processing step in the literature (Zhen & Yeung, 2012). The visual modality is represented by the same 500 dimensional bag-of-words (BoW) feature descriptors described in the context of NUS-WIDE in Section 7.1.1. The visual descriptors are L_2-normalised to unit length and mean centered.

7.3 Nearest Neighbour Groundtruth Definition

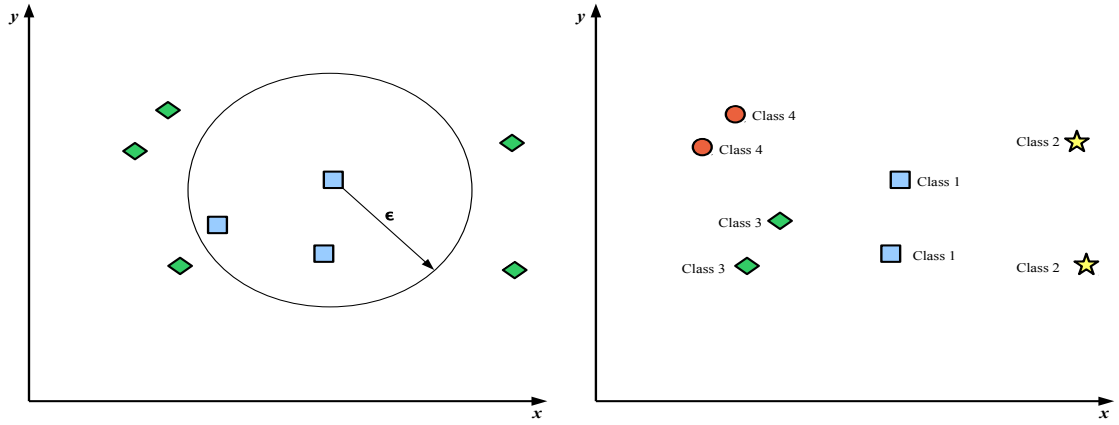
In order to evaluate the retrieval effectiveness of a hashing model we need to define which data-points in the database are considered to be nearest neighbours of the query data-points. We refer to these data-points as the *true* nearest neighbours of a query. Typically the system is penalised depending on the degree to which it fails to return the true nearest neighbours for a query. The definition of the groundtruth nearest neighbours varies widely between publications. In this review we consider two of the strategies commonly used to define groundtruth which involves either constructing an ϵ-ball around the query data-point (Section 7.3.1) or using human assigned class-labels (Section 7.3.2). To the best of our knowledge, there has been no work to verify whether or not the ϵ-ball groundtruth definition correlates with user search satisfaction. We discuss this point further in Section 8 as part of possible future work.

7.3.1 ϵ-Ball Nearest Neighbours

The ϵ-nearest neighbour (ϵ-NN) groundtruth definition is a popular way to evaluate hashing models (Figure 34a)[44]. In this paradigm a ball of radius ϵ is defined around a query data-point in the input feature space and the true nearest neighbours are defined as those data-points enclosed within the ball. To compute the ϵ-NN groundtruth, previous related work (Kong et al., 2012)(Kong & Li, 2012a)(Kulis & Darrell, 2009)(Gong & Lazebnik, 2011) randomly sample 100 data-points from the training dataset to compute the Euclidean distance at which each data-point has R nearest neighbours on average. The ϵ-ball radius is then set to equal this average distance. The parameter R is set to 50 nearest neighbours in the literature (Kong et al., 2012) (Kong & Li, 2012a) (Kong & Li, 2012b) (Kulis & Darrell, 2009) (Raginsky & Lazebnik, 2009) (Gong & Lazebnik, 2011). The groundtruth matrix $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$ is then derived by computing the Euclidean distance $\mathbf{D} \in \mathbb{R}^{N_{trd} \times N_{trd}}$ between a small subset of the data-points ($N_{trd} \ll N$) and thresholding the distances by ϵ. This method of groundtruth generation is presented in Equation 64 and Figure 34a.

$$\mathbf{S} = \begin{cases} S_{ij} = 1, & \text{if} \quad D_{ij} \leq \epsilon \\ S_{ij} = 0, & \text{if} \quad D_{ij} > \epsilon \end{cases} \tag{64}$$

44. Defining ground-truth nearest neighbours can also be achieved by computing a k-NN graph. In this case related data-points to a query are those that have the k smallest distances to the query.



(a) **ϵ-nearest neighbour ground truth** (b) **Class based groundtruth**

Figure 34: Two definitions of groundtruth nearest neighbours (NN). In Figure (a) we show how to define nearest neighbours of a data-point using an ϵ-ball. All data-points enclosed by the ball are nearest neighbours of the data-point at the center. In Figure (b) we show a class-based definition of nearest neighbours. Nearest neighbours are defined as those data-points sharing at least one class label in common.

7.3.2 Class-Based Nearest Neighbours

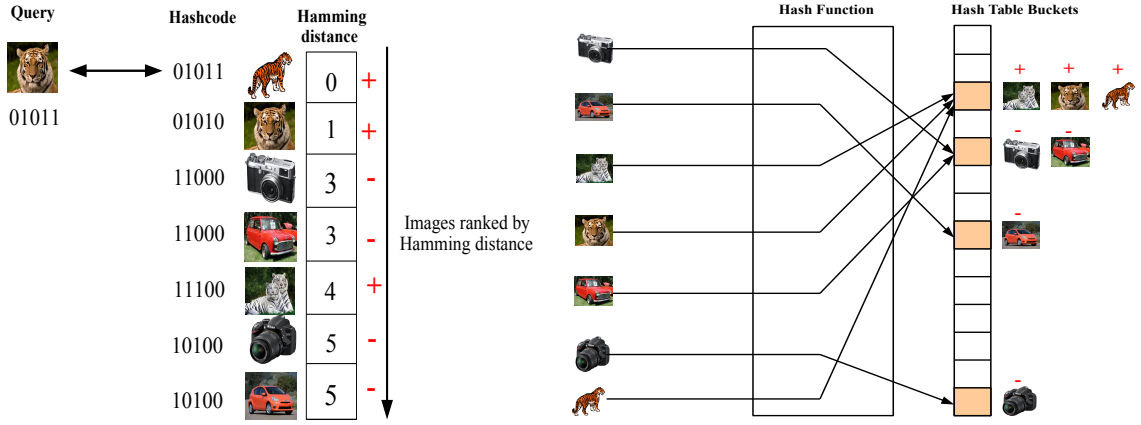
Class label-based groundtruth is often used in the literature (Figure 34b) as an alternative or a complementary groundtruth to the ϵ-NN evaluation paradigm. Class labels are also crucial for cross-modal retrieval experiments where it is not possible to directly compute the Euclidean distance between two feature vectors of a different type. In this scenario $S_{ij} = 1$ for an element of the groundtruth matrix $\mathbf{S}$ if the corresponding pair of data-points $\mathbf{x}_i, \mathbf{x}_j$ share *at least one* class label or annotation in common, and $S_{ij} = 0$ otherwise (Gong & Lazebnik, 2011; Liu et al., 2012).

7.4 Evaluation Paradigms

There are two main paradigms for evaluating hashing models: the *Hamming ranking* evaluation paradigm (Section 7.4.1) and the *hashtable bucket* evaluation paradigm (Section 7.4.2). Both paradigms are illustrated in Figure 35. The Hamming ranking paradigm is standard within the learning to hash research literature while the hashtable bucket evaluation paradigm is frequently used in practical hashing applications in which a fast query-time is of prime importance. We introduce both paradigms in Sections 7.4.1-7.4.2 before explaining why the Hamming ranking evaluation paradigm is typically employed throughout most of the learning-to-hash literature (Section 7.4.3).

7.4.1 Hamming Ranking Evaluation

Most existing research (Kong et al., 2012) (Kong & Li, 2012a) (Kong & Li, 2012b) (Liu et al., 2012) (Liu et al., 2011) (Gong & Lazebnik, 2011) (Zhang et al., 2010) (Kulis & Grauman, 2009) evaluate retrieval effectiveness using the widely accepted *Hamming ranking evaluation*



(a) **Hamming ranking evaluation** (b) **Hashtable bucket evaluation**

Figure 35: Two evaluation paradigms for hashing. In both cases relevant images are those depicting similar objects. In Figure (a) the Hamming distance between the query hashcode and the database images is computed. The images are ranked and the resulting ranked list used to compute a retrieval metric such as average precision (AP). In this case we find an $AP = 0.87$. The average precision scores are aggregated across queries by computing the mean average precision (mAP). In Figure (b) we show the hashtable evaluation strategy. Each image is hashed to a bucket. A count is then made of the number of true positives (TPs), false positives (FPs) and false negatives (FNs) colliding in the same buckets. In this toy example, $TP = 3$, $FP = 1$, $FN = 2$ which equates to a micro-average F_1-measure of 0.78. On both diagrams $+$ indicates a true positive while a $-$ indicates a false positive/negative.

paradigm. In this evaluation paradigm, binary hashcodes are generated for both the query and the database images. The Hamming distance is then computed from the query images to all of the database images, with the database dataset images ranked in ascending order of the Hamming distance. The resulting ranked lists are then used to compute retrieval evaluation metrics such as area under the precision recall curve (AUPRC) (Section 7.6.3) and mean average precision (mAP) (Section 7.6.4). The Hamming ranking evaluation paradigm is a proxy for evaluating hashing accuracy over the range of user preferences (precision/recall) and without having to specify the parameters (K, L) of a specific hashtable implementation. We discuss this latter point further in Section 7.4.3.

7.4.2 Hashtable Bucket-Based Evaluation

The Hamming ranking evaluation paradigm is by definition of $\mathcal{O}(N)$ time complexity for a single query data-point. The *hash bucket-based evaluation* has a constant $\mathcal{O}(1)$ search time independent of the dataset size. A hashtable lookup evaluation is much closer to how the hashing models would be used in a real-world application where a fast query time is a necessity. Despite this fact a hashtable evaluation is rarely reported in the learning to hash literature with the Hamming ranking paradigm being the preferred evaluation methodology. We previously described the application of hashtables in the context of Locality Sensitive

Hashing (LSH) (Section 4). In this evaluation paradigm the hashcodes are generated for the query and database points which are then used as the indices into the buckets of L hashtables. The union is then taken over all the data-points that collide in the same buckets as the query across the L hashtables. The set of data-points thus formed can then be used to compute the effectiveness metrics of precision, recall and F_β-measure. We describe these metrics in more detail in Section 7.6, but the intuition is that we want to reward the algorithm if it returns many true nearest neighbours to the query in the retrieved set while penalising it for returning unrelated data-points. As we examine all colliding data-points for a query we are therefore using the second hashtable query strategy which was discussed in the description of LSH in Section 4.

7.4.3 Hamming Ranking versus Hashtable Bucket Evaluation

The hashtable bucket evaluation paradigm is heavily dependent on both the particular hashtable implementation (i.e. values of K, L, whether or not chaining is used, etc) and on the end application itself, for example is the hashtable on a drone and therefore do we have limited available main memory? If we opt for a hashtable evaluation paradigm we either need to pick a default setting of K and L and tie our evaluation to this specific hashtable implementation, or alternatively we can measure the hashing model performance over many different values of the hashtable parameters leading to an explosion in the number of results to be reported. To abstract away from the specifics of a particular hashtable implementation and to obtain a single number summarising the quality of the hashcodes, researchers in the Computer Vision literature prefer to evaluate their hashing models by Hamming ranking which involves computing the *Hamming distance* between the hashcodes, rather than using a hashtable-based setup (Gong & Lazebnik, 2011) (Liu et al., 2011). The Hamming distance given in Equation 4 (Section 3) measures the number of bits that are different between two hashcodes and is therefore likely to be a good indicator of the quality of a hashtable lookup using those hashcodes. The more bits in common the greater the likelihood of a collision between the corresponding data-points.

There is an interesting, but not immediately obvious link between the Hamming ranking evaluation paradigm and the hashtable evaluation paradigm[45]. Measuring the quality of a set of ranked lists using mAP and AUPRC is effectively acting as a *proxy* for *many* different settings of K and L in a corresponding hashtable evaluation. To confirm this fact, we make reference to Figure 36 in which we show five hashcodes ranked in ascending order of Hamming distance from the query (marked in the diagram in bold font). Observe that each threshold effectively defines a set of "colliding" data-points, that is those above the threshold with the lowest Hamming distance to the query. The thresholded ranked list therefore corresponds to settings of K and L that ensure the data-points above the threshold will collide in at least one hypothetical hashtable. The L hashtables in Figure 36 are formed by splitting the hashcodes into L K-bit segments, with each K-bit segment indexing into a specific bucket of one of the L hashtables. Just as choosing a particular setting of K and L is application specific so is choosing a particular threshold in the Hamming ranking evaluation paradigm. Usefully the mAP and AUPRC provide a single number measure of ranking quality that is computed by aggregating across many different settings of the

45. This observation arose from a discussion with Victor Lavrenko.

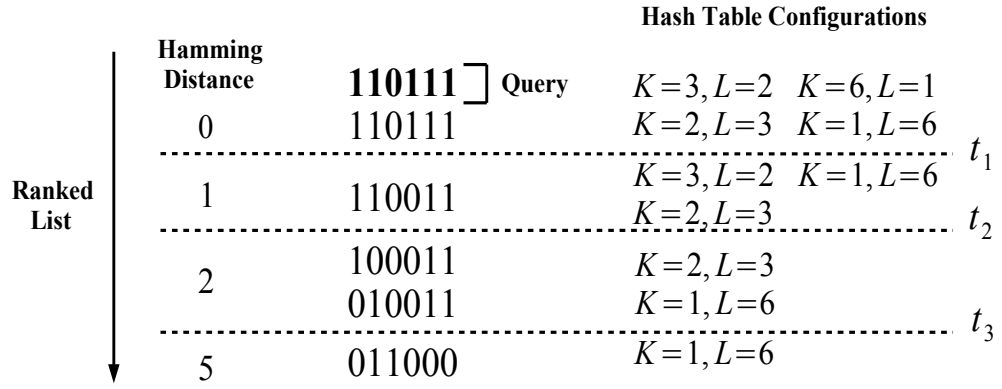


Figure 36: An analogy between the Hamming ranking evaluation paradigm and the hashtable bucket evaluation paradigm. We rank five hashcodes in ascending order by Hamming distance from the query hashcode (shown here in bold). The ranked list is thresholded at *three* different points (t_1, t_2, t_3), with the thresholds indicated by the dashed horizontal lines. The hashcodes above the threshold can be considered to be "colliding" with the query hashcode. The settings of K and L that would cause the collision are shown for the four different Hamming distances. For example, for the *bottom most* thresholding splitting the hashcodes into $L = 6$ segments of $K = 1$ bits will cause the hashcode at Hamming distance 5 to collide in the same bucket as the hashcodes at Hamming distance 2, 1 and 0. In this way we see that a particular thresholding of a ranked list of hashcodes is equivalent to many different settings of K and L in a hashtable evaluation.

ranked list threshold, and consequently many different values of K and L. The Hamming ranking evaluation paradigm is therefore a more general evaluation strategy for hashing that is able to measure the overall quality of hashcodes without being tied to a particular end-application. In effect it indicates how good the hashing-based ANN search would be if we found the best setting of K and L in a hashtable bucket evaluation.

7.5 Constructing Random Dataset Splits

In this section we describe how the datasets introduced in Section 7.1 are partitioned to form testing and validation queries and training and database splits for the purposes of learning and evaluating the hash functions. Two strategies for forming splits will be described: in Section 7.5.1 we describe the literature standard strategy that is widely used by the research community, while in Section 7.5.2 we describe the proposed splitting methodology that seeks to remedy concerns with the accepted evaluation strategy. In both cases, when class-based ground-truth is used, it is common to sample the splits so as to obtain a balanced distribution of classes within each partition.

7.5.1 Literature Standard Splits

In previous work *repeated random subsampling* cross-validation over ten independent runs is used to evaluate the quality of the learnt hash functions (Liu et al., 2012) (Kong et al., 2012) (Kong & Li, 2012a) (Liu et al., 2014) (Wang et al., 2012). Figure 37 shows an example of a

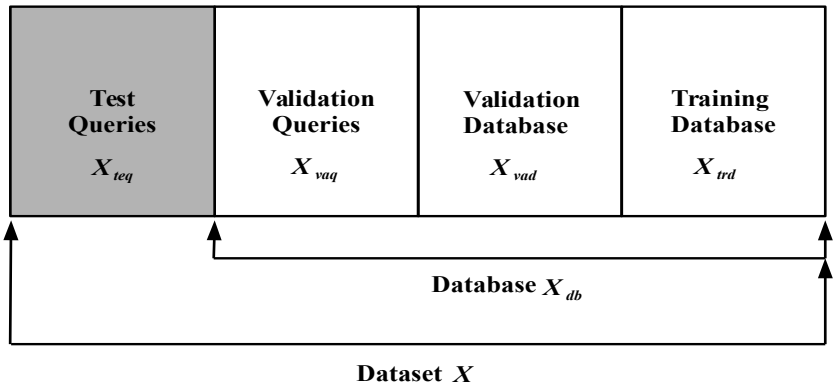


Figure 37: The literature standard dataset splitting procedure. The standard procedure used in the literature for splitting a dataset into testing and training partitions. The entire dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$ is represented as the concatenation of the individual rectangles, each of which highlights a particular partition. The rectangle in grey represents the split of the dataset that is held-out and only used once for computing the final measure of retrieval effectiveness.

random dataset split for one run. The entire dataset is denoted as $\mathbf{X} \in \mathbb{R}^{N \times D}$. This dataset is divided into a held-out set of test queries $\mathbf{X}_{teq} \in \mathbb{R}^{N_{teq} \times D}$ and a database split $\mathbf{X}_{db} \in \mathbb{R}^{N_{db} \times D}$. The test queries are used once when we come to compute the evaluation metric by ranking the database split (Section 7.4.1). The database split also doubles as the training dataset for learning the hash functions. The best setting of model *hyperparameters*[46] is found by grid search on the validation split of the dataset. In practice this grid search is conducted by running a set of validation queries $\mathbf{X}_{vaq} \in \mathbb{R}^{N_{vaq} \times D}$ against a validation database $\mathbf{X}_{vad} \in \mathbb{R}^{N_{vad} \times D}$, both of which are sampled from the database $\mathbf{X}_{db}$.This can be considered as a form of nested cross-validation in which the optimal hyperparameters are determined for each run. The training database $\mathbf{X}_{trd} \in \mathbb{R}^{N_{trd} \times D}$ is used to learn the *parameters* (hyperplanes, quantisation thresholds) of the hash functions and is itself a subset of $\mathbf{X}_{db}$. In the remainder of this review we refer to this splitting strategy as the *literature standard splitting strategy*. We illustrate this method of forming dataset splits in Figure 37.

7.5.2 Improved Splitting Strategy

Unfortunately, there is a potential overfitting concern with the standard dataset splitting strategy described in Section 7.5.1 given that the database points which are ranked or indexed with respect to the test queries are also used as the training dataset for learning the hash functions themselves. Ideally there should be a clean separation between the split of the dataset that is used to learn the hash functions and the split of the dataset that is ranked/indexed in order to compute the final measure of retrieval effectiveness. This ensures that we can evaluate the true generalisation performance of the hash functions when there is not only unseen queries but also an unseen database that is to be ranked/indexed with

46. Hyperparameters are parameters *other* than the hashing hyperplanes or quantisation thresholds. Example of hyperparameters are the flexibility of margin C for the SVM and the kernel bandwidth parameter γ for the RBF kernel.

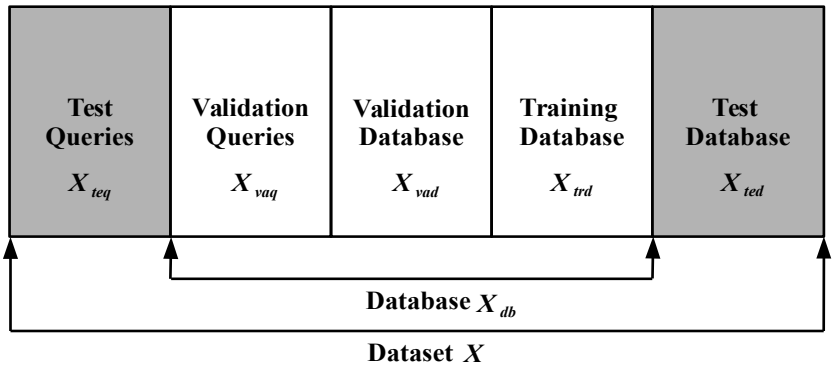


Figure 38: My improved dataset splitting procedure. My proposed splitting strategy for overcoming the overfitting concern with the literature standard strategy. In addition to the test queries we also advocate holding out a split of the dataset to act as the testing database. The held-out splits of the dataset are shown in grey. At test time the testing queries are used to retrieve related items from the testing database. This retrieval run is used to compute the final measure of effectiveness for determining the quality of the hash functions.

respect to those queries. Currently the literature is only concerned with the generalisation performance with respect to unseen query data-points and where the database is known a-priori and can be used for hash function learning. To the best of our knowledge this review is the first in the literature to note this technical flaw in the standard method for forming dataset splits. To mitigate this overfitting concern we propose a new method for generating splits of the dataset. In this new strategy we again perform repeated random subsampling cross-validation over ten runs. However, the makeup of a random split for a run now differs from the literature standard splitting strategy. In our suggested dataset splitting strategy we divide the dataset into *five* splits as shown in Figure 38. We have a set of held-out test queries $\mathbf{X}_{teq} \in \mathbb{R}^{N_{teq} \times D}$ and also a *held-out test database* $\mathbf{X}_{ted} \in \mathbb{R}^{N_{ted} \times D}$ against which those test queries are run. Both of the test queries and test database are only used once when we come to compute the final retrieval effectiveness metric for that particular run. The remainder of the dataset forms the database split $\mathbf{X}_{db} \in \mathbb{R}^{N_{db} \times D}$ which is used for setting the parameters and hyperparameters of the hashing models. The database split is further divided into a set of validation queries $\mathbf{X}_{vaq} \in \mathbb{R}^{N_{vaq} \times D}$, a validation database split $\mathbf{X}_{vad} \in \mathbb{R}^{N_{vad} \times D}$ which is ranked/indexed against the validation queries and a training split that is used to learn the hash functions $\mathbf{X}_{trd} \in \mathbb{R}^{N_{trd} \times D}$. We refer to this splitting strategy as the *improved splitting strategy*.

7.6 Evaluation Metrics

We follow previous research in the learning to hash literature and judge the retrieval effectiveness by the standard Information Retrieval (IR) metrics of *precision*, *recall*, F_β*-measure* (Section 7.6.1), *area under the precision recall curve (AUPRC)* (Section 7.6.3) and *mean average precision (mAP)* (Section 7.6.4).

7.6.1 Precision, Recall, F_β-Measure

Precision, *recall* and their harmonic mean, the *F_β-measure*, are set-based evaluation metrics that can be used to ascertain the quality of an unranked collection of images. The retrieved set can be determined by looking into the colliding hashtable buckets for the Hashtable bucket evaluation or by defining a Hamming radius threshold for the Hamming ranking evaluation. In terms of the Hamming ranking evaluation, precision and recall can be computed by counting the number of true nearest neighbours that are within a fixed Hamming radius (true positives, TPs), the number of non nearest neighbours that are within a fixed Hamming radius (false positives, FPs) and the number of related data-points that are *not* are within a fixed Hamming radius to the query (false negatives, FNs).

More formally, we denote the groundtruth matrix as $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$ (Sections 7.3.1-7.3.2). The groundtruth adjacency matrix specifies which data-points are *true nearest neighbour* pairs ($S_{ij} = 1$) and which data-point pairs are unrelated ($S_{ij} = 0$). As we discussed in Section 7.3, in the context of hashing-based ANN search, a data-point $\mathbf{x}_j$ is denoted as a true nearest neighbour ($S_{ij} = 1$) if it is within an ϵ-ball of the query data-point $\mathbf{q}_i$ or shares at least one class label in common with the query. Following a retrieval run, the ranked data-points within a certain Hamming radius (D) of the query are those data-points considered to be related to the query, while those data-points outside of the Hamming radius D are considered to be unrelated[47]. The results of a ranked retrieval for a certain Hamming distance threshold D are represented by the square matrix $\mathbf{R} \in \{0,1\}^{N \times N}$ given in Equation 65.

$$R_{ij} = \begin{cases} 1, & \text{if} \quad \mathbf{x}_j \quad \text{is within Hamming radius D to the query} \quad \mathbf{q}_i \\ 0, & \text{otherwise.} \end{cases} \tag{65}$$

Given the definitions of $\mathbf{S}$ and $\mathbf{R}$, the number of *true positives* for a single query data-point $\mathbf{q}_i$ is defined in Equation 66

$$TP(\mathbf{q}_i) = \sum_j S_{ij} \cdot R_{ij} \tag{66}$$

A *false negative (FN)* is a true nearest neighbour ($S_{ij} = 1$) that is outside of the Hamming radius around the query $\mathbf{q}_i \in \mathbb{R}^D$. The total false negative count for the query is given in Equation 67

$$FN(\mathbf{q}_i) = \sum_j S_{ij} - TP(\mathbf{q}_i) \tag{67}$$

A *false positive (FP)* is a non-nearest neighbour ($S_{ij} = 0$) that falls within the query Hamming radius $\mathbf{q}_i \in \mathbb{R}^D$ (Equation 68)

$$FP(\mathbf{q}_i) = \sum_j (1 - S_{ij}) \cdot R_{ij} \tag{68}$$

47. This is the Hamming ranking evaluation paradigm discussed in Section 7.4.1.

Given Equations 66-68 the precision and recall metrics can then be defined as in Equations 69-70

$$P(\mathbf{q}_i) = \frac{TP(\mathbf{q}_i)}{TP(\mathbf{q}_i) + FP(\mathbf{q}_i)} \tag{69}$$

Precision is therefore the fraction of true nearest neighbours that are within the fixed Hamming radius out of all data-points that are within the fixed Hamming radius to the query data-point

$$R(\mathbf{q}_i) = \frac{TP(\mathbf{q}_i)}{TP(\mathbf{q}_i) + FN(\mathbf{q}_i)} \tag{70}$$

Recall is then the fraction of true nearest neighbours that are within the fixed Hamming radius to the query out of all possible true nearest neighbours for that query, regardless whether or not they are within the specified Hamming radius.

In a typical image retrieval experiment we have more than one query data-point. The question arises as to how we aggregate the precision and recall scores for all Q queries. There are effectively two ways which involve either taking a *micro-average* or a *macro-average*. For example, the *micro-average* sums the TPs, FPs and FNs across all queries before computing the total precision and recall (Equations 71-72).

$$P_{micro} = \frac{\sum_{i=1}^{Q} TP(\mathbf{q}_i)}{\sum_{i=1}^{Q} TP(\mathbf{q}_i) + \sum_{i=1}^{Q} FP(\mathbf{q}_i)} \tag{71}$$

$$R_{micro} = \frac{\sum_{i=1}^{Q} TP(\mathbf{q}_i)}{\sum_{i=1}^{Q} TP(\mathbf{q}_i) + \sum_{i=1}^{Q} FN(\mathbf{q}_i)} \tag{72}$$

The weighted harmonic mean of recall and precision is known as the F_β-measure and is presented in Equation 73 (Rijsbergen, 1979):

$$\begin{aligned} F_\beta = & \frac{(1+\beta^2) P_{micro} R_{micro}}{\beta^2 P_{micro} + R_{micro}} \\ = & \frac{(1+\beta^2) TP_{micro}}{(1+\beta^2) TP_{micro} + \beta^2 FN_{micro} + FP_{micro}} \end{aligned} \tag{73}$$

F_β-measure can be used to combine precision and recall resulting from both a macro or micro-average. The free parameter $\beta \in \mathbb{R}_+$ is used to adjust the contribution from the precision and recall. Setting $\beta < 1$ in Equation 73 weights precision higher than recall, and vice-versa for a setting of $\beta > 1$. In most applications β is set to 1.0 giving the commonly used F_1-measure that provides an equal balance between the contribution of precision and recall to the final score. The greater the F_β-measure the more effective are the hash functions at returning true nearest neighbours in the same hashtable buckets.

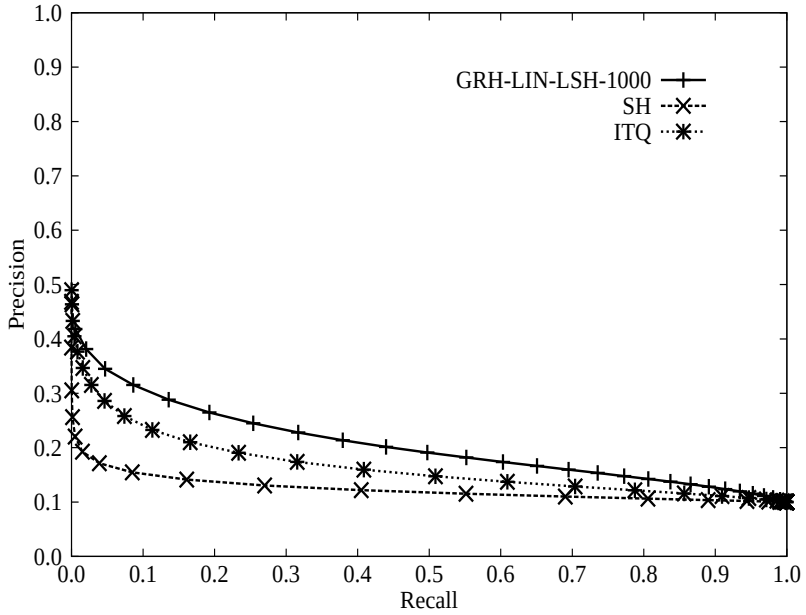


Figure 39: Precision recall curves for the CIFAR-10 dataset for a hashcode length of 32 bits. The three hashing algorithms indicated on this graph are GRH (Section 6.4.5), SH (Section 6.3.2) and ITQ (Section 6.3.3).

7.6.2 Precision Recall Curve (PR Curve)

The precision and recall set-based evaluation metrics discussed in Section 7.6.1 are computed at a fixed operating point of the hashing algorithm. This operating point is usually derived from a particular parameter setting that is itself driven by user or system constraints. For example, in the context of a hashtable bucket evaluation paradigm (Section 7.4.2) this threshold could be implicitly defined by varying the number of hashtables L and the number of hashcode bits K. For the Hamming ranking evaluation paradigm (Section 7.4.1) the threshold is the radius of the Hamming ball around the queries. Database points with a Hamming distance to the query that puts them outside of the radius are not considered part of the retrieved set and therefore do not contribute to the computation of the precision and recall metrics. In contrast to the set-based evaluation metrics, the *precision-recall (PR) curve* measures the effectiveness of a ranked list of items across a range of different operating points. For the Hamming ranking evaluation paradigm, the PR curve is constructed by finding all the data-points within a certain Hamming radius D of the query set and computing the precision and recall over the corresponding retrieved set. By varying the Hamming radius from unity to the maximum Hamming radius D_{max} exhibited by database hashcodes we can trace out a PR curve using the resulting D_{max} precision-recall values. This curve depicts the trade-off between precision and recall as the Hamming radius from the queries is gradually increased. We expect that as the Hamming radius is increased the precision will drop (as more non-relevant data-points are encountered) while the recall will increase (as more relevant data-points are retrieved). An example precision-recall curve is presented in Figure 39.

7.6.3 Area Under the Precision Recall Curve

In many situations a single number summarising the ranking effectiveness captured by the precision-recall curve is required. Given its wide application in previously related research (Kong et al., 2012) (Kong & Li, 2012a) (Kong & Li, 2012b) (Moran et al., 2013a) (Moran et al., 2013b) the *area under the precision-recall curve (AUPRC)* is one of the main single number effectiveness metrics used consistently throughout the field. The AUPRC is a real-valued number constrained to be within the limits of 0 and 1 and provides a summary of the retrieval effectiveness across all levels of recall. The computation of AUPRC is defined in Equation 74.

$$\begin{aligned} AUPRC = & \int_0^1 P(R)dR \\ = & \sum_{d=1}^{D_{max}} P(d)\delta R(d) \end{aligned} \tag{74}$$

where $P(R)$ denotes the micro precision at micro recall R, $P(d)$ is the precision at Hamming radius d and $\delta R(d)$ is the change in micro recall between Hamming radius $d-1$ and d[48]. The greater the area under the PR curve (AUPRC) the higher the retrieval effectiveness of the associated hashing model. The ideal PR curve has a precision of 1.0 across all recall levels leading to an AUPRC of 1.0.

7.6.4 Mean Average Precision (mAP)

Mean average precision (mAP) is also a commonly applied single-number evaluation metric for summarising the effectiveness of a ranking. However, in contrast to AUPRC which is directly computed from the precision-recall curve, mAP is calculated from the Q ranked lists that are obtained by computing the Hamming distance from every query data-point $\left\{\mathbf{q}_i \in \mathbb{R}^D\right\}_{i=1}^{Q}$ to all the database data-points $\left\{\mathbf{x}_j \in \mathbb{R}^D\right\}_{j=1}^{N}$. Given a set of Q ranked lists, mAP is defined as follows (Wu, Yang, Zheng, Wang, & Wang, 2015): denote as L the number of true nearest neighbours for query $\mathbf{q}$ among the retrieved data-points, $P_{\mathbf{q}}(r)$ as the precision for query data-point $\mathbf{q}$ when the top r data-points are returned, and $\delta(r)$ as an indicator function which returns '1' when the r^{th} data-point is a true nearest neighbour of the query and '0' otherwise. The average precision (AP) for a single query $\mathbf{q}$ is then given in Equation 75 while the average of this quantity across all Q queries, the mean average precision or mAP, is defined in Equation 76.

$$AP(\mathbf{q}) = \frac{1}{L}\sum_{r=1}^{R} P_{\mathbf{q}}(r)\delta(r) \tag{75}$$

$$mAP = \frac{1}{|Q|}\sum_{i=1}^{Q} AP(\mathbf{q}_i) \tag{76}$$

48. The finite sum representation for the AUPRC can be computed using the trapezoidal rule. This is implemented as the `trapz` function in Matlab.

Equation 75 computes the precision at each point when a new relevant image is retrieved. The average precision (AP) for a single query $\mathbf{q}$ is then the mean of these precision values. The mAP is then computed by simply taking the mean of the average precisions across all Q queries (Equation 76). mAP is a real-valued number between 0.0 and 1.0, with a higher number indicating a more effective ranked retrieval and favours relevant images retrieved at higher (better) ranks. mAP is frequently used as a single-number evaluation metric in certain sub-fields of the learning to hash literature, particularly supervised and unsupervised data-dependent projection (Liu et al., 2011) (Liu et al., 2012) (Gong & Lazebnik, 2011) (Zhang et al., 2010).

7.6.5 Comparing and Contrasting AUPRC and mAP

The application of AUPRC and mAP as an evaluation metric is not consistent across the learning to hash literature, with some sub-fields (particularly binary quantisation) favouring AUPRC while others (such as data-dependent projection) appear to favour mAP. It is well-known that mAP is approximately the average of the AUPRC for a set of queries (Turpin & Scholer, 2006) so it is interesting to briefly consider here the retrieval scenarios where both metrics are expected to be in agreement and when they are likely to differ.

AUPRC is a *micro-average* in which the individual true positives, false positives and false negatives are aggregated across all Q queries for a specific threshold. The total aggregated counts are then used to compute the precision and recall for each possible setting of the threshold. The resulting precision and recall values can then be used to compute the AUPRC as given by Equation 74. In contrast the mAP is a *macro-average* which is found by computing the true positives, false positives and resulting precision per query, per relevant document retrieved and then averaging those precision values across all Q queries (Equations 75-76).

In practice, differences between the mAP and AUPRC will only arise in retrieval applications in which the distribution of relevant documents across queries is skewed. In this scenario the AUPRC will favour models that return more relevant documents from the queries with a larger number of relevant documents to the detriment of those queries that have a smaller number of relevant documents. In contrast the mAP will weight the contribution of every query equally even if many documents are relevant to some queries and very few to other queries. This equal weighting of queries ensures that mAP is insensitive to the performance variation between those queries that have many relevant documents and other queries that have very few relevant documents. To achieve a high mAP score the system must aim to do well across all queries and not just those with many relevant documents.

In a practical scenario, where the distribution of relevant documents per query is highly imbalanced, the choice of summarising the ranking effectiveness with either mAP or AUPRC is application specific (Sebastiani, 2002). In some cases we may be primarily interested in high effectiveness for the queries with a greater number of relevant documents (AUPRC). This may be appropriate for evaluating system orientated tasks in which we wish to quantify how well the system does as a whole in returning pairs of true nearest neighbours (e.g. plagiarism detection). In other cases we may be equally interested in queries with a much smaller number of true positives (mAP). The latter scenario may arise in a user evaluation

situation such as web search where the information retrieval system must not be seen to prioritise retrieval effectiveness for one user over another.

7.7 Summary

In this section we introduced the evaluation methodology that is commonly employed in the related research literature. We began in Section 7.1 by outlining a collection of image and document datasets that are commonly used for nearest neighbour (NN) search experiments. The datasets were divided into unimodal (image only) and cross-modal datasets (image-document), and were shown to encompass a large variability in the feature descriptors used to encode the images and documents, as well as the type of objects depicted in the images, their resolution and the total number of images (from 22,019 up to 1.3 million images) per dataset.

The definition of groundtruth is an important facet of any experimental methodology. In Section 7.3, we introduced two main strategies for judging the quality of a nearest neighbour search algorithm. The first strategy constructs a ball of radius ϵ around a query and any data-points falling within that radius are deemed true nearest neighbours (Section 7.3.1). The second strategy sets true nearest neighbours to be those data-points that share at least one class label in common with the query (Section 7.3.2). The latter groundtruth definition is required for cross-modal retrieval experiments in which the feature descriptors occupy incommensurate feature spaces making an ϵ-NN evaluation impractical.

In Section 7.4, we then defined the nearest neighbour search strategy to be used in evaluating the quality of the hashcodes. One natural option is to index the database and query images into hashtable buckets and count the number of true nearest neighbours that fall within the same buckets as the query (Section 7.4.2). Surprisingly we discussed how this *hashtable lookup* evaluation strategy is not at all common in the learning to hash literature. Instead most publications of note use what is termed the *Hamming ranking* evaluation paradigm where the Hamming distance is exhaustively computed from the query to every data-point in the dataset (Section 7.4.1). The data-points are then ranked in ascending order of Hamming distance and the resulting ranked list is used to compute ranking-based evaluation metrics.

The next point we addressed in Section 7.5 was how to split the datasets into random partitions. In a retrieval setting we need a set of held-out test queries and a database over which retrieval will be performed. The accepted methodology in the literature (the *literature standard splitting strategy*) was to randomly select a set of held-out test queries and to use the remaining data-points as the database to be ranked *and* as the training dataset for learning the hash functions (Section 7.5.1). We identified a potential overfitting concern with this strategy and advocated an approach (the *improved splitting strategy*) where a certain split of the dataset forms a held-out database that cannot be used to learn the hash functions at training time (Section 7.5.2).

Finally in Section 7.6 we introduced the evaluation metrics frequently used to quantify the retrieval effectiveness of hashing models with respect to prior art. These include the standard the application of the standard Information Retrieval (IR) metrics of *area under the precision recall curve (AUPRC)* and *mean average precision (mAP)* to evaluate the quality of the hashcodes (Sections 7.6.3-7.6.4).

8. Avenues for Future Research

We conclude this review with suggestions for fruitful avenues of future research. The models presented in this review have but only scratched the surface of this important and flourishing field of research and the potential scope for future research is both many and varied. We will attempt to highlight several potential future directions of research that we consider particularly promising in this last section.

8.1 Groundtruth and Evaluation Metric Correlation with Human Judgments

There has been little previous work that examines the extent to which the evaluation metrics and groundtruth used in the learning to hash field are sensible for learning hashcodes that correlate well with user search satisfaction. For example, ideally it should be the case that a significant increase in the area under the precision recall curve (AUPRC) should also lead to a significant increase in user satisfaction with the retrieved images or documents. Furthermore, in Section 7 we introduced the class-based and ϵ-NN based groundtruth definitions that are used to evaluate the models in the literature. Many datasets of interest do not have manually assigned class labels, and so it would be useful to conduct a user-study as to how metric definitions of nearest neighbour groundtruth, such as the ϵ-NN groundtruth paradigm outlined in Section 7.3.1, align with human judgements of item-item similarity. Ideally we would want many related data-points to a given query, as judged by a user, to be contained within the same ϵ-ball. For the class-based groundtruth Section 7.3.2 this is less of an issue because those labels have been specifically assigned to the images by humans. The outcome of this user study would be expected to inform future developments in the evaluation procedures for hashing-based ANN search algorithms, and would be a valuable contribution to the community.

8.2 Online/Streaming Learning of the Hashing Hypersurfaces

A commonality between all of the reviewed models is the construction of the hashing hyperplanes in a batch fashion. This assumed the entire training dataset would be immediately available for learning, and ss soon as the hyperplanes were learnt they were never updated. This batch learning assumption is flawed when we consider many modern data sources of prime interest such as social media streams (e.g. Twitter). Twitter posts, for example, can be modelled as a never-ending, effectively infinite stream of data that could never be inspected in its entirety in a batch fashion (Petrović et al., 2010). Furthermore streaming data sources are highly likely to exhibit a drift in the distribution of the data over time as, for example, new topics are discussed and the vocabulary changes. Simply learning a set of hyperplanes once with no possibility of further updates would be an entirely suboptimal approach in this situation. It would be particularly interesting to adapt the bacth-based hashing models to the streaming data scenario in which the hyperplanes are capable of being updated in an online manner after each labelled pair of data-points are encountered in the stream. For example, to achieve this goal one could potentially investigate the effectiveness of using passive aggressive (PA) classifiers (Crammer, Dekel, Keshet, Shalev-Shwartz, & Singer, 2006) in place of the support vector machines (SVMs) used in the Self Taught or Graph Regularised Hashing models (Sections 6.4.4-6.4.5). The PA classifier is particularly

amenable to online learning and would make an ideal starting point for future research on this topic. We believe such a model would have significant potential impact in the field. An interesting challenge in this context would be how to efficiently update the hashcodes of existing data-points in the face of changing data. Furthermore, implementation of this model would address a common criticism of this field, namely the application of the algorithms to datasets of medium size (1 million data-points or less) and of relatively low dimensionality ($D \leq 512$).

8.3 Hashing Documents Written in Different Languages

The cross-modal hashing models extension are generally only tested on images and textual data, both of which are typically represented as low dimensional feature vectors. A particularly interesting avenue for future work would involve exploring how the models could be adapted to hash *cross-lingual documents*, for example English and Spanish Wikipedia articles. In this task the goal would be to cluster related cross-lingual documents in the same hashtable buckets, without using any form of machine translation. In contrast to the image and text features used in the literature, multi-lingual document data sources are likely to be very high dimensional when encoded as TF-IDF vectors. The large freely available *parallel and comparable corpora*[49] consisting of similar documents written in different languages would provide the needed pairwise supervision for learning the hashing hypersurfaces, negating any tedious manual effort to obtain the required labels. The cross-lingual projection function could be directly compared and evaluated against (Ture, Elsayed, & Lin, 2011), a solution based on machine translation and traditional unimodal Locality Sensitive Hashing (LSH). Given the significant gains in retrieval effectiveness for the cross-modal hashinbg models we have strong reason to suspect that cross-lingual hashing with a suitable adaptation of my graph regularised projection function would attract similar gains in performance. Given that more and more data on the Web is written in different languages we also foresee an online version of this cross-lingual hash function being particularly exciting future work. For example, a fast steaming algorithm for clustering similar tweets written in many different languages into the same hashtable buckets could prove useful to analysts in the financial industry or to linguists interested in studying the linguistic properties of Twitter and other related micro-blogs (Zanzotto, Pennacchiotti, & Tsioutsiouliklis, 2011).

8.4 Learning Dependent Hyperplanes and Quantisation Thresholds

The multiple threshold quantisation models introduced in Section 5 positioned the quantisation thresholds *independently* across each projected dimension. In other words, the learning of the quantisation thresholds for one projected dimension was independent of the learning of the quantisation thresholds for another projected dimension. Inspired by the body of research into *multivariate discretisation* (Bay, 2001) a potential future avenue of research could examine the benefits of inducing a degree of *dependence* between the quantisation thresholds across projected dimensions. A particularly simple, albeit contrived example of a dataset that would not be quantised correctly by independently optimised thresholds is the two dimensional XOR dataset (Bay, 2000). In this case the quantisation algorithm

49. `http://www.statmt.org/europarl/`

would need to account for the correlation between the different feature dimensions in order to find the optimal positioning of the thresholds.

In a similar vein of research, many of the supervised hash functions introduced in Section 6.4 constructed each hyperplane independently in a simple sequential fashion. Inducing a degree of dependence between the learning of the hyperplanes might contribute to a reduced redundancy between bits while also permitting hyperplanes learnt later in the sequence to focus on data-point pairs incorrectly classified by hyperplanes learnt earlier in the sequence. A straightforward starting point would be to assign a weight to each pair in the adjacency matrix in a similar manner to the Adaboost algorithm (Schapire & Freund, 2012). True nearest neighbours assigned the same bits by earlier hypersurfaces could have their weight decreased while non-nearest neighbours assigned the same bits could have their weights increased. In this way the learning of the hashing hyperplanes could be gradually biased to focus on data-points pairs that are more difficult to classify, potentially resulting in enhanced retrieval effectiveness.

9. Conclusion

This review intoduced the field of learning-to-hash for approximte nearest neighbour search. We began the review in Section 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 we argued is dominated by the seminal method of Locality Sensitive Hashing (LSH). We saw in Section 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. We discussed how the LSH hash function family for the inner product similarity forms the focus of this review. 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 naive brute-force linear scan.

In Sections 5-6, we then discussed how the contributions in the learning-to-hash field 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. We 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. In this survey our 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 6.3) or supervised (Section 6.4) signal to avoid placing related unimodal (Sections 6.3-6.4) or cross-modal (Section 6.5) data-points into different hashtable buckets.

Acknowledgments

The author would like to thank Victor Lavrenko, Robert Fisher, Iadh Ounis, Manfred Ulz and Aidan Reid for their helpful feedback on this review. Victor Lavrenko in particular, was instrumental in helping to crystallise and formalise the structure and layout of the review.

References

Aggarwal, C. C., & Reddy, C. K. (Eds.). (2014). *Data Clustering: Algorithms and Applications*. CRC Press.

Albakour, M.-D., Macdonald, C., & Ounis, I. (2015). Using sensor metadata streams to identify topics of local events in the city. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pp. 711–714, New York, NY, USA. ACM.

Andoni, A., & Indyk, P. (2008). Near-optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *Commun. ACM*, *51*(1), 117–122.

Arun, K. S., Huang, T. S., & Blostein, S. D. (1987). Least-Squares Fitting of Two 3-D Point Sets. *IEEE Trans. Pattern Anal. Mach. Intell.*, *9*(5), 698–700.

Baluja, S., & Covell, M. (2008). Learning to Hash: Forgiving Hash Functions and Applications. *Data Min. Knowl. Discov.*, *17*(3), 402–430.

Bawa, M., Condie, T., & Ganesan, P. (2005). LSH Forest: Self-tuning Indexes for Similarity Search. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pp. 651–660, New York, NY, USA. ACM.

Bay, S. D. (2000). Multivariate Discretization of Continuous Variables for Set Mining. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pp. 315–319, New York, NY, USA. ACM.

Bay, S. D. (2001). Multivariate Discretization for Set Mining. *Knowledge and Information Systems*, *3*(4), 491–512.

Belkin, M., & Niyogi, P. (2003). Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Comput.*, *15*(6), 1373–1396.

Bengio, Y., Paiement, J.-F., Vincent, P., Delalleau, O., Roux, N. L., & Ouimet, M. (2004). Out-of-Sample Extensions for LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering. In *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA.

Bentley, J. L. (1975). Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM*, *18*(9), 509–517.

Berchtold, S., Böhm, C., Braunmüller, B., Keim, D. A., & Kriegel, H.-P. (1997). Fast Parallel Similarity Search in Multimedia Databases. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, pp. 1–12, New York, NY, USA. ACM.

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

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet Allocation. *J. Mach. Learn. Res.*, *3*, 993–1022.

Broder, A. (1997). On the Resemblance and Containment of Documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES '97, pp. 21–, Washington, DC, USA. IEEE Computer Society.

Bronstein, M. M., Bronstein, A. M., Michel, F., & Paragios, N. (2010). Data fusion through cross-modality metric learning using similarity-sensitive hashing. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, *0*, 3594–3601.

Charikar, M. S. (2002). Similarity Estimation Techniques from Rounding Algorithms. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pp. 380–388, New York, NY, USA. ACM.

Chawla, N. V. (2005). Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*, pp. 853–867. Springer US.

Chua, T.-S., Tang, J., Hong, R., Li, H., Luo, Z., & Zheng, Y.-T. (2009). NUS-WIDE: A Real-World Web Image Database from National University of Singapore. In *Proc. of ACM Conf. on Image and Video Retrieval (CIVR'09)*, Santorini, Greece.

Chum, O., Philbin, J., & Zisserman, A. (2008). Near Duplicate Image Detection: min-Hash and tf-idf Weighting. In *Proceedings of the British Machine Vision Conference 2008, Leeds, September 2008*, pp. 1–10.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., & Singer, Y. (2006). Online Passive-Aggressive Algorithms. *J. Mach. Learn. Res.*, *7*, 551–585.

Datar, M., Immorlica, N., Indyk, P., & Mirrokni, V. S. (2004). Locality-sensitive Hashing Scheme Based on P-stable Distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, pp. 253–262, New York, NY, USA. ACM.

Dean, T., Ruzon, M., Segal, M., Shlens, J., Vijayanarasimhan, S., & Yagnik, J. (2013). Fast, Accurate Detection of 100,000 Object Classes on a Single Machine. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Washington, DC, USA.

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm.. *J. Royal Statistical Society, Series B*, *39*(1), 1–38.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., & Li, F. (2009). ImageNet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pp. 248–255.

Diaz, F. (2007). Regularizing Query-based Retrieval Scores. *Inf. Retr.*, *10*(6), 531–562.

Doersch, C., Singh, S., Gupta, A., Sivic, J., & Efros, A. A. (2012). What Makes Paris Look Like Paris?. *ACM Trans. Graph.*, *31*(4), 101:1–101:9.

Dougherty, J., Kohavi, R., & Sahami, M. (1995). Supervised and Unsupervised Discretization of Continuous Features. In *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*, pp. 194–202.

Fan, R., Chang, K., Hsieh, C., Wang, X., & Lin, C. (2008). LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, *9*, 1871–1874.

Fayyad, U. M., & Irani, K. B. (1993). Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, pp. 1022–1029.

Feng, J. (2012). Mobile Product Search with Bag of Hash Bits and Boundary Reranking. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pp. 3005–3012, Washington, DC, USA. IEEE Computer Society.

Finkel, R. A., & Bentley, J. L. (1974). Quad Trees: A Data Structure for Retrieval on Composite Keys.. *Acta Inf.*, *4*, 1–9.

Freund, Y., & Schapire, R. E. (1997). A Decision-theoretic Generalization of On-line Learning and an Application to Boosting. *J. Comput. Syst. Sci.*, *55*(1), 119–139.

Garcia, S., Luengo, J., Saez, J. A., Lopez, V., & Herrera, F. (2013). A Survey of Discretization Techniques: Taxonomy and Empirical Analysis in Supervised Learning. *IEEE Trans. on Knowl. and Data Eng.*, *25*(4), 734–750.

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pp. 580–587, Washington, DC, USA. IEEE Computer Society.

Goemans, M. X., & Williamson, D. P. (1995). Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *J. ACM*, *42*(6), 1115–1145.

Goldberger, J., Roweis, S., Hinton, G., & Salakhutdinov, R. (2004). Neighbourhood components analysis. In *Advances in Neural Information Processing Systems 17*, pp. 513–520. MIT Press.

Golub, G. H., & Van Loan, C. F. (1996). *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA.

Gong, Y., & Lazebnik, S. (2011). Iterative Quantization: A Procrustean Approach to Learning Binary Codes. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pp. 817–824, Washington, DC, USA. IEEE Computer Society.

Grauman, K., & Darrell, T. (2004). Fast Contour Matching Using Approximate Earth Mover's Distance.. In *CVPR (1)*, pp. 220–227.

Grauman, K., & Darrell, T. (2007). Pyramid Match Hashing: Sub-Linear Time Indexing Over Partial Correspondences. In *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), 18-23 June 2007, Minneapolis, Minnesota, USA*.

Grauman, K., & Fergus, R. (2013). Learning Binary Hash Codes for Large-Scale Image Search. In Cipolla, R., Battiato, S., & Farinella, G. M. (Eds.), *Machine Learning for Computer Vision*, Vol. 411 of *Studies in Computational Intelligence*, pp. 49–87. Springer Berlin Heidelberg.

Gray, F. (1953). Pulse code communication.. US Patent 2,632,058.

Gray, R. M., & Neuhoff, D. L. (2006). Quantization. *IEEE Trans. Inf. Theor.*, *44*(6), 2325–2383.

Hanson, R. J., & Norris, M. J. (1981). Analysis of Measurements Based on the Singular Value Decomposition. *SIAM Journal on Scientific and Statistical Computing*, *2*(3), 363–373.

Hardoon, D. R., Szedmak, S., & Shawe-Taylor, J. (2003). Canonical correlation analysis; An overview with application to learning methods. Technical report, Royal Holloway, University of London.

He, K., Wen, F., & Sun, J. (2013). K-Means Hashing: An Affinity-Preserving Quantization Method for Learning Binary Compact Codes. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '13, pp. 2938–2945, Washington, DC, USA. IEEE Computer Society.

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *J. Educ. Psych.*, *24*.

Indyk, P., & Motwani, R. (1998). Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pp. 604–613, New York, NY, USA. ACM.

Jegou, H., Douze, M., & Schmid, C. (2011). Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.*, *33*(1), 117–128.

Joachims, T. (2006). Training Linear SVMs in Linear Time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pp. 217–226, New York, NY, USA. ACM.

Johnson, W., & Lindenstrauss, J. (1984). Extensions of Lipschitz mappings into a Hilbert space. In *Contemp Math 26*.

Katayama, N., & Satoh, S. (1997). The SR-tree: An Index Structure for High-dimensional Nearest Neighbor Queries. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, pp. 369–380, New York, NY, USA. ACM.

Kerber, R. (1992). ChiMerge: Discretization of Numeric Attributes. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, pp. 123–128. AAAI Press.

Kokiopoulou, E., Chen, J., & Saad, Y. (2011). Trace optimization and eigenproblems in dimension reduction methods. *Numerical Lin. Alg. with Applic.*, *18*(3), 565–602.

Kong, W., & Li, W. (2012a). Double-Bit Quantization for Hashing. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*

Kong, W., & Li, W. (2012b). Isotropic Hashing. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pp. 1655–1663.

Kong, W., Li, W.-J., & Guo, M. (2012). Manhattan Hashing for Large-scale Image Retrieval. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pp. 45–54, New York, NY, USA. ACM.

Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto.*

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pp. 1106–1114.

Kulis, B. (2013). Metric Learning: A Survey. *Foundations and Trends in Machine Learning*, *5*(4), 287–364.

Kulis, B., & Darrell, T. (2009). Learning to Hash with Binary Reconstructive Embeddings. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pp. 1042–1050.

Kulis, B., & Grauman, K. (2009). Kernelized locality-sensitive hashing for scalable image search. In *IEEE 12th International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, September 27 - October 4, 2009*, pp. 2130–2137.

Kumar, S., & Udupa, R. (2011). Learning Hash Functions for Cross-view Similarity Search. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI '11, pp. 1360–1365. AAAI Press.

Lai, H., Pan, Y., Liu, Y., & Yan, S. (2015). Simultaneous feature learning and hash coding with deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pp. 3270–3278.

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pp. 2278–2324.

Lin, M., Chen, Q., & Yan, S. (2014). Network in network..

Liong, V. E., Lu, J., Wang, G., Moulin, P., & Zhou, J. (2015). Deep hashing for compact binary codes learning.. In *CVPR*, pp. 2475–2483. IEEE Computer Society.

Liu, W., Mu, C., Kumar, S., & Chang, S. (2014). Discrete Graph Hashing. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 3419–3427.

Liu, W., Wang, J., Ji, R., Jiang, Y., & Chang, S. (2012). Supervised hashing with kernels. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pp. 2074–2081.

Liu, W., Wang, J., Kumar, S., & Chang, S. (2011). Hashing with Graphs. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pp. 1–8.

Liu, X., He, J., & Lang, B. (2013). Reciprocal Hash Tables for Nearest Neighbor Search. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*.

Lloyd, S. (1982). Least Square Quantization in PCM. *IEEE Trans. Inform. Theory*, *28*, 129–137.

Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, *60*(2), 91–110.

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

Matei, B., Shan, Y., Sawhney, H. S., Tan, Y., Kumar, R., Huber, D., & Hebert, M. (2006). Rapid Object Indexing Using Locality Sensitive Hashing and Joint 3D-Signature Space Estimation. *IEEE Trans. Pattern Anal. Mach. Intell.*, *28*(7), 1111–1126.

Mehta, S., Parthasarathy, S., & Yang, H. (2005). Toward Unsupervised Correlation Preserving Discretization.. Vol. 17, pp. 1174–1185, Piscataway, NJ, USA. IEEE Educational Activities Department.

Minsky, M., & Papert, S. (1969). *Perceptrons*. MIT Press, Cambridge, MA.

Moran, S. (2016). Learning to Project and Binarise for Hashing-Based Approximate Nearest Neighbour Search. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16.

Moran, S., & Lavrenko, V. (2014). Sparse Kernel Learning for Image Annotation. In *Proceedings of International Conference on Multimedia Retrieval*, ICMR '14, pp. 113:113–113:120, New York, NY, USA. ACM.

Moran, S., & Lavrenko, V. (2015a). Graph Regularised Hashing. In *Advances in Information Retrieval - 37th European Conference on IR Research, ECIR 2015, Vienna, Austria, March 29 - April 2, 2015. Proceedings*, pp. 135–146.

Moran, S., & Lavrenko, V. (2015b). Regularised Cross-Modal Hashing. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pp. 907–910. ACM.

Moran, S., Lavrenko, V., & Osborne, M. (2013a). Neighbourhood Preserving Quantisation for LSH. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, pp. 1009–1012, New York, NY, USA. ACM.

Moran, S., Lavrenko, V., & Osborne, M. (2013b). Variable Bit Quantisation for LSH. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 2: Short Papers*, pp. 753–758.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.

Oliva, A., & Torralba, A. (2001). Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope. *Int. J. Comput. Vision*, *42*(3), 145–175.

Osborne, M., Moran, S., McCreadie, R., von Lünen, A., Sykora, M. D., Cano, A. E., Ireson, N., Macdonald, C., Ounis, I., He, Y., Jackson, T., Ciravegna, F., & O'Brien, A. (2014). Real-Time Detection, Tracking, and Monitoring of Automatically Discovered Events in Social Media. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*, pp. 37–42.

Petrovic, S. (2012). *Real-Time Event Detection in Massive Streams*. Ph.D. thesis, University of Edinburgh.

Petrović, S., Osborne, M., & Lavrenko, V. (2010). Streaming First Story Detection with Application to Twitter. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pp. 181–189, Stroudsburg, PA, USA. Association for Computational Linguistics.

Raginsky, M., & Lazebnik, S. (2009). Locality-sensitive binary codes from shift-invariant kernels. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pp. 1509–1517.

Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pp. 1177–1184.

Rajaraman, A., & Ullman, J. D. (2011). *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA.

Rasiwasia, N., Costa Pereira, J., Coviello, E., Doyle, G., Lanckriet, G., Levy, R., & Vasconcelos, N. (2010). A New Approach to Cross-Modal Multimedia Retrieval. In *ACM International Conference on Multimedia*, pp. 251–260.

Rasiwasia, N., Mahajan, D., Mahadevan, V., & Aggarwal, G. (2014). Cluster Canonical Correlation Analysis. In *Proceedings of International Conference on Artificial Intelligence and Statistics*.

Rastegari, M., Choi, J., Fakhraei, S., III, H. D., & Davis, L. S. (2013). Predictable dual-view hashing. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pp. 1328–1336.

Ravichandran, D., Pantel, P., & Hovy, E. (2005). Randomized Algorithms and NLP: Using Locality Sensitive Hash Function for High Speed Noun Clustering. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pp. 622–629, Stroudsburg, PA, USA. Association for Computational Linguistics.

Razavian, A. S., Azizpour, H., Sullivan, J., & Carlsson, S. (2014). CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, *abs/1403.6382*.

Rijsbergen, C. J. V. (1979). *Information Retrieval* (2nd edition). Butterworth-Heinemann, Newton, MA, USA.

Russell, B. C., Torralba, A., Murphy, K. P., & Freeman, W. T. (2008). LabelMe: A Database and Web-Based Tool for Image Annotation. *Int. J. Comput. Vision*, *77*(1-3), 157–173.

Saad, Y. (Ed.). (2011). *Numerical Methods for Large Eigenvalue Problems, 2nd revised edition*. SIAM.

Salakhutdinov, R., & Hinton, G. (2009). Semantic Hashing. *Int. J. Approx. Reasoning*, *50*(7), 969–978.

Schapire, R. E., & Freund, Y. (2012). *Boosting: Foundations and Algorithms*. The MIT Press.

Schönemann, P. (1966). A generalized solution of the orthogonal procrustes problem. *Psychometrika*, *31*(1), 1–10.

Sebastiani, F. (2002). Machine Learning in Automated Text Categorization. *ACM Comput. Surv.*, *34*(1), 1–47.

Shakhnarovich, G., Darrell, T., & Indyk, P. (2006). *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing)*. The MIT Press.

Shakhnarovich, G., Viola, P., & Darrell, T. (2003). Fast Pose Estimation with Parameter-Sensitive Hashing. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pp. 750–, Washington, DC, USA. IEEE Computer Society.

Shalev-Shwartz, S., Singer, Y., & Srebro, N. (2007). Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pp. 807–814, New York, NY, USA. ACM.

Shen, F., Shen, C., Liu, W., & Tao Shen, H. (2015). Supervised Discrete Hashing. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 37–45.

Shi, J., & Malik, J. (2000). Normalized Cuts and Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, *22*(8), 888–905.

Smeulders, A. W. M., Worring, M., Santini, S., Gupta, A., & Jain, R. (2000). Content-Based Image Retrieval at the End of the Early Years. *IEEE Trans. Pattern Anal. Mach. Intell.*, *22*(12), 1349–1380.

Song, J., Yang, Y., Yang, Y., Huang, Z., & Shen, H. T. (2013). Inter-media Hashing for Large-scale Retrieval from Heterogeneous Data Sources. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pp. 785–796, New York, NY, USA. ACM.

Terasawa, K., & Tanaka, Y. (2007). Spherical LSH for Approximate Nearest Neighbor Search on Unit Hypersphere. In Dehne, F., Sack, J.-R., & Zeh, N. (Eds.), *Algorithms and Data Structures*, Vol. 4619 of *Lecture Notes in Computer Science*, pp. 27–38. Springer Berlin Heidelberg.

Torralba, A., Fergus, R., & Weiss, Y. (2008). Small codes and large image databases for recognition. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*.

Ture, F., Elsayed, T., & Lin, J. (2011). No Free Lunch: Brute Force vs. Locality-sensitive Hashing for Cross-lingual Pairwise Similarity. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pp. 943–952, New York, NY, USA. ACM.

Turpin, A., & Scholer, F. (2006). User Performance Versus Precision Measures for Simple Search Tasks. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pp. 11–18, New York, NY, USA. ACM.

Ulz, M. H., & Moran, S. J. (2013). Optimal kernel shape and bandwidth for atomistic support of continuum stress. *Modelling and Simulation in Materials Science and Engineering*, *21*(8), 085017.

Wang, J., Shen, H. T., Song, J., & Ji, J. (2014). Hashing for Similarity Search: A Survey. *CoRR*, *abs/1408.2927*.

Wang, J., Kumar, O., & Chang, S. (2010a). Semi-supervised hashing for scalable image retrieval. In *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*, pp. 3424–3431.

Wang, J., Kumar, S., & Chang, S. (2010b). Sequential Projection Learning for Hashing with Compact Codes. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pp. 1127–1134.

Wang, J., Kumar, S., & Chang, S.-F. (2012). Semi-Supervised Hashing for Large-Scale Search. *IEEE Trans. Pattern Anal. Mach. Intell.*, *34*(12), 2393–2406.

Wang, J., Liu, W., Kumar, S., & Chang, S.-F. (2015a). Learning to hash for indexing big data - a survey.. *CoRR*, *abs/1509.05472*.

Wang, Z., Duan, L.-Y., Lin, J., Wang, X., Huang, T., & Gao, W. (2015b). Hamming Compatible Quantization for Hashing. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, IJCAI '15.

Weber, R., Schek, H.-J., & Blott, S. (1998). A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pp. 194–205, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Weiss, Y., Torralba, A., & Fergus, R. (2008). Spectral Hashing. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pp. 1753–1760.

Williams, C. K. I., & Seeger, M. (2001). Using the Nyström Method to Speed Up Kernel Machines. In Leen, T., Dietterich, T., & Tresp, V. (Eds.), *Advances in Neural Information Processing Systems 13*, pp. 682–688. MIT Press.

Wu, B., Yang, Q., Zheng, W.-S., Wang, Y., & Wang, J. (2015). Quantized Correlation Hashing for Fast Cross-modal Search. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI '15, pp. 3946–3952. AAAI Press.

Xia, R., Pan, Y., Lai, H., Liu, C., & Yan, S. (2014). Supervised hashing for image retrieval via image representation learning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, pp. 2156–2162. AAAI Press.

Xu, D., Cham, T. J., Yan, S., Duan, L., & Chang, S.-F. (2010). Near Duplicate Identification With Spatially Aligned Pyramid Matching. *IEEE Transactions on Circuits and Systems for Video Technology*, *20*, 1068–1079.

Xu, H., Wang, J., Li, Z., Zeng, G., Li, S., & Yu, N. (2011). Complementary Hashing for Approximate Nearest Neighbor Search. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pp. 1631–1638, Washington, DC, USA. IEEE Computer Society.

Yang, Y., & Webb, G. I. (2009). Discretization for naive-Bayes Learning: Managing Discretization Bias and Variance. *Mach. Learn.*, *74*(1), 39–74.

Yuille, A. L., & Rangarajan, A. (2003). The Concave-convex Procedure. *Neural Comput.*, *15*(4), 915–936.

Zanzotto, F. M., Pennacchiotti, M., & Tsioutsiouliklis, K. (2011). Linguistic Redundancy in Twitter. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pp. 659–669, Stroudsburg, PA, USA. Association for Computational Linguistics.

Zhang, D., Wang, J., Cai, D., & Lu, J. (2010). Self-taught Hashing for Fast Similarity Search. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pp. 18–25, New York, NY, USA. ACM.

Zhen, Y., & Yeung, D. (2012). Co-Regularized Hashing for Multimodal Data. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pp. 1385–1393.

Zhu, X., & Ghahramani, Z. (2002). Learning from Labeled and Unlabeled Data with Label Propagation. Tech. rep., Technical Report CMU-CALD-02-107, Carnegie Mellon University.