Linear Spectral Hashing

Zalán Bodó[a,*], Lehel Csató[a]

[a] *Babeş–Bolyai University, Faculty of Mathematics and Computer Science, Kogălniceanu 1., 400084 Cluj-Napoca, Romania*

Abstract

Spectral hashing assigns binary hash keys to data points. This is accomplished via thresholding the eigenvectors of the graph Laplacian and obtaining binary codewords. While calculation for inputs in the training set is straightforward, an intriguing and difficult problem is how to compute the hash codewords for previously unseen data. For specific problems we propose linear scalar products as similarity measures and analyze the performance of the algorithm. We implement the linear algorithm and provide an inductive – generative – formula that leads to a codeword generation method similar to random hyperplane-based *locality-sensitive hashing* for a new data point.

Keywords: nearest-neighbor search, spectral hashing, spectral clustering
2000 MSC: 62M15,
2000 MSC: 58C40,
2000 MSC: 62H30

1. Introduction

In recent years several algorithms were proposed for fast approximate nearest-neighbor search, providing sub-linear search times for a query. Introduced by Cover and Hart [1], the k-nearest neighbor algorithm is one of the most successful and most investigated methods in the machine learning community. However, for large datasets it has a serious drawback: it must go through all the *training* data to find the nearest neighbors. This implies a linear time complexity for a single point. The method of k-nearest neighbors

*Corresponding author
Email addresses: zbodo@cs.ubbcluj.ro (Zalán Bodó), lehel.csato@cs.ubbcluj.ro (Lehel Csató)

can be used in machine learning to label an unknown point based on the majority label among the neighbors. Considering the nearest neighbors of a point, however, is not used only in machine learning; the properties of the neighbors can be beneficially utilized to infer supposedly valid facts about the point in question. Without pretension to completeness, we can mention important applications in information retrieval, computer vision, coding theory, recommendation systems, computational geometry, etc. [2] Therefore, speeding up the search for nearest neighbors we consider a task of great importance.

One of the first approaches to reduce the number of comparisons in finding the nearest-neighbors of a given point was to store points in a special data structure, a k-d tree [3]. K-d trees are binary space-partitioning trees, in which every node is a k-dimensional point, used to hierarchically decompose the space by hyperplanes along orthogonal dimensions. The main disadvantage of using such data structures lies in its sensibility to high-dimensional points, in which case a nearest-neighbor search evaluates most of the points in the tree.

Learned binary embeddings for large datasets, where nearest-neighbors of a given point needed to be found, are efficient and powerful tools for indexing these sets. These embeddings are designed to approximately preserve similarities in the embedding Hamming space. The beneficial properties of the codewords make possible to use Hamming distance computations for nearest-neighbor search, by which the searching process can be substantially sped up: to compute the Hamming distance of two vectors, a XOR operation has to be performed between the vectors and the resulting set bits have to be counted.

We differentiate between two parts of fast nearest-neighbor search with binary embeddings. The first part consists of generating the binary codes, and the second part is the actual searching process. The simplest method for determining the nearest-neighbors is linear scan: calculate the Hamming distance to every codeword from the database. Although it is a brute-force method, it is practical even for very large datasets. Semantic hashing [4] maps binary codes to memory addresses, where at each address a pointer is stored to a list of points that share the respective codeword. Querying the neighbors of a point is done by calculating the hash codeword, flipping some of the bits to obtain the desired Hamming ball around the query, and taking the points with these codewords. Another approach is locality-sensitive hashing (LSH) [5], a randomized framework for generating hash codewords and searching for nearest-neighbors. It creates a hash table where similar points will likely

to be put in the same bucket or nearby buckets having a small Hamming distance. Linear scan and semantic hashing can be used with any codeword generation method, however, in semantic hashing the codeword length is limited by the size of the physical memory. LSH provides an elegant way to control the size of the Hamming ball in the original feature space, but choosing an appropriate LSH function limits its flexibility [6].

One can distinguish between unsupervised, supervised and semi-supervised codeword generation methods, based on the information they use to obtain the embedding [6]. Unsupervised methods use only the information carried by the points themselves. Additional information can be given to supervised methods in form of label information as in a supervised machine learning setting, as well as giving the neighborhood list for a subset of points, or as paired constraints – defining the points which ought or ought not cluster together. Finally, semi-supervised methods exploit the supervised information, besides which other clustering or regularization approaches are used.

Locality-sensitive hashing with hyperplanes [7] is perhaps the most popular unsupervised hashing method, generating binary codewords by taking random vectors as normal vectors of separating hyperplanes. The hash function is the sign of the dot product between the data point and the random vectors. In [8] it was extended to support arbitrary kernels by using a clever method to generate random vectors in the feature space. Spectral hashing [9] picks codewords by searching minimum-distance binary hash sequences between similar points, similarity being defined by an appropriate proximity matrix. The optimization leads to a clustering problem, from which it becomes intriguing and difficult to generalize to new data points. In [9] this is solved using the eigenfunctions of the weighted Laplace–Beltrami operators, assuming a multidimensional uniform distribution. Shift-invariant kernel-based binary codes [10] use the random projections of [11] giving a low-dimensional similarity-preserving representation of points, where similarity is defined by the kernel. These low-dimensional vectors are then used to obtain binary codes for hashing. The method is limited to use shift-invariant (e.g. Gaussian) kernels. Self-taught hashing [12] uses a two-step approach for codeword generation. The first step consists of the unsupervised learning of binary codes using a similar objective function to spectral hashing. To generalize the obtained mappings, in the second step it considers the set of training examples together with their labels, obtained in the previous step, and trains a support vector machine for each bit of the codeword. We also mention Laplacian co-hashing [13], a hashing method for document

retrieval. It considers the documents and terms a bipartite graph, where a term is connected to a document if appears in it, and finds binary codes that best preserves the similarities encoded by the term–document matrix. It differs from existing methods by simultaneously minimizing Hamming distances between document and term codewords.

Parameter-sensitive hashing [14] is an extension of LSH which trains classifiers for the individual regression tasks using label information. In [15] the codeword generation method learns a parametrization of the Mahalanobis distance, based on the supervised information given, and simultaneously encodes the learned information into randomized hash functions. This approach uses the hash function family proposed by [7] with generalized dot products. The embedding proposed in [4] uses multiple layers of restricted Boltzmann machines to assign efficient binary codes to documents. The learning process consists of two phases: an unsupervised pre-training and a supervised fine-tuning of the weights using label information.

The recently proposed semi-supervised approaches from works [16, 17] minimize the empirical error on the labeled training data and an information theoretic regularizer over both labeled and unlabeled set.

Getting supervised information is often a costly process. Spectral hashing is a successful unsupervised codeword generation method that *learns* short binary codewords by minimizing the codeword distances between similar points. Assuming a multidimensional uniform distribution is a very limiting factor, however, spectral hashing performs surprisingly well in practice on a large variety of data. Furthermore, in the original formulation of the method the Gaussian kernel is used, which is not well-suited for specific problems.

In this paper we propose another method for computing the hash codewords for previously unseen points using the linear kernel. The codeword is obtained by computing the dot product of the new vector with a set of precomputed vectors and thresholding the resulting values at zero. The vectors output by the method are normal vectors of maximum-margin separating hyperplanes. Figure 1 illustrates our method, hashing three-dimensional points to two dimensions.

The structure of the paper is as follows: Section 2 presents spectral clustering in general. It presents normalized spectral clustering and its generalization using maximum-margin separating hyperplanes. After presenting spectral hashing in Section 3, the maximum-margin formulation is used in Section 4 to derive a new algorithm for computing the codewords of new data

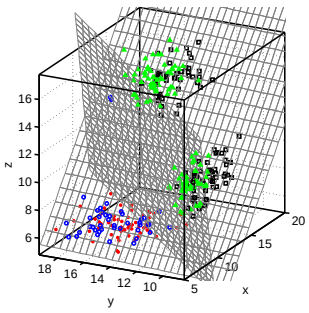


Figure 1: Points sampled from three multivariate Gaussians and their hashing to two dimensions performed using linear spectral hashing.

points. Sections 5 and 6 describe our experiments and discuss the results.

2. Spectral clustering and max-margin hyperplanes

2.1. Spectral clustering

In *spectral clustering* [18] – one of the most successful clustering algorithms – partitioning is done using the minimum cut in the neighborhood graph, where minimum cut is defined as removing edges such that the graph is partitioned into disconnected subgraphs with minimal sum of edges, $\sum_{i \in A, j \in \overline{A}} W_{ij}$, where A and $\overline{A}$ denotes the clusters, V is the entire vertex set, $A \cup \overline{A} = V$, and W_{ij} is the proximity of the i-th and j-th examples.

The sum of edges alone as objective function favors unbalanced and small clusters. To overcome this problem, normalized cut was introduced [19], where instead of considering only the sum of weights, the ratio of the sum of weights between partitions and the sum of weights to all the nodes from the partitions is taken, that is

$$\frac{\sum_{i \in A, j \in \overline{A}} W_{ij}}{\sum_{i \in A, v \in V} W_{iv}} + \frac{\sum_{i \in A, j \in \overline{A}} W_{ij}}{\sum_{j \in \overline{A}, v \in V} W_{jv}} \tag{1}$$

The above formulation of spectral clustering considers only binary partitioning, but the problem can be extended to support multiple clusters. The different methods of spectral clustering for multi-cluster partitioning are discussed in the extensive work of von Luxburg [18].

To avoid the combinatorial explosion, the original problem is relaxed to the following optimization on a continuous domain [19, 20]:

$$\begin{aligned} \mathbf{z}^* &= \underset{\mathbf{z}\in\mathbb{R}^N}{\operatorname{argmax}} \frac{\mathbf{z}'\mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}\mathbf{z}}{\mathbf{z}'\mathbf{z}} \\ &\text{s.t. } \mathbf{z}'\mathbf{D}^{1/2}\mathbf{1} = 0 \end{aligned} \tag{2}$$

where $\mathbf{z}$ is the vector of – continuous – labels, $\mathbf{z}'$ denotes its transpose, $\mathbf{W}$ is a similarity matrix, and $\mathbf{D} = \operatorname{diag}(\mathbf{W1})$ is the diagonal degree matrix. Similarity can be defined using any positive definite function, and often it is defined using the Gaussian kernel: $W_{ij} = k_G(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$. Problem (2) is solved by the eigenvector with the second largest (< 1) eigenvalue of the normalized graph Laplacian $\mathbf{L}_{\text{sym}} = \mathbf{D}^{-1/2}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-1/2}$, where $\mathbf{L} = \mathbf{D} - \mathbf{W}$ denotes the Laplacian.[1] Having the solution, the cluster indicator is the thresholded vector of $\mathbf{D}^{-1/2}\mathbf{z}^*$ or simply $\mathbf{z}^*$.

2.2. Generalization

An equivalent view of the optimization from Eq. (2) is that of separating the points by a maximum-margin hyperplane in the feature space defined by the same proximity matrix $\mathbf{W}$ [20], that we detail in the following.

Let us consider the matrix $\mathbf{\Phi} = [\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \ldots, \phi(\mathbf{x}_N)]$ as the projection of the training dataset in a feature space defined by the mapping $\phi : X \to \mathcal{H}$. We are looking for a *separating hyperplane* in that feature space, defined by the normal $\mathbf{w} \in \mathcal{H}$. Consider the following problem:

$$\mathbf{w}^* = \underset{\mathbf{w}\in\mathcal{H}}{\operatorname{argmax}} \|\mathbf{w}'\mathbf{\Phi}\mathbf{D}^{-1/2}\|^2 \tag{3}$$

where $\mathbf{D}$ is the degree matrix from the previous section. To ensure that the clusters induced by the hyperplane are approximately the same size, the constraint $\sum_{i=1}^{N} \mathbf{w}'\phi(\mathbf{x}_i) = 0$ is introduced, and we also restrict $\|\mathbf{w}\| = 1$. Then we rewrite the optimization problem as

$$\mathbf{w}^* = \underset{\mathbf{w}\in\mathcal{H}}{\operatorname{argmax}} \frac{\mathbf{w}'\mathbf{\Phi}\mathbf{D}^{-1}\mathbf{\Phi}'\mathbf{w}}{\mathbf{w}'\mathbf{w}} \tag{4}$$

[1] There are two normalized variants of the graph Laplacian used in the literature: random walk and symmetric [18]. Throughout the paper we use the symmetric normalized Laplacian, and by a slight abuse of terminology we simply call it the normalized (graph) Laplacian.

$$\text{s.t.} \sum_{i=1}^{N} \mathbf{w}'\phi(\mathbf{x}_i) = 0$$

Similarly to spectral clustering, the solution of the above optimization is the eigenvector of the second largest eigenvalue of the matrix $\mathbf{\Phi D}^{-1}\mathbf{\Phi}'$ [20]. As this is a feature space entity, its dimensionality is often infinite, therefore we aim at expressing the result using a smaller dimensional entity. We can use the spectral decomposition of the matrices $\mathbf{AA}' = \mathbf{D}^{-1/2}\mathbf{\Phi}'\mathbf{\Phi D}^{-1/2} = \mathbf{D}^{-1/2}\mathbf{WD}^{-1/2}$ and $\mathbf{A}'\mathbf{A} = \mathbf{\Phi D}^{-1}\mathbf{\Phi}'$ and establish that if $\mathbf{v}_2$ is the second eigenvector of $\mathbf{AA}'$, then $\mathbf{u}_2 = \mathbf{\Phi D}^{-1/2}\mathbf{v}_2$ is the second eigenvector of $\mathbf{A}'\mathbf{A}$, with e_2 the second largest eigenvalue.[2]

The result is important: we obtain an *inductive clusterizer*, a hyperplane in the feature space $\mathcal{H}$ with normal $\mathbf{w}^* = \mathbf{u}_2 = \mathbf{\Phi D}^{-1/2}\mathbf{v}_2$. We highlight the inductive nature of the result: no repeated eigenvector computations are needed, but the cluster label of a new data point $\mathbf{x}$ is calculated by

$$f_2(\mathbf{x}) = \phi(\mathbf{x})'\mathbf{u}_2 = \phi(\mathbf{x})'\mathbf{\Phi D}^{-1/2}\mathbf{v}_2 \tag{5}$$

where $f_2(\mathbf{x})$ refers to the fact that the scalar product with the second largest eigenvalue is computed. Furthermore, the separator from above assigns the same cluster labels to the points as normalized spectral clustering, since

$$\begin{aligned} \operatorname{sgn}\left(\mathbf{\Phi}'\mathbf{w}^*\right) &= \operatorname{sgn}\left(\mathbf{\Phi}'\mathbf{\Phi D}^{-1/2}\mathbf{v}_2\right) \\ &= \operatorname{sgn}\left(\mathbf{D}^{-1/2}\mathbf{\Phi}'\mathbf{\Phi D}^{-1/2}\mathbf{v}_2\right) \\ &= \operatorname{sgn}\left(\mathbf{v}_2 e_2\right) = \operatorname{sgn}\left(\mathbf{v}_2\right) \end{aligned}$$

thus completing the proof.

The above formula can be applied to every eigenvector. We use this result in spectral hashing for computing the hash codeword for a previously unseen point.

3. Spectral hashing

Spectral hashing [9] is a novel method for hashing-based nearest-neighbor search. It is based on minimizing the weighted distances between binary codewords, and the weighting is based on the similarity between data points. As

[2]The eigenvalues are counted with multiplicity.

in hyperplane LSH, this binary sequence clusters the points, but in contrast to the classical LSH output, the length of the sequence can be significantly smaller.

We want thus to find a mapping that assigns codewords to data points such that these codewords are close to each other for nearby data points, where we can consider the similarity matrix $\mathbf{W}$ from Section 2. If $\mathbf{y}_i$, $i = 1, \ldots, N$ denotes the codewords for the training points, and the vectors $\mathbf{y}'_i$ are organized into the rows of the matrix $\mathbf{Y}$, then the following optimization problem has to be solved:

$$\begin{aligned} \mathbf{Y}^* &= \underset{\mathbf{Y}\in\{-1,1\}^{N\times r}}{\operatorname{argmin}} \sum_{i,j=1}^{N} W_{ij}\|\mathbf{y}_i - \mathbf{y}_j\|^2 \\ &\quad \text{s.t. } \sum_{i=1}^{N} \mathbf{y}_i = \mathbf{0}, \quad \frac{1}{N}\sum_{i=1}^{N} y_i y'_i = \mathbf{I} \end{aligned} \tag{6}$$

where the first condition is for maintaining the balance between the r bits – bits are distributed evenly over $\{-1, 1\}$ – and the second one makes the bits uncorrelated. Since the optimization problem above is NP-hard [9], analogously to spectral clustering, using the graph Laplacian $\mathbf{L}$, it is relaxed into

$$\begin{aligned} \mathbf{Y}^* &= \underset{\mathbf{Y}\in\mathbb{R}^{N\times r}}{\operatorname{argmin}} \operatorname{tr}(\mathbf{Y}'\mathbf{L}\mathbf{Y}) \\ &\quad \text{s.t. } \mathbf{Y}'\mathbf{1} = \mathbf{0}, \quad \mathbf{Y}'\mathbf{Y} = \mathbf{I} \end{aligned} \tag{7}$$

Excluding the first constraint, the solution of the problem is given by the first r eigenvectors of $\mathbf{L}$, $\mathbf{Y}^* = [\mathbf{v}_1,\ \mathbf{v}_2, \ldots,\ \mathbf{v}_r]$ [21]. Thus $\mathbf{y}_i^{*\prime} = [v_{1i},\ v_{2i},\ \ldots,\ v_{ri}]$. The first constraint is satisfied by all eigenvectors, except the first one, the constant one eigenvector with eigenvalue 0, because $\mathbf{Y}^{*\prime}\mathbf{1} = \mathbf{0}$ is equivalent to requiring the dot product of each eigenvector with the constant one vector to equal zero. Therefore, the final solution is given by $\mathbf{Y}^* = [\mathbf{v}_2,\ \mathbf{v}_3,\ \ldots,\ \mathbf{v}_{r+1}]$ where $\mathbf{v}_2$ denotes the eigenvector corresponding to the second smallest positive eigenvalue.

In [9], using the Gaussian kernel, generalization was done using the eigenfunctions of the weighted Laplace–Beltrami operators, assuming a multidimensional uniform distribution. Although, this assumption is very limiting, spectral hashing performs surprisingly well on a variety of data having different distribution [9]. In this paper we use the results from Section 2, providing a more simple and elegant approach for the linear case.

4. Linear spectral hashing

The crucial problem is to generalize spectral hashing for new data points. That could be solved by including the point in the dataset and recalculating the first r eigenvectors. It is however obvious that this is not admissible, since turns the problem into a more complex one than the initial search was.

In this section we connect spectral hashing with spectral clustering and use the generalization results from Section 2. For this, in problem (7) we simply change the Laplacian to the normalized Laplacian $\mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$, thus the solution is now given by the first r eigenvectors of the normalized Laplacian with strictly positive eigenvalues. Subsequently, we can use Eq. (5) to compute the *cluster labels* of a point $\mathbf{x}$ by substituting the first r eigenvectors – starting from the second largest eigenvalue – into $\mathbf{u}_k$ or $\mathbf{v}_k$:

$$f(\mathbf{x}) = \operatorname{sgn}\left([f_2(\mathbf{x}),\ f_3(\mathbf{x}),\ \ldots,\ f_{r+1}(\mathbf{x})]'\right)$$

and we assumed that the new $\mathbf{x}$ comes from the same distribution as the training points, thus using (5) we approximate the assigned labels.

However, computing $\phi(\mathbf{x})'\mathbf{\Phi}$ requires N kernel computations, which is the same as comparing the new point $\mathbf{x}$ to every training example, as in the original nearest-neighbor search. Thus, instead of simplification we actually increased the number of comparisons, i.e. similarity or distance computations. Nevertheless, if we use dot products, that is we handle the points in the *input space* ($\mathbf{\Phi} = \mathbf{X}$), we can compute the r vectors $\mathbf{g}_k := \mathbf{u}_k$ or $\mathbf{g}_k := \mathbf{X}\mathbf{D}^{-1/2}\mathbf{v}_k$, $k = 2, \ldots, r+1$, and, when a new point $\mathbf{x}$ arrives, calculate the dot products $\mathbf{x}'\mathbf{g}_k$ and threshold the resulting values to obtain the binary codeword. Thus, the computation of a new codeword requires only r dot products of input space vectors. Whether to calculate $\mathbf{u}_k$ or $\mathbf{v}_k$ depends on the size and dimensionality of the dataset. For large but relatively low-dimensional data – which we assume is the case – the eigendecomposition of $\mathbf{X}\mathbf{D}^{-1}\mathbf{X}'$ is performed. The codewords assigned to training points are composed by the first r eigenvectors with strictly positive eigenvalues of the normalized Laplacian, computed using the eigenvectors of $\mathbf{X}\mathbf{D}^{-1}\mathbf{X}'$.

Our method has the limitation that the similarity matrix $\mathbf{W} = \mathbf{X}\mathbf{X}'$ must be positive, but in turn offers a simple and fast codeword generation process for previously unseen points. The procedure of computing the codeword of a new point is summarized in Algorithm 1. We emphasize that computing the eigenvectors has to be done just once.

Algorithm 1 Codeword calculation for a new data point in linear spectral hashing

Input: Data $\mathbf{X}$, $\mathbf{D} = \mathrm{diag}(\mathbf{X}'\mathbf{X}\mathbf{1})$, matrices of eigenvectors $\mathbf{V}$ and $\mathbf{U}$, and point $\mathbf{x}$

Output: Codeword $f(\mathbf{x})$

1: Take the first r eigenvectors of $\mathbf{D}^{-1/2}\mathbf{X}'\mathbf{X}\mathbf{D}^{-1/2}$ or $\mathbf{X}\mathbf{D}^{-1}\mathbf{X}'$, starting with the second largest eigenvalue:

$$[\mathbf{v}_2,\ \mathbf{v}_3,\ \ldots,\ \mathbf{v}_{r+1}] \quad \text{or}$$
$$[\mathbf{u}_2,\ \mathbf{u}_3,\ \ldots,\ \mathbf{u}_{r+1}]$$

2: Define the following function:

$$f_i(\mathbf{x}) = \mathbf{x}'\mathbf{u}_i = \mathbf{x}'\mathbf{X}\mathbf{D}^{-1/2}\mathbf{v}_i, \quad i = 2, \ldots, r+1$$

3: The codeword of a previously unseen point is then calculated as

$$f(\mathbf{x}) = \mathrm{sgn}\left([f_2(\mathbf{x}),\ f_3(\mathbf{x}),\ \ldots,\ f_{r+1}(\mathbf{x})]'\right)$$

4.1. Analysis of the algorithm

Random hyperplane-based locality-sensitive hashing [7] generates random normal vectors to separate the points by hyperplanes, thus dividing the input space into slices and assigning binary codewords to the points by the signed distances to the hyperplanes. Linear spectral hashing chooses the hyperplanes in a non-random manner. In this section, we prove the following statement.

Statement 1. *Linear spectral hashing finds a set of orthogonal vectors that are normal vectors of maximum-margin separating hyperplanes.*

In Section 2.2 we showed that the eigenvectors of $\mathbf{X}\mathbf{D}^{-1}\mathbf{X}'$ and $\mathbf{D}^{-1/2}\mathbf{X}'\mathbf{X}\mathbf{D}^{-1/2}$ imply the same solution, therefore – depending on the dataset size and dimensionality of the data – we can work with the appropriate matrix. However, we assume, that the dataset is large – otherwise we would not resort to fast nearest-neighbor search – and the dimensionality is relatively small (e.g. of order of thousands). In this case we use $\mathbf{X}\mathbf{D}^{-1}\mathbf{X}'$ and calculate the first $r+1$ eigenvectors with largest eigenvalues, omitting the first one. Then a point is

mapped to the Hamming space of dimension r by projecting the point onto these eigenvectors and taking the sign of each component.

Problem (3) – with the constraint $\|\mathbf{w}\| = 1$ – is equivalent to searching for a hyperplane that separates points with a maximum margin.[3] We know that the distance between point $\mathbf{x}$ and hyperplane $\{\mathbf{z} \mid \mathbf{w}'\mathbf{z} = 0, \|\mathbf{w}\| = 1\}$ equals $|\mathbf{w}'\mathbf{x}|$. Equation (3) can be rewritten as

$$\mathbf{w}^* = \operatorname*{argmax}_{\mathbf{w} \in X} \sum_{i=1}^{N} D_{ii}^{-1} (\mathbf{w}'\mathbf{x}_i)^2$$

where $(\mathbf{w}'\mathbf{x}_i)^2$ is the squared point–hyperplane distance and D_{ii} is a weighting factor that equals $\|\mathbf{x}_i\| \|\overline{\mathbf{x}}\| \cos\theta_i$, assigning a larger weight to distant points. Here θ_i denotes the angle enclosed by $\mathbf{x}_i$ and the mean data vector $\overline{\mathbf{x}} = \sum_{i=1}^{N} \mathbf{x}_i$. Hence, by solving (3) we obtain a maximum-margin separating hyperplane. Let us rewrite the problem in the following equivalent form:

$$\begin{aligned} \mathbf{w}^* \quad = \quad & \operatorname*{argmax}_{\mathbf{w} \in X} \frac{\mathbf{w}'\mathbf{X}\mathbf{D}^{-1}\mathbf{X}'\mathbf{w}}{\mathbf{w}'\mathbf{w}} \\ & \text{s.t. } \mathbf{w}'\mathbf{X}\mathbf{1} = 0 \end{aligned} \tag{8}$$

The solution is given by the eigenvector of $\mathbf{X}\mathbf{D}^{-1}\mathbf{X}'$ with the second largest eigenvalue. Thus the first vector in linear spectral hashing (i.e. the second eigenvector $\mathbf{u}_2$ of $\mathbf{X}\mathbf{D}^{-1}\mathbf{X}'$) separates the points into two *distant* clusters.

To prove that the subsequent vectors are also normals to maximum-margin separating hyperplanes, we use the following property of the Rayleigh quotient [22, 19]: if $\mathbf{A}$ is a real symmetric matrix, then if we require $\mathbf{u}$ to be orthogonal to the $j-1$ largest eigenvectors (i.e. eigenvectors with largest eigenvalues) $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{j-1}$, the Rayleigh quotient $\frac{\mathbf{u}'\mathbf{A}\mathbf{u}}{\mathbf{u}'\mathbf{u}}$ is maximized by the next largest eigenvector $\mathbf{u}_j$. Although it was not mentioned, this property was also used when choosing the second eigenvector. Thus we proved the statement.

4.2. Connections to Laplacian eigenmaps

The Laplacian eigenmap technique [23] – like Isomap [24] and LLE [25] – is a dimensionality reduction technique that offers an *optimal* low-dimensional

[3] In linear spectral hashing the feature mapping is the identity function, therefore $\mathbf{\Phi} = \mathbf{X}$.

embedding, by optimally preserving the local neighborhood information. Laplacian eigenmaps computes the eigenvectors with the smallest eigenvalues of the $\mathbf{D}^{-1}\mathbf{W}$ normalized Laplacian. Omitting the eigenvectors with eigenvalue zero, the embedding of the points is given by the eigenvector components:

$$\mathbf{x}_i \mapsto [v_{2i}, v_{3i}, \ldots, v_{ri}]'$$

where $\mathbf{v}_i$ is the i-th eigenvector, v_{ki} denotes the k-th component of this vector and r is the embedding dimension.

We know that random walk and symmetric normalized Laplacian have the same spectra and the eigenvectors are connected by $\mathbf{v} = \mathbf{D}^{1/2}\mathbf{u}$, where $\mathbf{u}$ and $\mathbf{v}$ are the eigenvectors of these matrices respectively [18]. Since $\mathbf{D}$ is positive, the eigenvectors have the same sign. Linear spectral hashing therefore uses the signs of the embedding given by Laplacian eigenmaps.

4.3. Connections to spectral clustering

The binary case of spectral clustering was already discussed in Section 2. Having $k > 2$ clusters, we arrive to problem (7) without the first constraint $\mathbf{Y}'\mathbf{1} = \mathbf{0}$. The solution is given by the first k eigenvectors of the Laplacian organized into the columns of $\mathbf{Y}^*$. The eigenvectors thresholded at zero of the Laplacian hold cluster information, therefore in spectral clustering the rows of $\mathbf{Y}^*$ are used to represent points in a lower dimensional space – these are clustered using the k-means algorithm [19, 26, 18].

The first constraint of spectral hashing (7) is introduced to maintain the balance between the bits of hash codewords. This requirement shoots out the first eigenvector of the Laplacian, but the subsequent eigenvectors are common in the solutions.

4.4. Time complexity

The time complexity of linear spectral hashing can be broken down into two parts: training and codeword calculation complexity. By training we refer to the process of learning the binary projections. Suppose the points $\mathbf{x}_i, i = 1, \ldots, N$, are taken from the set $X \subseteq \mathbb{R}^d$. The complexity of the training phase is $O(d^3)$ or $O(N^3)$, depending whether the dimensionality or the size of the dataset is smaller. What is more important is the codeword calculation complexity, since training has to be done only once, but one has to compute the codeword whenever a new point arrives. Because of the dot products, codeword computation can be done in $O(rd)$ time, where r is the dimensionality of the Hamming space.

Dataset	Dimensionality	Training points	Test points	No. of classes
Reuters-21578	5000 (reduced)	9583	3745	90
20Newsgroups	5000 (reduced)	11314	7532	20
OptDigits	64	3823	1797	10
DNA	180	2000	1186	3

Table 1: Datasets used in the experiments.

Spectral hashing has an $O(d^3)$ or $O(N^3)$ training complexity, because of the eigendecomposition of the covariance matrix. The training phase also involves the computation of the analytical eigenfunctions of the one-dimensional Laplacian and a sorting step to find the smallest eigenvalues. The codeword calculation complexity is $O(rd)$.

5. Experimental results

The experiments were run on two corpora frequently used for evaluating text categorization systems – since linear methods perform well on bag-of-words document vectors – as well as on a handwritten digit recognition and a DNA sequence dataset. Since the superiority of spectral hashing to other methods has already been assessed [9], and because our main objective was to prove that the proposed method can outperform spectral hashing where dot products provide good similarity, we only compare our algorithm to spectral hashing.

The first two datasets used were the Reuters-21578 and 20Newsgroups[4] datasets. These datasets contain newswire articles and newsgroup documents respectively, and we use the bag-of-words model with term weighting to represent these as real vectors. For both corpora, a stopword list with 199 elements[5] was used to eliminate the irrelevant words, after which the remaining most frequent 5000 words were selected as the dimensions of the representational space. In the final step the *tf-idf* transformation [27] was applied and the documents were normalized to unit length. No stemming was performed for the selected terms.[6] In Reuters, the distribution of the

[4]The datasets can be downloaded from `http://disi.unitn.it/moschitti/corpora.htm` and `http://qwone.com/~jason/20Newsgroups/20news-bydate.tar.gz`.

[5]The stoplist was taken from WordNet::Similarity 2.05, `http://wn-similarity.sourceforge.net`.

[6]Stemming is the process of reducing the words to their stem and conflating related

documents in the classes is very uneven, ranging from 1650 documents (*acquisitions* class) to a single document per class (e.g. the *castor-oil* class) in the training dataset. The documents in the 20Newgroups dataset are distributed approximately evenly among the classes.

OptDigits[7] is a handwritten digit recognition database in which the examples are 17-color (greyscale) images of handwritten digits (0–9), therefore the attributes are integers between 0 and 16. The training and test examples are distributed approximately equally among the 10 classes.

The DNA Statlog dataset[8] is a database containing DNA sequences (A, C, G, T) for recognizing exon/intron boundaries in order to splice out redundant data. We call exons the segments with *meaning*, while introns are those segments that do not carry relevant information for "producing" the protein. There are three classes: exon/intron sites, intron/exon sites and sequences with no splicing sites. Each sequence is of length 60, starting from position -30 and ending at 30. The categorical attributes are converted to binary using 3 bits for each, thus transforming an example to a sequence of length 180. The training and test examples are distributed approximately 25%–25%–50% among the three classes.

Table 1 lists the datasets and its properties.

For evaluating the performance of the studied methods we used three approaches. The first two of these do not use the classification labels, and measures the precision of finding the true neighbors – for each test document we determined its 50 nearest neighbors from the training set. The evaluation measures used were *precision* and *recall* [27]. We also measured and compared the accuracies of the k-nearest neighbor classifiers built using Euclidean distances between the points and Hamming distances between the generated hash codewords.

In Figure 2(a)–(d) the areas under recall-precision curves are shown for spectral hashing and linear spectral hashing for the datasets by varying the codeword bits from 8 to 256 or the maximum possible value (63 and 179

words to improve on the bag-of-words representation of the document, however its application is controversial, may lead to performance degradation.

[7]Optical Recognition of Handwritten Digits Data Set, `http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits`

[8]The DNA dataset can be found at `http://archive.ics.uci.edu/ml/support/Statlog+Project` or `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html#dna`.

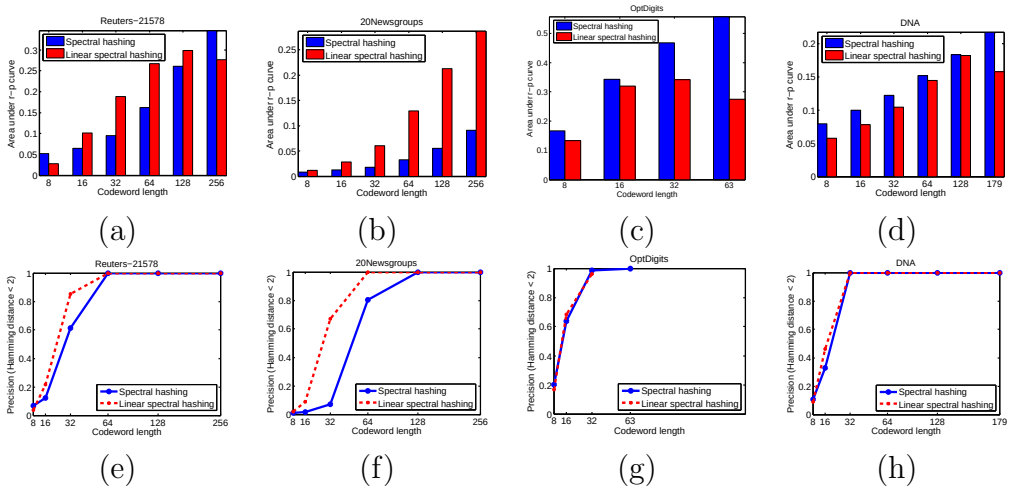


Figure 2: (a)–(d): area under the recall-precision curve as a function of the codeword length, (e)–(h): precision results for Hamming distance < 2.

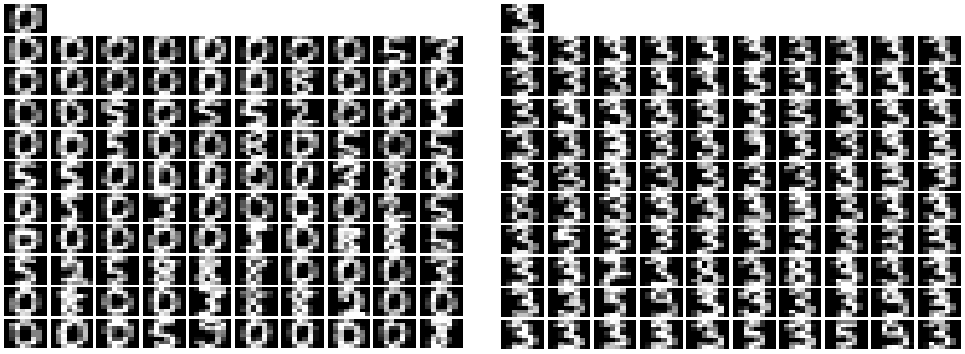
Dataset	k**-NN**	**Spectral hashing**	**Linear spectral hashing**
Reuters-21578	0.7028	0.6510 ($r = 256$)	0.6694 ($r = 256$)
20Newsgroups	0.6217	0.5013 ($r = 256$)	0.5489 ($r = 256$)
OptDigits	0.9755	0.8603 ($r = 16$)	0.8470 ($r = 16$)
DNA	0.8423	0.7546 ($r = 16$)	0.7479 ($r = 16$)

Table 2: Accuracy results obtained for the k-nearest neighbor classifier using Euclidean distances between the points and Hamming distances between the hash codewords; k was set to 10.

for OptDigits and DNA), every time doubling the length. Precision and recall was calculated as the function of the Hamming neighborhood, and the number of evaluations was determined by the length of the codeword. For example, for 64 bits the precisions and recalls were calculated for Hamming distances of $\{0, 1, 2, \ldots, 64\}$. Figure 2(e)–(h) shows the results obtained with spectral hashing and linear spectral hashing respectively, where the precision was plotted for Hamming distance < 2, increasing the number of codeword bits from 8 to 256 or the maximum possible value.[9]

In Table 2 the accuracy results of the k-nearest neighbor classifiers are shown. As a baseline we tested k-NN with Euclidean distances between the

[9]For spectral hashing we used the MATLAB code from `http://www.cs.huji.ac.il/~yweiss/SpectralHashing`.



(a) (b)

Figure 3: Showing the first 100 nearest neighbors found by linear spectral hashing of digits '0' and '3' (upper left corners) from the OptDigits dataset. The codeword length was set to 16.

points and compared the results to those obtained with spectral hashing and linear spectral hashing using Hamming distances between the generated hash codewords. When determining the k-nearest neighbors, if two points had the same distance, the one with the lower index was taken first. For the Reuters-21578 and 20Newsgroups dataset we used a larger codeword length ($r = 256$), because of the higher dimensionality of the data. For the lower dimensional OptDigits and DNA dataset r was set to 16.

In order to visualize the neighbors found using linear spectral hashing, in Figure 3 we showed the first 100 nearest neighbors of two digits (the first and the fourth image of the test data) from the OptDigits dataset. The codeword length was set to 16.

6. Discussion

We presented a method based on spectral hashing but using a different generalization for previously unseen points. This method outputs r input space vectors, which are used to compute the binary hash sequence of a point by calculating dot products as in hyperplane LSH. However, compared to the random hyperplane selection of LSH, linear spectral hashing chooses the hyperplanes in a principled way: it finds a set of orthogonal vectors that are normal vectors of maximum-margin separating hyperplanes.

Based on the results obtained we may say that the proposed method can be used for large and relatively low-dimensional data, where dot products

offer a *good* similarity measure. Experiments on bag-of-words data show that the proposed method outperforms spectral hashing in the [16 .. 128] interval (area under the recall–precision curve), and k-NN with codewords generated by linear spectral hashing yields a better accuracy. For the OptDigits and DNA datasets the results show a slight deterioration of performance. We claim that this is caused by using the linear kernel, because these data desire other feature mappings. For example optical character recognition with non-linear kernels (e.g. second-order polynomial or Gaussian kernel) is known to produce better results [28].

For a better experimental evaluation we plan to test the proposed method on a wider range of datasets. We also plan to compare linear spectral hashing to other unsupervised hashing methods mentioned in Section 1. Studying the possibility of applying arbitrary kernels – for example by using low-dimensional approximations [29, 11] – remains the objective of a future research. Finally, we want to investigate the possible extensions of linear spectral hashing to support semi-supervised learning scenarios.

Acknowledgement

The authors acknowledge the support of the Romanian Ministry of Education and Research via grant PN-II-RU-TE-2011-3-0278. The research was also supported by the Communitas Foundation grant UTA-13/1-0065.

References

[1] T. M. Cover, P. E. Hart, Nearest neighbor pattern classification, IEEE Transactions on Information Theory IT-13.

[2] A. Rajaraman, J. D. Ullman, Mining of massive datasets, Cambridge University Press, 2012.

[3] J. L. Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM 18 (9) (1975) 509–517.

[4] R. Salakhutdinov, G. E. Hinton, Semantic hashing, Int. J. Approx. Reasoning 50 (7) (2009) 969–978.

[5] A. Gionis, P. Indyk, R. Motwani, Similarity search in high dimensions via hashing, in: Proceedings of the 25th International Conference on

Very Large Data Bases (VLDB '99), Morgan Kaufmann, San Francisco, 1999, pp. 518–529.

[6] K. Grauman, R. Fergus, Learning binary hash codes for large-scale image search, Machine Learning for Computer Vision 411 (2013) 49–87.

[7] M. Charikar, Similarity estimation techniques from rounding algorithms, in: STOC, 2002, pp. 380–388.

[8] B. Kulis, K. Grauman, Kernelized locality-sensitive hashing for scalable image search, in: ICCV, IEEE, 2009, pp. 2130–2137.

[9] Y. Weiss, A. B. Torralba, R. Fergus, Spectral hashing, in: NIPS, MIT Press, 2008, pp. 1753–1760.

[10] M. Raginsky, S. Lazebnik, Locality-sensitive binary codes from shift-invariant kernels, in: Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, A. Culotta (Eds.), NIPS, Curran Associates, Inc, 2009, pp. 1509–1517.

[11] A. Rahimi, B. Recht, Random features for large-scale kernel machines, in: J. C. Platt, D. Koller, Y. Singer, S. T. Roweis (Eds.), NIPS, 2007.

[12] D. Zhang, J. Wang, D. Cai, J. Lu, Self-taught hashing for fast similarity search, in: F. Crestani, S. Marchand-Maillet, H.-H. Chen, E. N. Efthimiadis, J. Savoy (Eds.), SIGIR, ACM, 2010, pp. 18–25.

[13] D. Zhang, J. Wang, D. Cai, J. Lu, Laplacian co-hashing of terms and documents, in: C. Gurrin, Y. He, G. Kazai, U. Kruschwitz, S. Little, T. Roelleke, S. M. Rüger, K. van Rijsbergen (Eds.), ECIR, Vol. 5993 of Lecture Notes in Computer Science, Springer, 2010, pp. 577–580.

[14] G. Shakhnarovich, P. A. Viola, T. J. Darrell, Fast pose estimation with parameter-sensitive hashing, in: ICCV, 2003, pp. 750–757.

[15] P. Jain, B. Kulis, K. Grauman, Fast image search for learned metrics, in: CVPR, IEEE Computer Society, 2008.

[16] J. Wang, O. Kumar, S.-F. Chang, Semi-supervised hashing for scalable image retrieval, in: CVPR, IEEE, 2010, pp. 3424–3431.

[17] J. Wang, S. Kumar, S.-F. Chang, Semi-supervised hashing for large-scale search, IEEE Trans. Pattern Anal. Mach. Intell 34 (12) (2012) 2393–2406.

[18] U. von Luxburg, A tutorial on spectral clustering, Statistics and Computing 17 (4) (2007) 395–416.

[19] J. Shi, J. Malik, Normalized cuts and image segmentation, in: CVPR, IEEE Computer Society, 1997, pp. 731–737.

[20] A. Rahimi, B. Recht, Clustering with normalized cuts is clustering with a hyperplane, in: Statistical Learning in Computer Vision, 2004.

[21] H. Lütkepohl, Handbook of matrices, John Wiley & Sons Ltd., Chichester, 1996.

[22] G. H. Golub, C. F. van Loan, Matrix computations, 3rd Edition, Johns Hopkins University Press, 1996.

[23] M. Belkin, P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, Neural Computation 15 (6) (2003) 1373–1396.

[24] J. B. Tenenbaum, V. de Silva, J. C. Langford, A global geometric framework for nonlinear dimensionality reduction, Science 290 (5500) (2000) 2319–2323.

[25] S. T. Roweis, L. K. Saul, Nonlinear dimensionality reduction by locally linear embedding, Science 290 (5500) (2000) 2323–2326.

[26] A. Ng, M. Jordan, Y. Weiss, On Spectral Clustering: Analysis and an algorithm, in: T. Dietterich, S. Becker, Z. Ghahramani (Eds.), Advances in Neural Information Processing Systems, MIT Press, 2001, pp. 849–856.

[27] F. Sebastiani, Machine learning in automated text categorization, ACM Computing Surveys 34 (1) (2002) 1–47.

[28] B. Schölkopf, A. J. Smola, Learning with Kernels, The MIT Press, Cambridge, MA, 2002.

[29] A. M. Frieze, R. Kannan, S. Vempala, Fast monte-carlo algorithms for finding low-rank approximations, in: FOCS, IEEE Computer Society, 1998, pp. 370–378.