Machine Learning (2022) 111:2297–2322
https://doi.org/10.1007/s10994-021-06124-1

Fast spectral analysis for approximate nearest neighbor search

Jing Wang[1] · **Jie Shen**[2]

Received: 4 March 2020 / Revised: 23 September 2021 / Accepted: 3 November 2021 /
Published online: 7 January 2022

Abstract
In large-scale machine learning, of central interest is the problem of approximate nearest neighbor (ANN) search, where the goal is to query particular points that are close to a given object under certain metric. In this paper, we develop a novel data-driven ANN search algorithm where the data structure is learned by fast spectral technique based on s landmarks selected by approximate ridge leverage scores. We show that with overwhelming probability, our algorithm returns the $(1+\epsilon/4)$-ANN for any approximation parameter $\epsilon \in (0,1)$. A remarkable feature of our algorithm is that it is computationally efficient. Specifically, learning k-length hash codes requires $O((s^3+ns^2)\log n)$ running time and $O(d^2)$ extra space, and returning the $(1+\epsilon/4)$-ANN of the query needs $O(k\log n)$ running time. The experimental results on computer vision and natural language understanding tasks demonstrate the significant advantage of our algorithm compared to state-of-the-art methods.

Keywords Approximate nearest neighbor search · Spectral analysis · Hashing · Noise · Subspace

1 Introduction

Nearest neighbor search is one of the most fundamental problems in computational geometry and machine learning. It has been broadly investigated in a large body of real-world scenarios such as data compression (Gersho and Gray, 2012), speech recognition (Makhoul et al., 1985), and information retrieval (Jegou et al., 2011). As a concrete example, for

Editor: Gustavo Batista.

✉ Jing Wang
jjinw@amazon.com

Jie Shen
jie.shen@stevens.edu

[1] Amazon, New York City, NY, USA

[2] Stevens Institute of Technology, Hoboken, NJ, USA

customers without any shopping history, it is often plausible to look up the customers in the database with similar profiles to help make recommendation on the items.

There are many early works for (exact) nearest neighbor search, such as k-d tree and R-tree (Bentley, 1975; Samet, 1990, 2006). These methods perform very well when the data lie in a low-dimensional space, say three dimensions, while suffering computational intractability in a high-dimensional space (Arya et al., 1995). In fact, an early attempt from Dobkin and Lipton (1976) provided the first algorithm for nearest neighbor search in d-dimensional space which takes double-exponential time of $O(n^{2^{d+1}})$ for preprocessing and $O(2^d \log n)$ for retrieval. Such problem is known as the curse of dimensionality, and to tackle the problem in high dimensions, the notion of *approximate* nearest neighbor was proposed as a practical alternative (Arya and Mount, 1993). To be a little formal, given any approximation factor $\epsilon > 0$, we say that a point $\boldsymbol{p}$ is an ϵ-nearest neighbor of a given query $\boldsymbol{q}$ if the ratio of distances from $\boldsymbol{p}$ to $\boldsymbol{q}$ and from $\boldsymbol{q}$ to its exact nearest neighbor is at most $(1+\epsilon)$.

We consider the data in real-world applications which are usually perturbed with noise (Abdullah et al., 2014). Formally, the observed data set $\boldsymbol{P} = \{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_n\}$ is generated by a clean data set $\boldsymbol{X} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n\}$ with random noise corruption, that is

$$\boldsymbol{p}_i = \boldsymbol{x}_i + \boldsymbol{t}_i, \ \forall\, i = 1, \cdots, n. \tag{1}$$

The query $\boldsymbol{q}$ is a superposition of the clean query point $\boldsymbol{y}$ corrupted by the same type of noise, i.e., $\boldsymbol{q} = \boldsymbol{y} + \boldsymbol{t}$. Suppose $\boldsymbol{x}^*$ is the (exact) nearest neighbor of $\boldsymbol{y}$, that is

$$\|\boldsymbol{y} - \boldsymbol{x}^*\|_2 \le 1 \text{ and } \forall \boldsymbol{x} \in \boldsymbol{X} \setminus \{\boldsymbol{x}^*\}, \|\boldsymbol{y} - \boldsymbol{x}\|_2 \ge 1 + \epsilon, \tag{2}$$

where $\|\cdot\|_2$ denotes the ℓ_2-norm. We will consider that the noise is bounded, in the sense that $\max\{\|\boldsymbol{t}_i\|_2, \|\boldsymbol{t}\|_2\} \le \epsilon/16$. Though it seems that the most natural assumption on the noise is Gaussian, we note that both Gaussian and bounded random variables are sub-Gaussian. So they admit the same tail bound. Under this smoothed problem setting, Indyk and Motwani (1998) proposed the celebrated locality-sensitive hashing (LSH) algorithm that achieves sub-linear query time. Under the locality-sensitive hashing framework, there have been a large body of works showing efficient computation is possible (Andoni and Indyk, 2008; Andoni et al., 2014, 2018). Notably, the construction of the hashing functions in LSH is independent of the data.

On the other spectrum, algorithms that incorporate machine learning techniques to *learn* the hash functions from the data have attracted a lot of interest in recent years (Kulis and Darrell, 2009; Liu et al., 2011; Kong and Li, 2012). For example, spectral graph has been widely studied to learn the binary codes that preserve the similarity structure of the database (Weiss et al., 2009; Abdullah et al., 2014). Supervised hashing methods learn the binary code representations of samples that are correlated with their labels (Shen et al., 2015). Recent works on representation learning using deep neural networks have shown practical values in various tasks, which motivates a surge of works to utilize convolutional neural networks as hash functions; see, for example, Çakir et al. (2018).

Though the learning based approaches outperform the locality-sensitive hashing based methods in many applications (Jegou et al., 2011; Xia et al., 2015), there seems a lack of theoretical understanding of the success of many of the existing algorithms. In this paper, we propose a data-dependent learning algorithm for approximate nearest neighbor search, and we aim to resolve two important technical barriers: (1) approximate the low-dimensional space efficiently; and (2) provide the theoretical guarantee that the mutual distance is preserved in the low-dimensional space. That is, if the data points are neighbors in the

original space, they should be close to each other in the low-dimensional space. Abdullah et al. (2014) provided the first justification for this disparity, which directly utilized principal component analysis with preprocessing time $O(nd^2 + d^3)$. In our algorithm, we learn the projection matrix by leverage score based sampling which is more computationally efficient (Alaoui and Mahoney, 2015; Musco and Musco, 2015; Cohen et al., 2016; Musco and Musco, 2017). In addition,it is demonstrated that leverage score based sampling approaches often give the strong provable guarantees for subspace approximation and statistical performance in downstream applications (Alaoui and Mahoney, 2015; Rudi et al., 2015; Gittens and Mahoney, 2016).

1.1 Summary of our contributions

In this work, we present a learning-to-hash algorithm based on ridge leverage score: it produces the hash function provably matching the accuracy of principal component analysis methods and the obtained low-dimensional subspace preserves the geometry structure of the database. The advantage of our method is twofold. First, approximating the low-dimensional space is significantly more efficient than many existing spectral methods (Weiss et al., 2009; Abdullah et al., 2014), as our sampling techniques used for subspace learning is performed on s landmark points. The preprocessing, in particular, takes time $O(n \cdot s^2)$, where $s \ll \min(n, d)$ is the number of landmarks. Second, we show that $(1 + \epsilon/4)$-approximate nearest neighbor of the query can be obtained with high probability.

In terms of empirical results, we evaluate the performance of our algorithm on real-world applications: computer vision and natural language understanding. The experiments are conducted on real-world data sets, including MNIST, Stanford Sentiment Treebank (SST-2) (Socher et al., 2013) (SST-2), Corpus of Linguistic Acceptability (CoLA) (Warstadt et al., 2019), Microsoft Paraphrase Corpus (MRPC) (Dolan and Brockett, 2005), Stanford Question Answering Natural Language Inference Corpus (QNLI) (Rajpurkar et al., 2016), and Glove (Pennington et al., 2014). Our algorithm achieves the best performance with various hash code lengths on all the data sets compared with the state-of-the-art algorithms.

1.2 Roadmap

In Sect. 2, we present a more concrete literature review and state the connection to this work. Section 3 gives the main algorithm with performance guarantee in Sect. 4. A comprehensive empirical study is carried out in Sect. 5. We conclude the paper in Sect. 6. The proof details can be found in the "Appendix".

Notation. We use lowercase letters to denote vectors and capital letters for matrices. For a vector $\boldsymbol{q}$, we denote its ℓ_2-norm by $\|\boldsymbol{q}\|_2$. We reserve $\boldsymbol{P} \in \mathbb{R}^{n\times d}$ for the database with n data points in d-dimensional feature space. We use $p_i^\top \in \mathbb{R}^d$ to denote the i-th row of $\boldsymbol{P}$, that is, the i-th sample in $\boldsymbol{P}$. We use two matrix norms: the *Frobenius norm* and *spectral norm*, defined as $\|\boldsymbol{P}\|_F = \sqrt{\sum_{i=1}^{d} \sigma_i(\boldsymbol{P})^2}$ and $\|\boldsymbol{P}\|_2 = \sigma_1(\boldsymbol{P})$ respectively, where $\sigma_i(\boldsymbol{P})$ represents the singular value of $\boldsymbol{P}$ in descending order $(\sigma_1(\boldsymbol{P}) \geq \sigma_2(\boldsymbol{P}) \geq \cdots \geq \sigma_d(\boldsymbol{P}) \geq 0)$. The distance between a data point $\boldsymbol{q}$ and the subspace $\boldsymbol{U}$ is defined as $d(\boldsymbol{q}, \boldsymbol{U}) := \inf_{\boldsymbol{y}\in\boldsymbol{U}} \|\boldsymbol{q} - \boldsymbol{y}\|_2 = \|\boldsymbol{q} - \boldsymbol{q}_{\boldsymbol{U}}\|_2$, where $\boldsymbol{q}_{\boldsymbol{U}}$ is the orthogonal projection onto the subspace $\boldsymbol{U}$. When we say a subspace is k-dimensional, we mean its intrinsic dimension is k.

Table 1 Summary of state-of-the-art results in terms of space and time bounds for approximate nearest neighbor search. k is the length of hash code, s is the number of landmarks, d is the original feature dimension of database with n data points

Work	Space	Time
Andoni and Indyk (2008)	$n^{1+1/c^2+\mathcal{O}(1)} + dn$	$dn^{1/c^2+\mathcal{O}(1)}$
Andoni et al. (2014)	$n^{1/c^2} + d\log n$	$n^{1+1/c^2} + d\log n$
Andoni et al. (2018)	$d^{\mathcal{O}(1)}n^{1+1/c}$	$d^{\mathcal{O}(1)}n^{1/c}$
Our results	$d^2 + kn$	$(ns^2 + k)\log n$

2 Related works

The core of nearest neighbor search is to find the data point most close to the query in the database, while the approximate nearest neighbor search returns the data points within $(1+\epsilon)dist$ of the query, where $dist$ is the distance between the query and the nearest neighbor. In either category, the search is usually performed on a collection of data points; the process to organize the database into certain data structure is called data processing, which is assumed to be independent of the number of queries. As the straightforward search is brute force which takes $O(n)$ time for 1-dimensional space, more efficient searching algorithms usually construct a data structure to make the query efficient in terms of space and time cost in processing and retrieval. For example, binary search method formed the balanced binary tree with time $O(n\log n)$ and answered the query in $\lfloor \log n \rfloor + 1$ time (Knuth, 1973). A plethora of related algorithms have been proposed in the literature, such as k-d trees, R-trees (Bentley, 1975; Samet, 1990; Sellis et al., 1997; Samet, 2006). These approaches are usually based on computational geometry. However, if the number of dimensions exceeds 20, searching in k-d trees and related structures requires the inspection of a large fraction of the database, thereby doing no better than brute-force linear search Gionis et al. (1999). Therefore, the approximate nearest neighbor search has attracted attention for practical problems with high-dimensional data.

Existing algorithms for approximated nearest neighbor search could be categorized as locality-sensitive-hashing families and learning based hashing, depending on how the data structure is constructed. Indyk and Motwani (1998) introduced the idea of *locality-sensitive hashing*. There are many related works discussing how to chose the parameters L (the number of buckets), r_1 (the radius of the ball centered at $\boldsymbol{q}$) and k (the length of hash code) to achieve the low failure probability guarantee. For example, Andoni and Indyk (2008) proposed an algorithm that utilized linear random projection to reduce the feature dimension to k ($k = O(\log n)$), then the approximated nearest neighbors could be returned in sublinear query time using nearly-linear space. Andoni et al. (2014) proposed a data-dependent hashing function with Johnson-Lindenstrauss dimension reduction procedure and got a better result. Andoni et al. (2018) presented a data structure for general symmetric norms. Very recently, Andoni et al. (2021) showed improved data structures for the high-dimensional approximate nearest neighbor search for ℓ_p distances for large values of p and for generalized Hamming distances. The details of related space and time bounds for Euclidean distance are summarized in Table 1.

Learning based hashing has seen a recent surge of interest (Gong and Lazebnik, 2011; Weiss et al., 2012; Erin Liong et al., 2015; Han et al., 2015; Liu et al., 2016). Much of this

excitement centers around the discovery that these approaches achieve outstanding performance in real-world applications, such as computer vision (Xia et al., 2015) and information retrieval (Jegou et al., 2011). There are some works focusing on supervised binary code projection methods (Liu et al., 2014; Shen et al., 2015). For example, sparse projection (SP) introduced the sparse projections for binary encoding which involved minimizing the distortion and adopted the variable-splitting techniques in optimization (Xia et al., 2015). The spectral analysis based unsupervised methods have attracted a lot of attention since the labeled data is precious. For example, Spectral Hashing utilized a subset of thresholded eigenvectors of the graph Laplacian matrix (Weiss et al., 2009). Iterative quantization (ITQ) proposed an efficient way to find the hash code by minimizing the quantization error of mapping the data to the vertices of a zero-centered binary hypercube (Gong and Lazebnik, 2011). Jegou et al. (2011) decomposed the space into a Cartesian product of low dimensional subspace and the hash code is composed of its subspace quantization indices. Liu et al. (2011) assumed that the data reside in a low-dimensional manifold and proposed a graph-based hashing method. Isotropic hashing (ISO) found the hash projection function with equal variances for different dimensions, called isotropic hashing (Kong and Li, 2012). Multidimensional Spectral Hashing (MDSH) learned the binary codes based on reconstructing the affinity between data points, rather than computing their distances (Weiss et al., 2012). bilinear projection based binary codes (BPBC) learned the similarity-preserving binary codes by compact bilinear projections instead of a single large projection matrix (Gong et al., 2013). The algorithm utilizes a spectral relaxation where the bits are mapped by thresholded eigenvectors of the affinity matrix. Circulant binary embedding (CBE) learned the data-dependent circulant projects by minimizing the objective in original and Fourier domains (Yu et al., 2014). Scalable graph hashing (SGH) is proposed to approximate the whole graph without explicitly computing the similarity graph matrix, but optimizing a sequential learning function to learn the compact hash codes in a bitwise manner (Jiang and Li, 2015). We follow this line of research and propose an inexact spectral analysis for approximate nearest neighbor search. The experimental results demonstrate the superiority of our algorithm compared with the state-of-the-art learning based hashing approaches mentioned in this section.

3 Main algorithm

In this section, we elaborate on our approach, which consists of two steps: Algorithm 1 samples the landmark points for the construction of data structure that can be used for efficient retrieval, and Algorithm 2 performs approximate nearest neighbor search.

3.1 Overview

Our pipeline consists of learning the hash codes and retrieval, where the primary idea is to find a good embedding of all original data points under which the mutual distance is well controlled with overwhelming probability. To this end, it seems that a straightforward approach is to utilizing principal component analysis (PCA). However, it is known that finding the exact principal components is computationally slow for large-scale problems. Therefore, we propose to first select a manageable number of landmark points followed by PCA. The selection process is based on the ridge leverage score which is a good measure of the importance of data points (Alaoui and Mahoney, 2015).

Definition 1 (Ridge leverage score) For any $\lambda > 0$, the λ-ridge leverage score of the row of $\boldsymbol{P} \in \mathbb{R}^{n\times d}$ is defined as:

$$l_i = \boldsymbol{p}_i(\boldsymbol{P}^\top \boldsymbol{P} + \lambda \mathbf{I})^{-1}\boldsymbol{p}_i^\top, \tag{3}$$

where $\mathbf{I}$ is the $n \times n$ identity matrix.

To be more concrete, when constructing the principal components of the training set, our algorithm runs in multiple iterations, where in each iteration a fraction of the training data are sampled and some of them will be selected as landmarks. The low-dimensional subspace is learned based on selected landmarks. The algorithm terminates when all the training data have been evaluated. When a new query comes in, it will be projected onto the learned subspace, through which retrieval is efficient.

Input: $P \in \mathbb{R}^{n\times d}$, k, ε, $\delta \in (0, 1/32)$.
Output: Low-dimensional projection matrix $Z \in \mathbb{R}^{d\times k}$, data partitions $\{\mathcal{J}_1, \cdots, \mathcal{J}_T\}$.
$\Omega = \{1, \cdots, n\}$, $T = O(\log n)$ the number of iterations.
while $n/2^T < 192\log(1/\delta)$ **do**
  $T \leftarrow T - 1$
end
Phase I
$\mathcal{J}_0 \leftarrow$ randomly sample $n/2^{T+1}$ samples from Ω.
$\tilde{S} = \{e_i : i \in \mathcal{J}_0\}$, $S = \tilde{S}$, $t = 1$.
while $\Omega \neq \emptyset$ **do**
  $\bar{S} = \{e_i, \forall\, i \in \mathcal{J}_{t-1}\}$.
  $\hat{S} = \bar{S} \cdot \tilde{S}$, $S \leftarrow \emptyset$.
  Approximate the ridge leverage scores of samples selected by $\hat{S}$ with $\lambda = \frac{\varepsilon}{k}\sum_{i=k+1}^{n}\sigma_i(PP^\top)$,

$$\tilde{l}_i = p_i(P^\top \hat{S}\hat{S}^\top P + \lambda \mathbf{I})^{-1}p_i^\top, \ \forall e_i \in \hat{S}.$$

  for *each sample* p_i *indexed by* $\hat{S}$ **do**
    Compute the selection probability η_i:

$$\eta_i = \min(1, 16\tilde{l}_i \log(\sum_i \tilde{l}_i/\delta)).$$

    Append $\frac{1}{\sqrt{\eta_i}}e_i$ into S with probability η_i.
  end
  $\tilde{S} \leftarrow S$.
  if $|\mathcal{J}_{t-1}| > |\Omega|$ **then**
    $\mathcal{J}_t \leftarrow \mathcal{J}_{t-1} \cup \Omega$;
  else
    $\mathcal{J}_t \leftarrow \mathcal{J}_{t-1} \cup \{$randomly select $|\mathcal{J}_{t-1}|$ samples from $\Omega\}$;
  end
  $\Omega = \Omega \setminus \mathcal{J}_t$.
end
Phase II
$\Sigma_k \in \mathbb{R}^{n\times k} \leftarrow$ top k eigenvectors of $PP^\top S$.
$X \in \mathbb{R}^{s\times n} \leftarrow (S^\top PP^\top S)^\dagger S^\top P\Sigma_k$.
return $Z \in \mathbb{R}^{d\times k} \leftarrow P^\top SX$.

Algorithm 1: Learning to Hash by Approximate Leverage Score Sampling.

3.2 Learning to hash

Algorithm 1 learns a low-dimensional projection matrix $\boldsymbol{Z} \in \mathbb{R}^{d\times k}$ that can be applied to embed the data. It consists of two major steps: Phase I selects the landmark points as indicated by the matrix $\tilde{\boldsymbol{S}} \in R^{d\times s}$, and Phase II runs PCA on selected landmark points to return the low-dimensional projection matrix. The algorithm starts with checking if the problem is in large scale, that is, whether the number of samples in $\boldsymbol{P}$ is greater than $192\log(1/\delta)$. If not, we could use PCA to get $\boldsymbol{Z}$ directly; otherwise, the algorithm enters Phase I in the while loop to sample important data points.

The key observation of our sampling approach is that uniform sampling is practical, but it only enjoys theoretical guarantees under strong regularity or incoherence assumptions on the data (Gittens, 2011). On the other hand, ridge leverage scores evaluate the importance of data points which have shown practical impact in downstream applications. However, the calculation of exact ridge leverage score is often slow. In this regard, we propose to combine these two widely used schemes.

First, note that we aim to approximately estimate the ridge leverage score of all data points in an iterative manner, and each data point is evaluated only once. To this end, the number of iterations T is initialized as $O(\log n)$. In each iteration, we randomly draw half of the data points that have not been accessed. The iteration terminates when the size of the remaining data is less than $192\log(1/\delta)$.

In particular, in each iteration, we construct uniform sampling matrix $\bar{\boldsymbol{S}}$ by selecting data points uniformly at random with probability 1/2, and $\tilde{\boldsymbol{S}}$ is the sampling matrix learned by approximated ridge leverage scores. Each column of $\tilde{\boldsymbol{S}}$ has one nonzero element that indicates the index of selected sample. In each iteration, we uniformly sample a subset $\mathcal{J}_i$ and approximate the ridge leverage score of the j-th sample as

$$\tilde{l}_i = \boldsymbol{p}_i(\boldsymbol{P}^\top\tilde{\boldsymbol{S}}\tilde{\boldsymbol{S}}^\top\boldsymbol{P} + \lambda\mathbf{I})^{-1}\boldsymbol{p}_i^\top. \tag{4}$$

Equation (4) is a good approximation of the original ridge leverage score computed as in Definition 1. With the fact $\boldsymbol{P}^\top\tilde{\boldsymbol{S}}\tilde{\boldsymbol{S}}^\top\boldsymbol{P} \preceq \boldsymbol{P}^\top\boldsymbol{P}$, $\tilde{l}_i$ is an upper bound of the ridge leverage score l_i, i.e.

$$\tilde{l}_i = \boldsymbol{p}_i(\boldsymbol{P}^\top\tilde{\boldsymbol{S}}\tilde{\boldsymbol{S}}^\top\boldsymbol{P} + \lambda\mathbf{I})^{-1}\boldsymbol{p}_i^\top \geq \boldsymbol{p}_i(\boldsymbol{P}^\top\boldsymbol{P} + \lambda\mathbf{I})^{-1}\boldsymbol{p}_i^\top = l_i. \tag{5}$$

Then we compute the sampling probability of data points based on the approximated leverage score as follows:

$$\eta_i = \min(1, 16\tilde{l}_i\log(\sum_i \tilde{l}_i/\delta)). \tag{6}$$

The data points are selected as landmarks with probability η_i. The corresponding column of the landmark point in the sampling matrix $\boldsymbol{S}$ is weighted by $1/\sqrt{\eta_i}$. $\boldsymbol{S}$ is assigned to $\tilde{\boldsymbol{S}}$ as the final selected sampling matrix in the current iteration. Then we get the next round data partition $\mathcal{J}_t$ by uniform sampling. We output a partition of data set $\{\mathcal{J}_1, \cdots, \mathcal{J}_T\}$ at the end of the algorithm.

Phase II seeks for a low-dimensional projection matrix $\boldsymbol{Z}$ based on the selected landmarks. A straightforward approach to learn $\boldsymbol{Z}$ is to optimize the following objective function:

$$\min_{\boldsymbol{Z}} \left\| \boldsymbol{P} - \boldsymbol{P}\boldsymbol{Z}\boldsymbol{Z}^{\top} \right\|_F^2. \tag{7}$$

As $\boldsymbol{Z}$ always lies in the column span of $\boldsymbol{P}^{\top}$, it can be represented by constructing a matrix $\boldsymbol{Y} \in \mathbb{R}^{n\times k}$, such that $\boldsymbol{Z} = \boldsymbol{P}^{\top}\boldsymbol{Y}$. We re-parameterize by writing $\boldsymbol{Y} = \boldsymbol{K}^{-1/2}\boldsymbol{W}$ where $\boldsymbol{K} = \boldsymbol{P}\boldsymbol{P}^{\top}$, thus $\boldsymbol{Z} = \boldsymbol{P}\boldsymbol{K}^{-1/2}\boldsymbol{W}$. Recall that Phase I selects s landmarks denoted by $\boldsymbol{S}$. Let Φ be the orthogonal projection onto the row span of $\boldsymbol{S}^{\top}\boldsymbol{P}$. We can approximate the database matrix as $\tilde{\boldsymbol{P}} \overset{\text{def}}{=} \boldsymbol{P}\Phi$, where $\Phi = \boldsymbol{P}^{\top}\boldsymbol{S}(\boldsymbol{S}^{\top}\boldsymbol{P}\boldsymbol{P}^{\top}\boldsymbol{S})^{+}\boldsymbol{S}^{\top}\boldsymbol{P}$. Since Φ is an orthogonal projection, $\Phi\Phi^{\top} = \Phi^2 = \Phi$, we can approximate $\boldsymbol{K}$ as $\tilde{\boldsymbol{K}} = \tilde{\boldsymbol{P}}\tilde{\boldsymbol{P}}^{\top} = \boldsymbol{K}\boldsymbol{S}(\boldsymbol{S}^{\top}\boldsymbol{K}\boldsymbol{S})^{+}\boldsymbol{S}^{\top}\boldsymbol{K} = \boldsymbol{P}$. So, the projection matrix is in the form $\boldsymbol{Z} = \Phi\boldsymbol{P}\tilde{\boldsymbol{K}}^{-1/2}\tilde{\boldsymbol{W}} = \boldsymbol{P}^{\top}\boldsymbol{S}(\boldsymbol{S}^{\top}\boldsymbol{P}\boldsymbol{P}^{\top}\boldsymbol{S})^{+}\boldsymbol{S}^{\top}\boldsymbol{P}\tilde{\boldsymbol{W}}$, where $\tilde{\boldsymbol{W}}$ minimizes the following function:

$$\mathrm{tr}\,(\tilde{\boldsymbol{K}}) - \mathrm{tr}\,(\boldsymbol{W}\boldsymbol{W}^{\top}\tilde{\boldsymbol{K}}\boldsymbol{W}\boldsymbol{W}^{\top}). \tag{8}$$

The optimization of (7) is equivalent to minimizing the above objective function which is standard in the literature (Woodruff et al., 2014). Since $\boldsymbol{W}$ can be taken as the top k eigenvectors of $\boldsymbol{K}$, we approximate it by performing singular value decomposition on $\boldsymbol{P}\boldsymbol{P}^{\top}\boldsymbol{S}$ and assign it to Σ_k in Algorithm 1.

3.3 Retrieval

In the retrieval phase, it is easy to learn the hash code of the data points by the projection matrix $\boldsymbol{Z}$, that is, the hash code of data point $\boldsymbol{p}^{\top} \in \mathbb{R}^d$ is $h(\boldsymbol{p}) = \mathrm{sign}(\boldsymbol{p}^{\top} \times \boldsymbol{Z})$. We can get the hash code of the query in the same way. The near neighbors of the query include the data points that conflict with the query in terms of the hash code. The neighbors could also be retrieved with Hamming distance within certain radius. The search procedure is performed on each data subset of $\{\mathcal{J}_1, \cdots, \mathcal{J}_T\}$ in parallel.

As shown in Algorithm 2, we set m as the desired number of approximate nearest neighbors to return. First, we learn the hash code of the data points in $\boldsymbol{P}$ and the query data point by projection matrix $\boldsymbol{Z}$. Then the data points conflict with the query is considered as the near neighbors of the query point. As the data set $\boldsymbol{P}$ is partitioned to several data subsets $\{\mathcal{J}_i\}_i^T$ ($T = O(\log n)$). The search in each subset could be implemented simultaneously. The search procedure terminates when the desired number of neighbors are returned. We ensure that the approximated nearest neighbor can be returned in low-dimensional query with high probability as shown in Theorem 3.

Input: A query point $q \in \mathbb{R}^d$, m (number of approximated nearest neighbors), the data partition $\{\mathcal{J}_1, \cdots, \mathcal{J}_T\}$, projection matrix $Z \in \mathbb{R}^{d\times k}$ from Algorithm 1.
Output: data set C consists of K (or less) neighbors of q.
Learn the hash code of $q \in \{\mathcal{J}_1, \cdots, \mathcal{J}_T\}$ and q' by projection matrix Z.
$C \leftarrow \emptyset$.
while $|C| < m$ **do**
  for $i = 1, \cdots, T$ **do**
    $C \leftarrow C \cup \{p \in \mathcal{J}_i | p$ conflicts with q' in the k-dimensional subspace$\}$.
  end
end

Algorithm 2: Approximate Nearest Neighbor Query

3.4 Time and memory cost

Phase I in Algorithm 1 performs at most $T = O(\log n)$ iterations in total. After $O(\log n)$ iterations, all data points will be identified by certain group. The time cost in the iterative procedure is dominated by the ridge leverage score computation which takes $O(ns^2)$ time. Since n is cut in half at each level of iteration, the total run time is $O(ns^2 + \frac{ns^2}{2} + \frac{ns^2}{4} + \cdots) = O(ns^2)$. The computation of top k eigenvectors of $\boldsymbol{P}\boldsymbol{P}^\top \boldsymbol{S}$ is $O(ns^2)$. Since $\boldsymbol{S}$ has $O(\frac{k}{\epsilon}\log\frac{k}{\delta\epsilon})$ columns, the computation of eigenvectors to get a low-dimensional projection matrix $\boldsymbol{Z}$ can be performed very efficiently. The construction of $\boldsymbol{Z}$ takes $O(s^3 + s^2)$. Hence, the total time complexity of Algorithm 1 is $O((s^3 + ns^2)\cdot \log n)$. Recall that the time cost of spectral analysis is usually polynomial in n or d, such as $O(nd^2 + d^3)$ (Abdullah et al., 2014). Clearly, our algorithm is more efficient. In terms of memory cost, the storage of $\boldsymbol{P}^\top \boldsymbol{P}$ requires $O(d^2)$ extra space which will be used in the ridge leverage score estimation. To search for the neighbors of the query data point as in Algorithm 2, it requires saving all the binary code of training data with space $O(nk)$ and query time $O(k \cdot \log n)$.

3.5 Hyper-parameter setting

Algorithm 1 learns low-dimensional projection matrix $\boldsymbol{Z} \in \mathbb{R}^{d\times k}$, where d is the feature dimension of data $\boldsymbol{P}$ and k is the dimension of projected space, $k < d$. The while-loop in Phase I terminates in $T = O(\log n)$ iterations as the uniform sampling will select a half of samples at each iteration from Ω. We assume that data $\boldsymbol{P}$ lives in low-dimensional space and k is rank of the data matrix. After data projection, we utilize sign function to get the hash code, hence k equals the length of hash code. The parameter k is tuned in the range of $[0, d]$. The input parameters of Algorithm 1 $\lambda = \frac{\epsilon}{k}\sum_{i=k+1}^{n}\sigma_i(\boldsymbol{K})$, ϵ, δ which are used to get sampling matrix $\boldsymbol{S} \in \mathbb{R}^{n\times s}$, s is the number of sampled data points which is in the order of $\frac{k}{\epsilon}\log\frac{k}{\delta\epsilon}$. The reason is that $s \le 2\sum_i \eta_i$ with probability $1-\delta$ by following Lemma 6. If the ridge leverage score is computed exactly, we bound $\sum_i l_i \le \frac{2k}{\epsilon}$ as shown in Lemma 9 of "Appendix A". Accordingly, $\sum_i \eta_i \le 32\frac{k}{\epsilon}\log\frac{k}{\delta\epsilon}$ as designed.

If the number of data points $n < 192\log(1/\delta)$, the while-loop is skipped. The $192\log(1/\delta)$ number of samples is set following the simplified Chernoff bounds in (Mitzenmacher and Upfal, 2017). That is, when $n \ge 192\log(1/\delta)$, $\mathbb{E}|\bar{S}| \ge 96\log(1/\delta)$, we have:

$$\Pr(1 \le |\bar{S}| \le 0.56n) \ge 1 - \delta, \tag{9}$$

as long as $\delta \leq 1/32$. Then the while-loop continues on the index set Ω of size ≥ 1 and $\leq 0.56n$. Accordingly, Theorem 1 holds for all data set $\mathcal{J}$ with size between 1 and $n-1$ with probability $1-\delta$.

λ is the parameter to approximate ridge leverage score which is initialized as $\frac{\epsilon}{k}\sum_{i=k+1}^{n}\sigma_i(\boldsymbol{P}\boldsymbol{P}^\top)$. Then we get the $(1+2\epsilon)$ relative Frobenius error guarantee among the approximated low-rank space and ground-truth. The quantity $192\log(1/\delta)$ is the minimum number of sampled set to compute leverage score. We assume that the number of samples in $\boldsymbol{P}$ is larger than $192\log(1/\delta)$, otherwise the low-rank matrix could be computed by singular value decomposition directly.

4 Performance guarantee

In this section, we use the following notations. Let $\boldsymbol{S}^\top\boldsymbol{P}$ denote the data matrix with s samples selected by weighted sampling matrix $\boldsymbol{S}$ from database $\boldsymbol{P}$. We write $\boldsymbol{K}=\boldsymbol{P}\boldsymbol{P}^\top \in \mathbb{R}^{n\times n}$. Note that the Nyström approximation of $\boldsymbol{K}$ based on $\boldsymbol{S}$ is $\tilde{\boldsymbol{K}}=\boldsymbol{K}\boldsymbol{S}(\boldsymbol{S}^\top\boldsymbol{K}\boldsymbol{S})^{+}\boldsymbol{S}^\top\boldsymbol{K}$.

Lemma 1 *For any $\delta \in (0,1/32)$, with probability $(1-3\delta)$, Algorithm 1 returns $\boldsymbol{S}$ with s columns that satisfies*:

$$\frac{1}{2}(\boldsymbol{P}^\top\boldsymbol{P}+\lambda\mathbf{I}) \preceq (\boldsymbol{P}^\top\boldsymbol{S}\boldsymbol{S}^\top\boldsymbol{P}+\lambda\mathbf{I}) \preceq \frac{3}{2}(\boldsymbol{P}^\top\boldsymbol{P}+\lambda\mathbf{I}).$$

We remark that Lemma 1 is a direct corollary of Lemma 6 and matrix Bernstein inequality.

Lemma 2 *For any $\delta \in (0,1/32)$, let $\boldsymbol{S}\in\mathbb{R}^{n\times s}$ be returned by Algorithm 1 with $s \leq 384\cdot\mu\log(\mu/\delta)$, where $\mu = \mathrm{tr}\,(\boldsymbol{K}(\boldsymbol{K}+\lambda\mathbf{I})^{-1})$ is the effective dimension of $\boldsymbol{K}=\boldsymbol{P}\boldsymbol{P}^\top$ with parameter λ. Denote Nyström approximation of $\boldsymbol{K}$ by $\tilde{\boldsymbol{K}}=\boldsymbol{K}\boldsymbol{S}(\boldsymbol{S}^\top\boldsymbol{K}\boldsymbol{S})^{+}\boldsymbol{S}^\top\boldsymbol{K}$. With probability $1-3\delta$, the following holds*:

$$\tilde{\boldsymbol{K}} \preceq \boldsymbol{K} \preceq \tilde{\boldsymbol{K}}+\lambda\mathbf{I}.$$

Proof By Lemma 1, we get

$$\frac{1}{2}(\boldsymbol{P}^\top\boldsymbol{P}+\lambda\mathbf{I}) \preceq (\boldsymbol{P}^\top\boldsymbol{S}\boldsymbol{S}^\top\boldsymbol{P}+\lambda\mathbf{I}) \preceq \frac{3}{2}(\boldsymbol{P}^\top\boldsymbol{P}+\lambda\mathbf{I}),$$

for a weighted sampling matrix $\boldsymbol{S}$. If we remove the weight from $\boldsymbol{S}$ so that it has all unit entries, by Lemma 5 and Nyström approximation, $\tilde{\boldsymbol{K}}$ satisfies:

$$\tilde{\boldsymbol{K}} \preceq \boldsymbol{K} \preceq \tilde{\boldsymbol{K}}+\lambda\mathbf{I}$$

as claimed. □

Now, we are ready to use Lemma 1 and Lemma 2 to give an efficient method to approximate the principal components of the data matrix $\boldsymbol{P}$.

Theorem 1 *Let $\boldsymbol{S}\in\mathbb{R}^{n\times s}$ returned by Algorithm 1 with $\lambda=\frac{\epsilon}{k}\sum_{i=k+1}^{n}\sigma_i(\boldsymbol{P}\boldsymbol{P}^\top)$ and $\delta\in(0,1/8)$. $\boldsymbol{V}\in\mathbb{R}^{d\times k}$ contains optimal top k row principal components of data matrix $\boldsymbol{P}$.*

From $\boldsymbol{P}^\top \boldsymbol{S}$, *we can compute a matrix* $\boldsymbol{X} \in \mathbb{R}^{n\times s}$ *such that if we set* $\boldsymbol{Z} = \boldsymbol{P}^\top \boldsymbol{S}\boldsymbol{X}$, *with probability* $1-\delta$:

$$\left\|\boldsymbol{P} - \boldsymbol{P}\boldsymbol{Z}\boldsymbol{Z}^\top\right\|_F^2 \leq (1+2\epsilon)\left\|\boldsymbol{P} - \boldsymbol{P}\boldsymbol{V}\boldsymbol{V}^\top\right\|_F^2,$$

with $s = O(\frac{k}{\epsilon}\log\frac{k}{\delta\epsilon})$.

The proof is presented in "Appendix B". In the following, we demonstrate that the nearest neighbor can be retrieved in the learned data structure. To this end, we first show that the nearest neighbor of the query remains consistent even corrupted with noise.

Lemma 3 *If the query* $\boldsymbol{y}$ *and its nearest neighbor* $\boldsymbol{x}^*$ *are corrupted with noise* $\boldsymbol{t}$, *that is*, $\boldsymbol{q} = \boldsymbol{y} + \boldsymbol{t}$, $\boldsymbol{p}^* = \boldsymbol{x}^* + \boldsymbol{t}$, *the nearest neighbor of* $\boldsymbol{q}$ *is* $\boldsymbol{p}^*$.

Proof Recalling that the noise is bounded, that is $\|\boldsymbol{t}\|_2 \leq \alpha$. Hence for all $i \in [0, n]$, we have

$$\left\|\boldsymbol{p}_i - \boldsymbol{x}_i\right\|_2 \leq \|\boldsymbol{t}\|_2 \leq \alpha.$$

By the triangle inequality,

$$\begin{aligned}\|\boldsymbol{q} - \boldsymbol{p}^*\|_2 &\leq \|\boldsymbol{q} - \boldsymbol{y}\|_2 + \|\boldsymbol{y} - \boldsymbol{x}^*\|_2 + \|\boldsymbol{x}^* - \boldsymbol{p}^*\|_2 \\ &\leq \|\boldsymbol{y} - \boldsymbol{x}^*\|_2 + 2\alpha.\end{aligned}$$

Then, for any other data point in the data set $\boldsymbol{P}$, that is $\boldsymbol{p} \in \boldsymbol{P}$ and $\boldsymbol{p} \neq \boldsymbol{p}^*$, we get

$$\|\boldsymbol{q} - \boldsymbol{p}\|_2 \geq \|\boldsymbol{y} - \boldsymbol{x}^*\|_2 + \epsilon - 2\alpha.$$

Then we get the guarantee that distances between data and low-dimensional subspace are bounded. □

Theorem 2 *Let* $\boldsymbol{Z} \in \mathbb{R}^{d\times k}$ *be the projection matrix learned by Algorithm* 1, $\tilde{\boldsymbol{U}}$ *be the corresponding subspace*, *then we have*:

$$\sum_{\boldsymbol{p}\in\boldsymbol{P}_i} d(\boldsymbol{p}, \tilde{\boldsymbol{U}})^2 \leq (1+2\epsilon)\sum_{i=k+1}^{n}\sigma_i(\boldsymbol{P}),$$

where σ_i *is the i-th singular value of* $\boldsymbol{P}$.

Proof Let $\boldsymbol{V} \in \mathbb{R}^{d\times k}$ contain the projection matrix obtained by singular value decomposition of $\boldsymbol{P}$ and $\boldsymbol{U}$ be corresponding k-dimensional subspace. The distance between a data point and subspace can be computed as:

$$\sum_{\boldsymbol{p}\in\boldsymbol{P}} d(\boldsymbol{p}, \boldsymbol{U})^2 = \sum_{\boldsymbol{p}\in\boldsymbol{P}}\inf_{\boldsymbol{w}\in\boldsymbol{U}}\|\boldsymbol{p} - \boldsymbol{w}\|_2^2 = \left\|\boldsymbol{P} - \boldsymbol{P}\boldsymbol{V}\boldsymbol{V}^\top\right\|_F^2.$$

$$\sum_{\boldsymbol{p}\in\boldsymbol{P}} d(\boldsymbol{p}, \tilde{\boldsymbol{U}})^2 = \sum_{\boldsymbol{p}\in\boldsymbol{P}}\inf_{\boldsymbol{w}\in\tilde{\boldsymbol{U}}}\|\boldsymbol{p} - \boldsymbol{w}\|_2^2 = \left\|\boldsymbol{P} - \boldsymbol{P}\boldsymbol{Z}\boldsymbol{Z}^\top\right\|_F^2.$$

Combining with Theorem 1, we show that

$$\left\|\boldsymbol{P}-\boldsymbol{P}\boldsymbol{Z}\boldsymbol{Z}^{\top}\right\|_F^2 \leq (1+2\epsilon)\left\|\boldsymbol{P}-\boldsymbol{P}\boldsymbol{V}\boldsymbol{V}^{\top}\right\|_F^2 \leq (1+2\epsilon)\sum_{i=k+1}^{n}\sigma_i(\boldsymbol{P}),$$

where σ_i is the i-th singular value of $\boldsymbol{P}$. For the case that k is close to the rank of $\boldsymbol{P}$, $\sum_{i=k+1}^{n}\sigma_i$ can be very small.

With Theorem 2, we can easily prove that the similarity among data points is preserved in the projected low-dimensional space as Lemma 4, which we defer to "Appendix C". Then we get our main result Theorem 3, that the nearest neighbor will be returned in the low-dimensional space.

Lemma 4 *Suppose the nearest neighbor of $\boldsymbol{q}$ is $\boldsymbol{p}^*$ in d-dimensional feature space. In the k-dimensional subspace projected by $\boldsymbol{Z}\in\mathbb{R}^{d\times k}$ which is learned by Algorithm* 1*, the nearest neighbor of $\boldsymbol{q}$ is $\boldsymbol{p}^*$.*

Theorem 3 *Algorithm* 2 *returns data point $\boldsymbol{p}^*$ from database $\boldsymbol{P}$ as a $(1+\epsilon/4)$-approximate nearest neighbor of query point $\boldsymbol{q}$.*

Proof Recall that noisy data $\boldsymbol{p}\in\boldsymbol{P}$ is permuted from clean data $\boldsymbol{x}\in\boldsymbol{X}$ with noise $\boldsymbol{t}$ ($\boldsymbol{p}=\boldsymbol{x}+\boldsymbol{t}$), and so is the query data $\boldsymbol{q}$ ($\boldsymbol{q}=\boldsymbol{y}+\boldsymbol{t}$ with $\boldsymbol{y}$ as clean data). Let the nearest neighbor of $\boldsymbol{y}$ be $\boldsymbol{x}^*\in\boldsymbol{X}$ which corresponds to $\boldsymbol{p}^*$ in $\boldsymbol{P}$. Assume that $\boldsymbol{p}^*$ is the returned nearest neighbor of $\boldsymbol{q}$. Fix $\boldsymbol{x}\neq\boldsymbol{x}^*$, using the triangle inequality to write

$$\begin{aligned}\left\|\boldsymbol{x}-\boldsymbol{p}_{\tilde{U}}\right\|_2 &\leq \|\boldsymbol{x}-\boldsymbol{p}\|_2+\left\|\boldsymbol{p}-\boldsymbol{p}_{\tilde{U}}\right\|_2\\ &\leq \|\boldsymbol{t}\|_2+(1+2\epsilon)\left\|\boldsymbol{p}-\boldsymbol{p}_{U}\right\|_2\\ &\leq \alpha+(1+2\epsilon)\sum_{i=k+1}^{n}\sigma_i\leq 3\alpha.\end{aligned}$$

The third inequality is derived from Theorem 2. Following the proof of Theorem 2, $\sum_{i=k+1}^{n}\sigma_i$ can be as small as possible and $\epsilon\in(0,1)$. Here we let $(1+2\epsilon)\sum_{i=k+1}^{n}\sigma_i\leq 2\alpha$ to get the last inequality. Similarly for $\boldsymbol{x}^*$, we have

$$\left\|\boldsymbol{x}^*-\boldsymbol{p}^*_{\tilde{U}}\right\|_2\leq 3\alpha.$$

Using the triangle inequality, we get

$$\begin{aligned}\left\|\boldsymbol{q}-\boldsymbol{p}^*_{\tilde{U}}\right\|_2 &\leq \|\boldsymbol{q}-\boldsymbol{y}\|_2+\|\boldsymbol{y}-\boldsymbol{x}\|_2+\left\|\boldsymbol{x}-\boldsymbol{p}^*_{\tilde{U}}\right\|_2\\ &\leq \|\boldsymbol{y}-\boldsymbol{x}\|_2+3\alpha,\end{aligned}$$

and

$$\begin{aligned}\left\|\boldsymbol{q}-\boldsymbol{p}_{\tilde{U}}\right\|_2 &\geq \|\boldsymbol{y}-\boldsymbol{x}\|_2-\|\boldsymbol{y}-\boldsymbol{q}\|_2-\left\|\boldsymbol{p}_{\tilde{U}}-\boldsymbol{x}\right\|_2\\ &\geq \|\boldsymbol{y}-\boldsymbol{x}\|_2-3\alpha.\end{aligned}$$

With α set as $16/\epsilon$, we can bound $\left\|\boldsymbol{q}-\boldsymbol{p}^*_{\tilde{U}}\right\|_2$, which implies

$$\begin{aligned}\frac{\left\|\boldsymbol{q}-\boldsymbol{p}_{\tilde{U}}\right\|_2}{\left\|\boldsymbol{q}-\boldsymbol{p}^*_{\tilde{U}}\right\|_2} &= \frac{\|\boldsymbol{y}-\boldsymbol{x}\|_2 \pm 4\cdot\alpha}{\|\boldsymbol{y}-\boldsymbol{x}^*\|_2 \pm 4\cdot\alpha} = \frac{\|\boldsymbol{y}-\boldsymbol{x}\|_2 \pm \frac{1}{4}\epsilon}{\|\boldsymbol{y}-\boldsymbol{x}^*\|_2 \pm \frac{1}{4}\epsilon} \\ &\geq \frac{\|\boldsymbol{y}-\boldsymbol{x}\|_2 - \frac{1}{4}\epsilon}{\|\boldsymbol{y}-\boldsymbol{x}^*\|_2 + \frac{1}{4}\epsilon} \geq \frac{\|\boldsymbol{y}-\boldsymbol{x}^*\|_2 + \frac{3}{4}\epsilon}{\|\boldsymbol{y}-\boldsymbol{x}^*\|_2 + \frac{1}{4}\epsilon} \\ &> 1+\tfrac{1}{4}\epsilon.\end{aligned}$$

By using Pythagoras' Theorem (recall both $\boldsymbol{p}_{\tilde{U}}, \boldsymbol{p}^*_{\tilde{U}} \in \tilde{\boldsymbol{U}}$),

$$\frac{\left\|\boldsymbol{q}_{\tilde{U}}-\boldsymbol{p}_{\tilde{U}}\right\|_2^2}{\left\|\boldsymbol{q}_{\tilde{U}}-\boldsymbol{p}^*_{\tilde{U}}\right\|_2^2} = \frac{\left\|\boldsymbol{q}-\boldsymbol{p}_{\tilde{U}}\right\|_2^2 - \left\|\boldsymbol{q}-\boldsymbol{q}_{\tilde{U}}\right\|_2^2}{\left\|\boldsymbol{q}-\boldsymbol{p}^*_{\tilde{U}}\right\|_2^2 - \left\|\boldsymbol{q}-\boldsymbol{q}_{\tilde{U}}\right\|_2^2} > (1+\tfrac{1}{4}\epsilon)^2.$$

Hence, $\boldsymbol{p}^*$ is reported as the nearest neighbor of $\boldsymbol{q}$ in the low-dimensional subspace. □

5 Experiments

In this section, we perform experiments on benchmark data sets to demonstrate the effectiveness of our algorithm. First, we describe our experimental settings.

5.1 Experimental setting

5.1.1 Baseline algorithms

We illustrate the effectiveness of our algorithm by comparing it with the celebrated data-independent hashing algorithm of locality-sensitive hashing (LSH) (Andoni and Indyk, 2008), and state-of-the-art data-dependent algorithms, including anchor graph hashing (AGH) (Liu et al., 2011), circulant binary embedding (CBE) (Yu et al., 2014), iterative quantization (ITQ) (Gong and Lazebnik, 2011), Isotropic hashing (ISO) (Kong and Li, 2012), multidimensional spectral hashing (MDSH) (Weiss et al., 2012), supervised discrete hashing (SDH) (Shen et al., 2015), scalable graph hashing (SGH) (Jiang and Li, 2015), spectral hashing (Weiss et al., 2009), sparse projection (SP) (Xia et al., 2015) and bilinear projection based binary codes (BPBC) (Gong et al., 2013). The parameters are set as suggested in the original works. We refer to our algorithm as Inexact Subspace Analysis for approximate Nearest Neighbor Search (ISANNS).

5.1.2 Data sets

We consider data sets from both computer vision and and natural language processing. In particular, for the computer vision application, we apply all the compared algorithms to the

Table 2 Statistics of experimental data sets. #Train and #Test are the size of training and testing sets, respectively

Data set	Description	Domain	#Train	#Test
MNIST	Handwritten digit images	Images	69k	1k
SST-2	Sentiment with positive or negative value	Movie reviews	67k	872
CoLA	Sentences with grammatical correctness indicator	Linguistic publications	8.5k	1k
MRPC	paraphrase from online news sources	News	3.7k	408
QNLI	Question answering/textual entailment	Wikipedia	105k	5.5k
Glove	vector representations for words	Wikipedia	1183k	10k

handwritten digit recognition data set MNIST[1], which consists of 70,000 digit images. We randomly sample 69,000 images for training and the left 1000 images for test where each image is represented as a 784-dimensional vector (i.e. the raw pixels).

For the natural language processing task, we use four data sets from the GLUE (General Language Understanding Evaluation) benchmark (Wang et al., 2019) and Glove (Pennington et al., 2014), a word representation data set for Wikipedia's entries. The data sets of GLUE benchmark include Stanford Sentiment Treebank (SST-2) (Socher et al., 2013) (SST-2), Corpus of Linguistic Acceptability (CoLA) (Warstadt et al., 2019), Microsoft Paraphrase Corpus (MRPC) (Dolan and Brockett, 2005) and Stanford Question Answering Natural Language Inference Corpus (QNLI) (Rajpurkar et al., 2016). More specifically, SST-2 consists of movie reviews, the sentiment of which is either positive or negative. CoLA consists of English sentences from books and journal articles. The sentences are grammatically acceptable or not. MRPC is formed by sentence pairs from online news sources. The label of the sentence pair represents whether the two sentences is semantically equivalent or not. QNLI is the data set with pairs of a question and the context sentence, the label of which represents whether the context sentence contains the answer to the question. We compute the representations for sentences and paragraphs with sentence transformers (Reimers and Gurevych, 2019) based on pretrained STS (Semantic Textual Similarity) model "stsb-roberta-base"[2]. Each example is represented with a 768-dimensional dense vector. The statistics of data sets is shown in Table 2.

5.1.3 Evaluation metrics

The performance of the methods are evaluated by two common metrics: Hamming distance ranking and hash table lookup. We retrieve the items within Hamming distance 2 and report related precision, recall and mean average precision (MAP). We also return the top 500 retrieved items and report and mean average precision as well as time complexity.

To compute the precision and recall, let k denote the elements within the Hamming radius 2, and n denote the total number of relevant items in the database, then

$$Precision = \frac{\#\text{relevant seen}}{k}, \quad Recall = \frac{\#\text{relevant seen}}{n}. \tag{10}$$

[1] http://yann.lecun.com/exdb/mnist/.

[2] https://github.com/UKPLab/sentence-transformers.

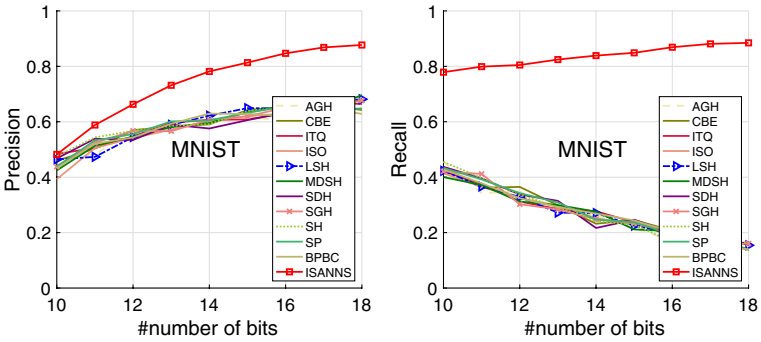

Fig. 1 Performance of precision and recall with the increase of hash code length on MNIST data set

Table 3 Results in terms of MAP of Hamming distance 2 (the column "MAP"), MAP of top 500 samples (the column "MAP@500") and training time (s) on MNIST data set with hash code lengths 10 and 16 bits, respectively

Methods	MAP		MAP@500		Training Time	
	10-bit	16-bit	10-bit	16-bit	10-bit	16-bit
AGH	0.4319	0.4132	0.6074	0.7025	0.6969	0.7018
CBE	0.4422	0.3940	0.5812	0.6881	0.7395	0.7202
ITQ	0.3988	0.4142	0.5410	0.7193	0.7166	0.6550
ISO	0.3965	0.4156	0.5389	0.6966	0.6840	0.6791
LSH	0.4071	0.4022	0.6037	0.6941	0.6548	0.6259
MDSH	0.3997	0.4054	0.6351	0.7178	0.7156	0.6330
SDH	0.4071	0.4149	0.5901	0.7233	0.6669	0.7561
SGH	0.4267	0.4205	0.6006	0.7215	0.6711	0.6423
SH	0.4110	0.4303	0.5974	0.7426	0.6609	0.6657
SP	0.4221	0.4256	0.6310	0.7236	0.6848	0.7342
BPBC	0.4241	0.3958	0.6254	0.7035	0.6560	0.6697
ISANNS	0.7528	0.8843	0.7312	0.8724	0.1054	0.0962

We show the performance with various lengths of hash code.

5.2 Empirical results

Figure 1 shows the precision and recall on the MNIST data set when we tune the hash code length. In terms of precision (left panel), our algorithm always outperforms the baselines, especially when the data are encoded with more bits. Perhaps more surprising, the increase of code length degrades the performance of baseline algorithms, while improves our algorithm. It demonstrates the effective of our algorithm in low-dimensional subspace.

The superiority of our algorithm is outstanding compared with baseline algorithms in terms of both precision and recall in almost all cases. Table 3 lists the hash table lookup results for 10-bit, 16-bit and 20-bit hash codes on MNIST data set. We observe that our

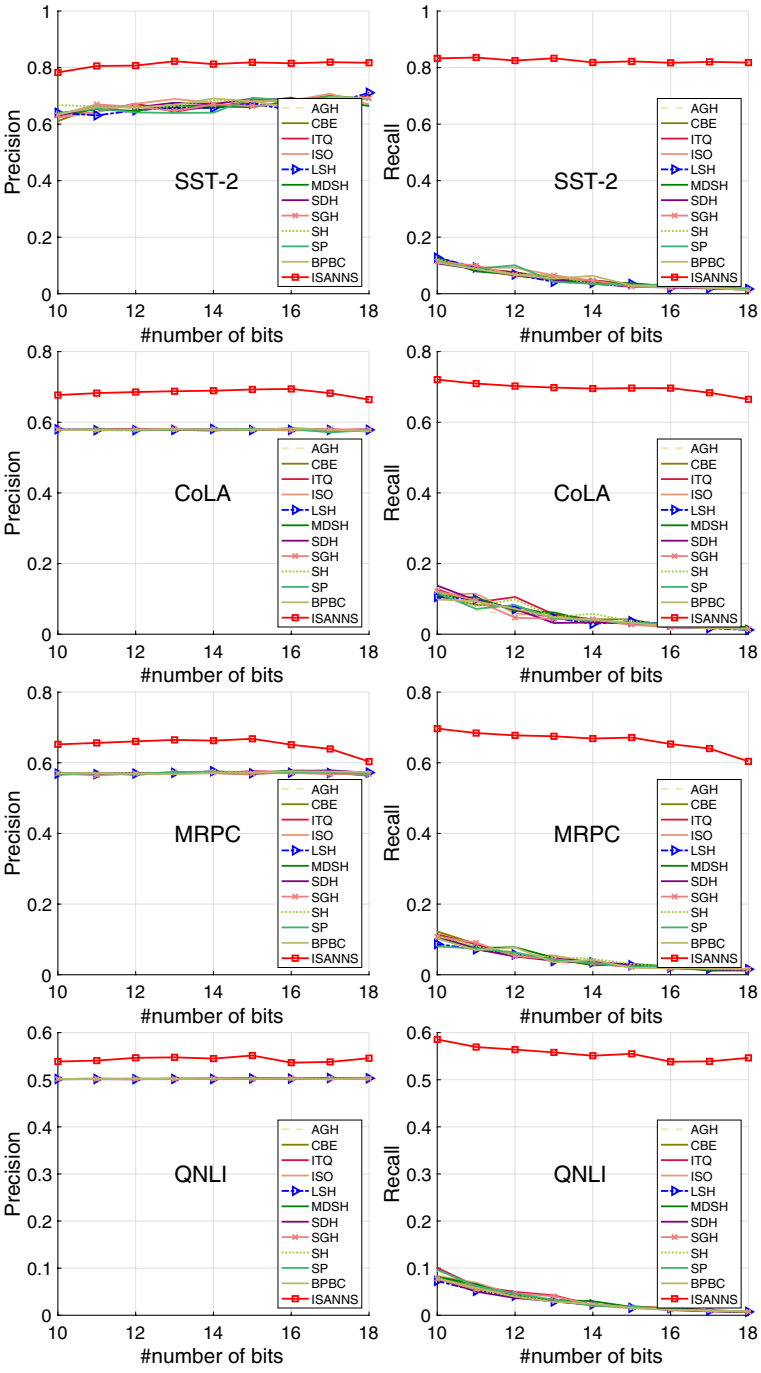

Fig. 2 Performance of precision and recall with the increase of hash code length for GLUE benchmark

Table 4 Results in terms of MAP of Hamming distance 2 for GLUE benchmark with hash code length 10-bit and 16-bit

Methods	SST-2		CoLA		MRPC		QNLI	
	10-bit	16-bit	10-bit	16-bit	10-bit	16-bit	10-bit	16-bit
AGH	0.5619	0.5607	0.5765	0.5776	0.5661	0.5623	0.5007	0.5007
CBE	0.5560	0.5529	0.5766	0.5769	0.5660	0.5650	0.5008	0.5008
ITQ	0.5637	0.5744	0.5769	0.5775	0.5643	0.5625	0.5008	0.5011
ISO	0.5531	0.5534	0.5762	0.5775	0.5626	0.5638	0.5007	0.5007
LSH	0.5754	0.5352	0.5779	0.5769	0.5633	0.5642	0.5008	0.5008
MDSH	0.5593	0.5564	0.5764	0.5778	0.5663	0.5654	0.5008	0.5009
SDH	0.5617	0.5638	0.5763	0.5776	0.5665	0.5642	0.5007	0.5009
SGH	0.5621	0.5614	0.5774	0.5768	0.5660	0.5627	0.5008	0.5009
SH	0.5869	0.5634	0.5765	0.5770	0.5638	0.5649	0.5008	0.5005
SP	0.5708	0.5736	0.5767	0.5769	0.5637	0.5652	0.5006	0.5009
BPBC	0.5687	0.5555	0.5769	0.5779	0.5646	0.5632	0.5010	0.5008
ISANNS	0.8714	0.8796	0.7415	0.7463	0.7257	0.7323	0.6841	0.6787

Table 5 Results in terms of MAP of top 500 retrieved samples for GLUE benchmark with hash code lengths 10-bit and 16-bit

Methods	SST-2		CoLA		MRPC		QNLI	
	10-bit	16-bit	10-bit	16-bit	10-bit	16-bit	10-bit	16-bit
AGH	0.6876	0.7012	0.5751	0.5786	0.5747	0.5756	0.5081	0.5088
CBE	0.6586	0.6981	0.5784	0.5777	0.5766	0.5791	0.5081	0.5083
ITQ	0.6729	0.7113	0.5747	0.5811	0.5749	0.5750	0.5077	0.5099
ISO	0.6707	0.7029	0.5758	0.5805	0.5744	0.5778	0.5086	0.5080
LSH	0.6774	0.6731	0.5780	0.5782	0.5749	0.5766	0.5088	0.5087
MDSH	0.6908	0.6940	0.5753	0.5800	0.5776	0.5800	0.5081	0.5086
SDH	0.6838	0.7078	0.5736	0.5800	0.5770	0.5781	0.5080	0.5096
SGH	0.6891	0.6876	0.5803	0.5784	0.5781	0.5765	0.5075	0.5089
SH	0.6985	0.7012	0.5761	0.5787	0.5759	0.5757	0.5088	0.5077
SP	0.6878	0.7066	0.5765	0.5801	0.5737	0.5749	0.5072	0.5093
BPBC	0.6812	0.6837	0.5774	0.5826	0.5778	0.5760	0.5074	0.5094
ISANNS	0.8188	0.8165	0.6934	0.6952	0.6657	0.6519	0.5484	0.5366

algorithm dramatically outperforms the compared algorithms. Specifically, in terms of 16-bit hash codes on MNIST data set, the MAP of our algorithm is up to 0.8843 while the others are below 0.5 with Hamming radius 2. In terms of MAP with top 500 retrieved data points, our algorithm achieves significant superiority compared with baseline approaches. Our algorithm also enjoys the best time efficiency. With the increase of hash code length, it requires more time to learn the hash codes. With more information, the performance of models is also improved. The experimental results in Figure 1 and Table 3 show the advantage of our algorithm in all cases.

Table 6 Training time cost (s) for GLUE benchmark with hash code length 10-bit and 16-bit

Methods	SST-2		CoLA		MRPC		QNLI	
	10-bit	16-bit	10-bit	16-bit	10-bit	16-bit	10-bit	16-bit
AGH	0.9178	0.8638	0.1596	0.1498	0.1712	0.1057	1.376	1.587
CBE	0.8760	0.8837	0.1395	0.1532	0.0927	0.0908	1.881	1.920
ITQ	0.8642	0.9195	0.2016	0.1469	0.1104	0.1791	2.758	1.778
ISO	0.8705	0.9088	0.1433	0.1675	0.0893	0.1014	2.932	1.959
LSH	0.8803	0.8789	0.1529	0.1447	0.0974	0.0927	1.722	1.915
MDSH	0.9757	0.8795	0.1406	0.1434	0.0906	0.0921	3.070	1.490
SDH	0.8807	0.8962	0.1716	0.1548	0.0881	0.1006	2.460	1.923
SGH	0.8889	0.9262	0.2187	0.1637	0.1114	0.0890	1.749	1.975
SH	0.8805	0.9828	0.1505	0.1334	0.1091	0.1052	2.071	2.046
SP	0.9064	0.8940	0.1574	0.1448	0.0974	0.0980	2.275	2.076
BPBC	0.8736	0.9946	0.1972	0.1681	0.1121	0.0891	3.245	4.341
ISANNS	0.5884	0.6245	0.0483	0.0716	0.0227	0.0261	1.409	1.599

Table 7 Recall and training time cost (s) on Glove with hash code length 8-bit

	CBE	ITQ	ISO	LSH	MDSH	SGH	SP	BPBC	ISANNS
Recall	0.9132	0.9704	0.9902	0.8112	0.8872	0.9810	0.9908	0.9628	1.0
Time cost (s)	174.39	183.91	102.88	75.79	105.40	104.96	106.24	105.99	22.32

Figure 2 shows the precision and recall of compared algorithms on GLUE benchmark with the increase of hash code length. Table 4 and Table 5 show MAP within Hamming radius 2 and MAP of top 500 retrieved samples for 10-bit and 16-bit hash code. It is shown that our algorithm achieves the best performance in all listed GLUE benchmark in almost all the cases.

Table 3 also lists the time cost of learning the hash projection matrix for different methods on MNIST data set referred as "training time". We report the training time cost on GLUE benchmark in Table 6 with hash code length 10-bit and 16-bit. We observe that our algorithm is efficient as the low-rank projection matrix is performed on the sampled matrix instead of the global data matrix. In terms of query time cost, the nearest neighbors in the experiments are computed based on the Hamming distance with radius 2. The dominant query time cost is the computation of the Hamming distance matrix between training data points and the testing data points. Hence, the query time complexity of various methods is same for certain data sets, such as 0.71 s for MNIST, 0.73 s for SST-2, 0.11 s for CoLA, 0.13 s for MRPC, and 10 s for QNLI.

Table 7 presents recall and training time cost of compared algorithms on Glove data set. There are memory issues while implementing AGH, SDH and SH on Glove data set, hence the results of these methods are not included. The experimental results show the advantage of our algorithm in terms of both Recall and training cost. Though SP achieves comparable performance in terms of Recall, our algorithm enjoys higher training efficiency.

In a nutshell, the experimental results on the computer vision and natural language understanding tasks show the practical values of our algorithm.

6 Conclusion

For the approximate nearest neighbor search problem, the high-dimensional and large-scale data raises various challenges. In this paper, we have proposed a spectral analysis for nearest neighbor search method that is based on inexact subspace estimation. Given the data set $\boldsymbol{P} \in \mathbb{R}^{n\times d}$ and the query $\boldsymbol{q}$, we have reduced the feature space of the data from d to k with $k < \log n$. By comparing the time complexity of our method and the spectral analysis based on principal component analysis, it is not hard to see that the computational cost of ours is proportional to ns^2 while that of PCA scales with nd^2. We have further provided the theoretical analysis that the $(1+\epsilon/4)$-approximate neighbors retrieved in the low-dimensional space are the data points close to the query in the original space. The experimental results have shown the significant improvement of our algorithm over state-of-the-art approaches.

Appendix A: Useful lemmas

Note that by Definition 1, the ridge leverage score equals the diagonal entry of $\boldsymbol{P}(\boldsymbol{P}^\top\boldsymbol{P}+\lambda\mathbf{I})^{-1}\boldsymbol{P}^\top$, that is:

$$l_i = (\boldsymbol{P}(\boldsymbol{P}^\top\boldsymbol{P}+\lambda\mathbf{I})^{-1}\boldsymbol{P}^\top)_{i,i}. \tag{11}$$

The computation is expensive when $\boldsymbol{P}$ is a large data matrix. We thus define a sampling matrix $\boldsymbol{S}$ to select some rows from $\boldsymbol{P}$ to approximate $\boldsymbol{P}^\top\boldsymbol{P}$. The approximated ridge leverage score is computed as:

$$\tilde{l}_i = (\boldsymbol{P}(\boldsymbol{P}^\top\boldsymbol{S}\boldsymbol{S}^\top\boldsymbol{P}+\lambda\mathbf{I})^{-1}\boldsymbol{P}^\top)_{i,i}. \tag{12}$$

Leverage score based sampling approaches have long been known to give strong theoretical guarantees for Nyström approximation, and here is the well studied spectral norm guarantee: Lemma 5 is from Gittens and Mahoney (2016) and Lemma 6 is from Musco and Musco (2017).

Lemma 5 *Suppose* $\lambda > 0$, $\delta \in (0, 1/8)$, *the sampling matrix* $\tilde{\boldsymbol{S}} \in \mathbb{R}^{n\times s}$ *is obtained by Algorithm* 1 *with ridge leverage score approximations* $\tilde{l}$ *and data sampling probability* μ, *then with probability* $(1-\delta)$, *the kernel* $\boldsymbol{K} = \boldsymbol{P}\boldsymbol{P}^\top$ *and approximation* $\tilde{\boldsymbol{K}} = \boldsymbol{K}\tilde{\boldsymbol{S}}(\tilde{\boldsymbol{S}}^\top\boldsymbol{K}\tilde{\boldsymbol{S}})^+\tilde{\boldsymbol{S}}^\top\boldsymbol{K}$ *satisfy*:

$$\tilde{\boldsymbol{K}} \preceq \boldsymbol{K} \preceq \tilde{\boldsymbol{K}} + \lambda\mathbf{I},$$

with $\sum_i \eta_i = O(\sum_i \tilde{l}_i \log(\sum_i \tilde{l}_i/\delta))$ *and the number of sampled data points* $s \le 2\sum_i \eta_i$.

Lemma 6 *Suppose* $\lambda > 0$, $\delta \in (0, 1/8)$, *the sampling matrix* $\boldsymbol{S} \in \mathbb{R}^{n\times s}$ *is obtained by sampling the standard basis vectors independently with probability* η_i *and rescaled with* $1/\sqrt{\eta_i}$, *then with probability* $(1-\delta)$, $1/2 \cdot \sum_i \eta_i \le s \le 2\sum_i \eta_i$ *and*:

$$\frac{1}{2}(\boldsymbol{P}^\top\boldsymbol{P}-\lambda\mathbf{I}) \preceq \boldsymbol{P}^\top\boldsymbol{S}\boldsymbol{S}^\top\boldsymbol{P} \preceq \frac{3}{2}(\boldsymbol{P}^\top\boldsymbol{P}+\lambda\mathbf{I}). \tag{13}$$

Corollary 1 *With probability* $1-\delta$, *Algorithm* 1 *run with ridge parameter* $\epsilon\lambda$ *returns* $\boldsymbol{S}\in\mathbb{R}^{n\times s}$ *such that* $s=O(\frac{\mu}{\epsilon}\log\frac{\mu}{\delta\epsilon})$ *with* $\mu=\operatorname{tr}(\boldsymbol{K}(\boldsymbol{K}+\epsilon\lambda\mathbf{I})^{-1})$, $\boldsymbol{K}=\boldsymbol{P}\boldsymbol{P}^\top$ *and* $\tilde{\boldsymbol{K}}=\boldsymbol{K}\boldsymbol{S}(\boldsymbol{S}^\top\boldsymbol{K}\boldsymbol{S})^{+}\boldsymbol{S}^\top\boldsymbol{K}$ *satisfy* $\tilde{\boldsymbol{K}}\preceq\boldsymbol{K}\preceq\tilde{\boldsymbol{K}}+\epsilon\lambda\mathbf{I}$.

Proof This follows from Lemma 5 by noting $\operatorname{tr}(\boldsymbol{K}(\boldsymbol{K}+\epsilon\lambda\mathbf{I})^{-1})\le\frac{1}{\epsilon}\operatorname{tr}(\boldsymbol{K}(\boldsymbol{K}+\lambda\mathbf{I})^{-1})$ since $(\boldsymbol{K}+\epsilon\lambda\mathbf{I})^{-1}\preceq\frac{1}{\epsilon}(\boldsymbol{K}+\lambda\mathbf{I})^{-1}$. □

Lemma 7 *For any* $\epsilon\in(0,1)$, $\delta\in(0,1/8)$, *Algorithm* 1 *runs with ridge parameter* $\lambda=\frac{\epsilon}{k}\sum_{i=k+1}^{n}\sigma_i(\boldsymbol{K})$ *returns matrix* $\boldsymbol{S}\in\mathbb{R}^{n\times s}$, *where* $\boldsymbol{K}=\boldsymbol{P}\boldsymbol{P}^\top$, $\sigma(\boldsymbol{K})$ *is the singular value of* $\boldsymbol{K}$. *Then with probability* $1-\delta$, $1/2\sum_i\mu_i\le s\le 2\sum_i\mu_i$, $\tilde{\boldsymbol{K}}=\boldsymbol{K}\boldsymbol{S}(\boldsymbol{S}^\top\boldsymbol{K}\boldsymbol{S})^{+}\boldsymbol{S}^\top\boldsymbol{K}$ *satisfies*, *for any rank* k *orthogonal projection* $\boldsymbol{W}$ *and a positive constant* c *independent of* $\boldsymbol{W}$:

$$\operatorname{tr}(\boldsymbol{K}-\boldsymbol{W}\boldsymbol{K}\boldsymbol{W}) \le \operatorname{tr}(\tilde{\boldsymbol{K}}-\boldsymbol{W}\tilde{\boldsymbol{K}}\boldsymbol{W})+c \le (1+\epsilon)\operatorname{tr}(\boldsymbol{K}-\boldsymbol{W}\boldsymbol{K}\boldsymbol{W}). \tag{14}$$

When ridge leverage scores are computed exactly, $\sum_i\mu_i=O(\frac{k}{\epsilon}\log\frac{k}{\delta\epsilon})$.

Proof Set $c=\operatorname{tr}(\boldsymbol{K})-\operatorname{tr}(\tilde{\boldsymbol{K}})$, which is ≥ 0 since $\tilde{\boldsymbol{K}}\preceq\boldsymbol{K}$ by Lemma 5. By linearity of trace:

$$\operatorname{tr}(\tilde{\boldsymbol{K}}-\boldsymbol{W}\tilde{\boldsymbol{K}}\boldsymbol{W})+c=\operatorname{tr}(\boldsymbol{K})-\operatorname{tr}(\boldsymbol{W}\tilde{\boldsymbol{K}}\boldsymbol{W}). \tag{15}$$

So to obtain (14) it suffices to show:

$$\operatorname{tr}(\boldsymbol{W}\boldsymbol{K}\boldsymbol{W})-\epsilon\operatorname{tr}(\boldsymbol{K}-\boldsymbol{W}\boldsymbol{K}\boldsymbol{W}) \le \operatorname{tr}(\boldsymbol{W}\tilde{\boldsymbol{K}}\boldsymbol{W}) \le \operatorname{tr}(\boldsymbol{W}\boldsymbol{K}\boldsymbol{W}). \tag{16}$$

$\boldsymbol{W}$ is a rank k orthogonal projection, we can write $\boldsymbol{W}=\boldsymbol{V}\boldsymbol{V}^\top$ where $\boldsymbol{V}\in\mathbb{R}^{n\times k}$ has orthonormal columns. Applying the cyclic property of the trace, and the spectral bound of Lemma 5:

$$\operatorname{tr}(\boldsymbol{W}\tilde{\boldsymbol{K}}\boldsymbol{W})=\operatorname{tr}(\boldsymbol{V}^\top\tilde{\boldsymbol{K}}\boldsymbol{V})=\sum_{i=1}^{k}v_i^\top\tilde{\boldsymbol{K}}v_i\le\sum_{i=1}^{k}v_i^\top\boldsymbol{K}v_i=\operatorname{tr}(\boldsymbol{V}^\top\boldsymbol{K}\boldsymbol{V})=\operatorname{tr}(\boldsymbol{W}\boldsymbol{K}\boldsymbol{W}). \tag{17}$$

This gives us the upper bound of (16). For the lower bound we apply Corollary 1:

$$\operatorname{tr}(\boldsymbol{W}\tilde{\boldsymbol{K}}\boldsymbol{W})=\sum_{i=1}^{k}v_i^\top\tilde{\boldsymbol{K}}v_i\ge\sum_{i=1}^{k}v_i^\top\boldsymbol{K}v_i-k\epsilon\lambda=\operatorname{tr}(\boldsymbol{W}\boldsymbol{K}\boldsymbol{W})-k\epsilon\lambda. \tag{18}$$

Finally, $\operatorname{tr}(\boldsymbol{K})=\sum_{i=1}^{n}\sigma_i(\boldsymbol{K})$ and $\operatorname{tr}(\boldsymbol{W}\boldsymbol{K}\boldsymbol{W})\le\sum_{i=1}^{k}\sigma_i(\boldsymbol{K})$, by the Eckart-Young theorem, we get $k\epsilon\lambda=\epsilon\sum_{i=k+1}^{n}\sigma_i(\boldsymbol{K})\le\epsilon\operatorname{tr}(\boldsymbol{K}-\boldsymbol{W}\boldsymbol{K}\boldsymbol{W})$. Plugging into (18) gives (16). The proof is complete. □

Lemma 8 *Algorithm* 1 *runs with ridge parameter* $\lambda=\frac{\epsilon}{k}\sum_{i=k+1}^{n}\sigma_i(\boldsymbol{K})$ *returns* $\tilde{\boldsymbol{S}}$, *let* $\boldsymbol{K}=\boldsymbol{P}\boldsymbol{P}^\top$ *and* $\tilde{\boldsymbol{K}}=\boldsymbol{K}\boldsymbol{S}(\boldsymbol{S}^\top\boldsymbol{K}\boldsymbol{S})^{+}\boldsymbol{S}^\top\boldsymbol{K}$. *Suppose* $\boldsymbol{W}_*$ *is the optimal solution for* $\arg\min\operatorname{tr}(\tilde{\boldsymbol{K}}-\boldsymbol{W}\boldsymbol{W}^\top\tilde{\boldsymbol{K}}\boldsymbol{W}\boldsymbol{W}^\top)$, *for* $\tilde{\boldsymbol{K}}$ *and let* $\tilde{\boldsymbol{W}}$ *be an approximately optimal solution satisfying*:

$$\operatorname{tr}(\tilde{\boldsymbol{K}}-\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^\top\tilde{\boldsymbol{K}}\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^\top)\le(1+\gamma)\operatorname{tr}(\tilde{\boldsymbol{K}}-\tilde{\boldsymbol{W}}_*\tilde{\boldsymbol{W}}_*^\top\tilde{\boldsymbol{K}}\tilde{\boldsymbol{W}}_*\tilde{\boldsymbol{W}}_*^\top). \tag{19}$$

Then, *if* $\boldsymbol{W}_*$ *is the optimal cluster indicator matrix for* $\boldsymbol{K}$:

$$\operatorname{tr}(\boldsymbol{K}-\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^{\top}\boldsymbol{K}\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^{\top}) \leq (1+\gamma)(1+\epsilon)\operatorname{tr}(\boldsymbol{K}-\boldsymbol{W}_{*}\boldsymbol{W}_{*}^{\top}\boldsymbol{K}\boldsymbol{W}_{*}\boldsymbol{W}_{*}^{\top}). \tag{20}$$

$$\begin{aligned}
\operatorname{tr}(\boldsymbol{K}-\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^{\top}\boldsymbol{K}\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^{\top}) &\leq tr(\tilde{\boldsymbol{K}}-\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^{\top}\tilde{\boldsymbol{K}}\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^{\top})+c \quad \text{(Lemma 7)}\\
&\leq (1+\gamma)\operatorname{tr}(\tilde{\boldsymbol{K}}-\tilde{\boldsymbol{W}}_{*}\tilde{\boldsymbol{W}}_{*}^{\top}\tilde{\boldsymbol{K}}\tilde{\boldsymbol{W}}_{*}\tilde{\boldsymbol{W}}_{*}^{\top})+(1+\gamma)c \quad \text{(by assumption)}\\
&\leq (1+\gamma)\operatorname{tr}(\tilde{\boldsymbol{K}}-\boldsymbol{W}_{*}\boldsymbol{W}_{*}^{\top}\tilde{\boldsymbol{K}}\boldsymbol{W}_{*}\boldsymbol{W}_{*}^{\top})+(1+\gamma)c \quad \text{(optimality of } \tilde{\boldsymbol{W}}_{*})\\
&\leq (1+\gamma)\operatorname{tr}(\tilde{\boldsymbol{K}}-\boldsymbol{W}_{*}\boldsymbol{W}_{*}^{\top}\tilde{\boldsymbol{K}}\boldsymbol{W}_{*}\boldsymbol{W}_{*}^{\top})+c \quad \text{(since } c\geq 0)\\
&\leq (1+\gamma)(1+\epsilon)\operatorname{tr}(\boldsymbol{K}-\boldsymbol{W}_{*}\boldsymbol{W}_{*}^{\top}\boldsymbol{K}\boldsymbol{W}_{*}\boldsymbol{W}_{*}^{\top}) \quad \text{(Lemma 7)}.
\end{aligned} \tag{21}$$

□

Lemma 9 *If the ridge leverage score in Algorithm* 1 *is computed exactly, the sum of approximated leverage scores is bounded as* $\sum_i l_i \leq \frac{2k}{\epsilon}$.

Proof

$$\begin{aligned}
\sum_i l_i &= \operatorname{tr}(\boldsymbol{K}(\boldsymbol{K}+\lambda\mathbf{I})^{-1})\\
&= \operatorname{tr}\left(\boldsymbol{K}\left(\boldsymbol{K}+\frac{\epsilon}{k}\sum_{i=k+1}^{n}\sigma_i(\boldsymbol{K})\mathbf{I}\right)^{-1}\right) \qquad \left(\lambda=\frac{\epsilon}{k}\sum_{i=k+1}^{n}\sigma_i(K) \text{ in Theorem 5}\right)\\
&\leq \frac{1}{\epsilon}\operatorname{tr}\left(\boldsymbol{K}\left(\boldsymbol{K}+\frac{1}{k}\sum_{i=k+1}^{n}\sigma_i(\boldsymbol{K})\mathbf{I}\right)^{-1}\right)\\
&= \frac{1}{\epsilon}\sum_{i=1}^{n}\frac{\sigma_i(\boldsymbol{K})}{\sigma_i(\boldsymbol{K})+\frac{1}{k}\sum_{i=k+1}^{n}\sigma_i(\boldsymbol{K})}\\
&= \frac{1}{\epsilon}\left(\sum_{i=1}^{k}\frac{\sigma_i(\boldsymbol{K})}{\sigma_i(\boldsymbol{K})+\frac{1}{k}\sum_{i=k+1}^{n}\sigma_i(\boldsymbol{K})}+\sum_{i=k+1}^{n}\frac{\sigma_i(\boldsymbol{K})}{\sigma_i(\boldsymbol{K})+\frac{1}{k}\sum_{i=k+1}^{n}\sigma_i(\boldsymbol{K})}\right)\\
&\leq \frac{1}{\epsilon}\left(k+\sum_{i=k+1}^{n}\frac{\sigma_i(\boldsymbol{K})}{\frac{1}{k}\sum_{i=k+1}^{n}\sigma_i(\boldsymbol{K})}\right)=\frac{2k}{\epsilon}.
\end{aligned} \tag{22}$$

□

Appendix B: Proof for Theorem 1

Proof $\boldsymbol{Z}$ contains orthonormal columns such that the value of the following objective function:

$$\left\|\boldsymbol{P}-\boldsymbol{P}\boldsymbol{Z}\boldsymbol{Z}^{\top}\right\|_{F}^{2} \tag{23}$$

is as small as possible. The above function equals:

$$\operatorname{tr}(\boldsymbol{P}\boldsymbol{P}^\top - (\boldsymbol{P}\boldsymbol{Z}\boldsymbol{Z}^\top)(\boldsymbol{P}\boldsymbol{Z}\boldsymbol{Z}^\top)^\top). \tag{24}$$

Note that $\boldsymbol{Z}$ lies in the column span of $\boldsymbol{P}$, we represent it by introducing a matrix $\boldsymbol{Y}$ such that $\boldsymbol{Z} = \boldsymbol{P}^\top\boldsymbol{Y}$. Let $\boldsymbol{K}$ denote the linear kernel of data set, that is $\boldsymbol{K} = \boldsymbol{P}\boldsymbol{P}^\top$, then minimizing (24) is equivalent to minimizing

$$\operatorname{tr}(\boldsymbol{K} - \boldsymbol{K}\boldsymbol{Y}\boldsymbol{Y}^\top\boldsymbol{K}). \tag{25}$$

It follows from the fact that $\boldsymbol{Z}$ is orthonormal as $(\boldsymbol{P}^\top\boldsymbol{Y})^\top\boldsymbol{P}^\top\boldsymbol{Y} = \boldsymbol{Y}^\top\boldsymbol{K}\boldsymbol{Y} = \mathbf{I}$. Then we design a matrix $\boldsymbol{W} \in \mathbb{R}^{n\times k}$ with orthonormal columns, and introduce

$$\boldsymbol{Y} = \boldsymbol{K}^{-1/2}\boldsymbol{W}, \quad \text{then} \quad \boldsymbol{Z} = \boldsymbol{P}^\top\boldsymbol{K}^{-1/2}\boldsymbol{W} \tag{26}$$

From the cyclic and linearity property of the trace, we can rewrite (25) as:

$$\operatorname{tr}(\boldsymbol{K} - \boldsymbol{K}\boldsymbol{Y}\boldsymbol{Y}^\top\boldsymbol{K}) = \operatorname{tr}(\boldsymbol{K}) - \operatorname{tr}(\boldsymbol{Y}^\top\boldsymbol{K}\boldsymbol{K}\boldsymbol{Y}) = \operatorname{tr}(\boldsymbol{K}) - \operatorname{tr}(\boldsymbol{W}^\top\boldsymbol{K}\boldsymbol{W}) = \operatorname{tr}(\boldsymbol{K}) - \operatorname{tr}(\boldsymbol{W}\boldsymbol{W}^\top\boldsymbol{K}\boldsymbol{W}\boldsymbol{W}^\top). \tag{27}$$

Let $\tilde{\boldsymbol{W}} = \arg\min_{\boldsymbol{W}} \operatorname{tr}(\tilde{\boldsymbol{K}}) - \operatorname{tr}(\boldsymbol{W}\boldsymbol{W}^\top\tilde{\boldsymbol{K}}\boldsymbol{W}\boldsymbol{W}^\top)$, and $\boldsymbol{W}_* = \arg\min_{\boldsymbol{W}} \operatorname{tr}(\boldsymbol{K}) - \operatorname{tr}(\boldsymbol{W}\boldsymbol{W}^\top\boldsymbol{K}\boldsymbol{W}\boldsymbol{W}^\top)$, where $\tilde{\boldsymbol{K}} = \boldsymbol{K}\boldsymbol{S}(\boldsymbol{S}^\top\boldsymbol{K}\boldsymbol{S})^\dagger\boldsymbol{S}^\top\boldsymbol{K}$. Following argument in Lemma 8, we have

$$\begin{aligned}\operatorname{tr}(\boldsymbol{K}) - \operatorname{tr}(\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^\top\boldsymbol{K}\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^\top) &\le (1+\epsilon)(\operatorname{tr}(\boldsymbol{K}) - \operatorname{tr}(\boldsymbol{W}_*\boldsymbol{W}_*^\top\boldsymbol{K}\boldsymbol{W}_*\boldsymbol{W}_*^\top)) \\ &= (1+\epsilon)\sum_{i=k+1}^{n}\sigma_i(\boldsymbol{K}),\end{aligned} \tag{28}$$

as $\boldsymbol{W}$ is an $n \times k$ matrix with orthonormal columns. Our problem is reduced to a low-rank approximation problem that looks like the k-means problem and $\boldsymbol{W}$ is the cluster indicator matrix. Hence $\tilde{\boldsymbol{W}}$ can be taken to equal the top k eigenvectors of $\tilde{\boldsymbol{K}}$ which can be solved by performing singular value decomposition of $\tilde{\boldsymbol{K}}$ with time cost $O(n \cdot s^2)$.

According to (26), we can get $\boldsymbol{Z} = \boldsymbol{P}^\top\boldsymbol{K}^{-1/2}\tilde{\boldsymbol{W}}$. However, $\boldsymbol{Z}$ cannot be represented efficiently and projecting new vectors to $\boldsymbol{Z}$ requires n kernel evaluations to multiply by $\boldsymbol{P}^\top$. Recalling that $\boldsymbol{K} = \boldsymbol{P}\boldsymbol{P}^\top$, $\boldsymbol{S}$ selects s data points $\boldsymbol{S}^\top\boldsymbol{P}$ and we approximate $\boldsymbol{P}$ using its projection onto these points. Informally, let $\Phi \in \mathbb{R}^{d\times d}$ be the orthogonl projection onto the row span of $\boldsymbol{S}^\top\boldsymbol{P}$. We approximate $\boldsymbol{P}$ by $\tilde{\boldsymbol{P}} \stackrel{\text{def}}{=} \boldsymbol{P}\Phi$. We can write $\Phi = \boldsymbol{P}^\top\boldsymbol{S}(\boldsymbol{S}^\top\boldsymbol{P}\boldsymbol{P}^\top\boldsymbol{S})^+\boldsymbol{S}^\top\boldsymbol{P}$. Since it is an orthogonal projection, $\Phi\Phi^\top = \Phi^2 = \Phi$, and we can write:

$$\tilde{\boldsymbol{Z}} = \Phi\boldsymbol{P}^\top\tilde{\boldsymbol{K}}^{-1/2}\tilde{\boldsymbol{W}}$$

$\tilde{\boldsymbol{Z}}$ is orthonormal as $\tilde{\boldsymbol{Z}}^\top\tilde{\boldsymbol{Z}} = \tilde{\boldsymbol{W}}^\top\tilde{\boldsymbol{K}}^{-1/2}\boldsymbol{P}^\top\Phi\boldsymbol{P}\tilde{\boldsymbol{K}}^{-1/2}\tilde{\boldsymbol{W}} = \mathbf{I}$. We argue that the approximated principal components offers a good solution to (24) as $\boldsymbol{Z} = \boldsymbol{P}^\top\boldsymbol{K}^{-1/2}\tilde{\boldsymbol{W}}$. To this end, we substitute $\boldsymbol{Z}$ with $\tilde{\boldsymbol{Z}}$ in (24) and get:

$$\begin{aligned}\operatorname{tr}(\boldsymbol{K} - \boldsymbol{P}\Phi\boldsymbol{P}^\top\tilde{\boldsymbol{K}}^{-1/2}\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^\top\tilde{\boldsymbol{K}}^{-1/2}\boldsymbol{P}\Phi\boldsymbol{P}^\top) &= \operatorname{tr}(\boldsymbol{K}) - \operatorname{tr}(\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^\top\tilde{\boldsymbol{K}}^{-1/2}\boldsymbol{P}\Phi\boldsymbol{P}^\top\boldsymbol{P}\Phi\boldsymbol{P}^\top\tilde{\boldsymbol{K}}^{-1/2}) \\ &= \operatorname{tr}(\boldsymbol{K}) - \operatorname{tr}(\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^\top\tilde{\boldsymbol{K}}^{-1/2}\tilde{\boldsymbol{K}}^2\tilde{\boldsymbol{K}}^{-1/2}) \\ &= \operatorname{tr}(\boldsymbol{K}) - \operatorname{tr}(\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^\top\tilde{\boldsymbol{K}}).\end{aligned} \tag{29}$$

Compare the objective function values of (23) obtained from $\tilde{\boldsymbol{Z}} = \Phi\boldsymbol{P}^\top\tilde{\boldsymbol{K}}^{-1/2}\tilde{\boldsymbol{W}}$ and $\boldsymbol{Z} = \boldsymbol{P}^\top\boldsymbol{K}^{-1/2}\tilde{\boldsymbol{W}}$:

$$\begin{aligned}\operatorname{tr}(\tilde{\boldsymbol{K}}) - \operatorname{tr}(\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^\top\tilde{\boldsymbol{K}}\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^\top) - \left[\operatorname{tr}(\tilde{\boldsymbol{K}}) - \operatorname{tr}(\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^\top\tilde{\boldsymbol{K}})\right] &= \operatorname{tr}(\tilde{\boldsymbol{W}}^\top(\boldsymbol{K}-\tilde{\boldsymbol{K}})\tilde{\boldsymbol{W}}) \\ &= \sum_{i=1}^{k} \boldsymbol{w}_i^\top(\boldsymbol{K}-\tilde{\boldsymbol{K}})\boldsymbol{w}_i \\ &\le \epsilon \sum_{i=1}^{k} \sigma_i(\boldsymbol{K}).\end{aligned} \tag{30}$$

The last step follows from Lemma 2 that $\boldsymbol{K} - \tilde{\boldsymbol{K}} \preceq \epsilon\lambda\mathbf{I}$. We rewrite (30) as:

$$\begin{aligned}\operatorname{tr}(\tilde{\boldsymbol{K}}) - \operatorname{tr}(\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^\top\tilde{\boldsymbol{K}}\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^\top) &\le (\operatorname{tr}(\tilde{\boldsymbol{K}}) - \operatorname{tr}(\tilde{\boldsymbol{W}}\tilde{\boldsymbol{W}}^\top\tilde{\boldsymbol{K}})) + \epsilon\sum_{i=1}^{k}\sigma_i(\boldsymbol{K}) \\ &\le (1+2\epsilon)\sum_{i=k+1}^{n}\sigma_i(\boldsymbol{K}).\end{aligned}$$

This gives the result. Note that $\Phi\boldsymbol{P}^\top\tilde{\boldsymbol{K}}^{-1/2}\tilde{\boldsymbol{W}} = \boldsymbol{P}^\top\boldsymbol{S}(\boldsymbol{S}^\top\boldsymbol{K}^\top\boldsymbol{S})^+\boldsymbol{S}^\top\tilde{\boldsymbol{K}}^{1/2}\tilde{\boldsymbol{W}}$, we set:

$$\begin{aligned}\boldsymbol{X} =& (\boldsymbol{S}^\top\boldsymbol{K}^\top\boldsymbol{S})^\dagger\boldsymbol{S}^\top\tilde{\boldsymbol{K}}^{1/2}\tilde{\boldsymbol{W}}, \\ \tilde{\boldsymbol{K}}^{1/2} =& (\boldsymbol{K}\boldsymbol{S}(\boldsymbol{S}^\top\boldsymbol{K}\boldsymbol{S})^\dagger\boldsymbol{S}^\top\boldsymbol{K})^{1/2} \\ =& (\boldsymbol{P}\boldsymbol{P}^\top\boldsymbol{S}(\boldsymbol{S}^\top\boldsymbol{P}\boldsymbol{P}^\top\boldsymbol{S})^\dagger\boldsymbol{S}^\top\boldsymbol{P}\boldsymbol{P}^\top)^{1/2} \\ =& \boldsymbol{P}\boldsymbol{P}^\top\boldsymbol{S}(\boldsymbol{S}^\top\boldsymbol{P})^\dagger = \boldsymbol{P}.\end{aligned}$$

The solution of (24) can be represented as $\tilde{\boldsymbol{Z}} = \boldsymbol{P}^\top\boldsymbol{S}\boldsymbol{X}$. □

Appendix C: Proof for Lemma 4

Proof Let $\boldsymbol{p}_{\tilde{U}}$, $\boldsymbol{p}^*_{\tilde{U}}$ and $\boldsymbol{q}_{\tilde{U}}$ denote the data points $\boldsymbol{p}$, $\boldsymbol{p}^*$ and $\boldsymbol{q}$ in $\boldsymbol{Z}$ projected k-dimensional subspace. For the nearest neighbor of the query, we have the projected distance bounded as

$$\begin{aligned}\left\|\boldsymbol{q}_{\tilde{U}} - \boldsymbol{p}^*_{\tilde{U}}\right\|_2 &\le \left\|(\boldsymbol{q}-\boldsymbol{p}^*)\boldsymbol{Z}\boldsymbol{Z}^\top\right\|_2 \\ &\le \|\boldsymbol{q}-\boldsymbol{p}^*\|_2\left\|\boldsymbol{Z}\boldsymbol{Z}^\top\right\|_2 \\ &\le \|\boldsymbol{q}-\boldsymbol{p}^*\|_2.\end{aligned}$$

Then, for any other data point in the data set $\boldsymbol{P}$, we get

$$\begin{aligned}\left\|\boldsymbol{q}_{\tilde{U}} - \boldsymbol{p}_{\tilde{U}}\right\|_2 &= \left\|(\boldsymbol{q}-\boldsymbol{p})\boldsymbol{Z}\boldsymbol{Z}^\top\right\|_2 \\ &\ge \left\|\boldsymbol{q}-\boldsymbol{q}\boldsymbol{Z}\boldsymbol{Z}^\top\right\|_2 - \left\|\boldsymbol{q}-\boldsymbol{p}\boldsymbol{Z}\boldsymbol{Z}^\top\right\|_2 \\ &\ge \|\boldsymbol{q}-\boldsymbol{p}\|_2 - \left\|\boldsymbol{q}-\boldsymbol{q}\boldsymbol{Z}\boldsymbol{Z}^\top\right\|_2 - \left\|\boldsymbol{p}-\boldsymbol{p}\boldsymbol{Z}\boldsymbol{Z}^\top\right\|_2.\end{aligned}$$

According to Theorem 2 that the distance between data and the projected space is bounded, we get

$$\left\|\boldsymbol{q}_{\tilde{U}} - \boldsymbol{p}_{\tilde{U}}\right\|_2 \geq \|\boldsymbol{q} - \boldsymbol{p}\|_2 - 2 \cdot (2 + \epsilon) \sum_{i=k+1}^{n} \sigma_i.$$

For the case that k is close to the rank of $\boldsymbol{P}$, $\sum_{k+1}^{n} \sigma_i$ can be as small as possible. Hence, in the low-dimensional subspace, the nearest neighbor of $\boldsymbol{q}$ is $\boldsymbol{p}^*$. □

Declarations

Funding Jie Shen is supported by NSF-IIS-1948133 and the startup funding of Stevens Institute of Technology.

Conflicts of interest/competing interests The authors have no relevant financial or non-financial interests to disclose.

Availability of data and material All the data sets used in the paper are publicly available, and have been cited in the main text.

Code availability All the codes will be release on the authors' personal websites after acceptance.

Authors' contributions All authors contributed to the work. The paper was written together by Jie Shen and Jing Wang. Jing Wang designed and ran the experiments and Jie Shen commented on ways for improvements.

Ethics approval Not applicable.

Consent to participate Not applicable.

Consent for publication. Not applicable.

References

Abdullah, A., Andoni, A., Kannan, R. & Krauthgamer, R. (2014). Spectral approaches to nearest neighbor search. In *Annual symposium on foundations of computer science* (pp. 581–590).

Alaoui, A., & Mahoney, M. W. (2015). Fast randomized kernel ridge regression with statistical guarantees. In *Advances in neural information processing systems* (pp. 775–783).

Andoni, A., & Indyk, P. (2008). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM, 51*(1), 117–122.

Andoni, A., Indyk, P., Nguyen, H.L. & Razenshteyn, I. (2014). Beyond locality-sensitive hashing. In *Proceedings of the 2014 ACM-SIAM symposium on discrete algorithms*, SIAM (pp. 1018–1028).

Andoni, A., Naor, A., Nikolov, A., Razenshteyn, I., & Waingarten, E. (2018). Data-dependent hashing via nonlinear spectral gaps. In *Annual ACM SIGACT symposium on theory of computing*. ACM (pp. 787–800).

Andoni, A,. Nikolov, A., Razenshteyn, I., & Waingarten, E. (2021). Approximate nearest neighbors beyond space partitions. In *Proceedings of the 2021 ACM-SIAM symposium on discrete algorithms* (pp. 1171–1190) SIAM.

Arya, S., & Mount, D.M. (1993). Approximate nearest neighbor queries in fixed dimensions. In *Proceedings of the fourth annual ACM/SIGACT-SIAM symposium on discrete algorithms* (pp. 271–280).

Arya, S., Mount, D. M., & Narayan, O. (1995). Accounting for boundary effects in nearest neighbor searching. In *Proceedings of the eleventh annual symposium on computational geometry*, Vancouver (pp. 336–344).

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

Çakir, F., He, K., Sclaroff, S. (2018). Hashing with binary matrix pursuit. In *European conference on computer vision* (pp. 344–361).

Cohen, M.B., Musco, C., & Pachocki, J.W. (2016). Online row sampling. In *Approximation, randomization, and combinatorial optimization. algorithms and techniques, APPROX/RANDOM* (pp 7:1–7:18).

Dobkin, D. P., & Lipton, R. J. (1976). Multidimensional searching problems. *SIAM Journal on Computing 5*(2), 181–186.

Dolan, W.B., & Brockett, C. (2005). Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the third international workshop on paraphrasing*.

Erin Liong, V., Lu, J., Wang, G., Moulin, P., Zhou, J. (2015). Deep hashing for compact binary codes learning. In *IEEE conference on computer vision and pattern recognition* (pp. 2475–2483).

Gersho, A., & Gray, R. M. (2012). Vector quantization and signal compression, vol 159. Springer.

Gionis, A., Indyk, P., & Motwani, R. (1999). Similarity search in high dimensions via hashing. *International Conference on Very Large Data Bases, 99,* 518–529.

Gittens, A. (2011). The spectral norm error of the naive nystrom extension. arXiv preprint arXiv:11105305

Gittens, A., & Mahoney, M. W. (2016). Revisiting the nyström method for improved large-scale machine learning. *The Journal of Machine Learning Research, 17*(1), 3977–4041.

Gong, Y., & Lazebnik, S. (2011). Iterative quantization: A procrustean approach to learning binary codes. In *IEEE conference on computer vision and pattern recognition* (pp. 817–824).

Gong, Y., Kumar, S., Rowley, H. A., & Lazebnik, S. (2013). Learning binary codes for high-dimensional data using bilinear projections. In *IEEE conference on computer vision and pattern recognition* (pp. 484–491).

Han, X., Leung, T., Jia, Y., Sukthankar, R., & Berg, A. C. (2015). Matchnet: Unifying feature and metric learning for patch-based matching. In *The 28th IEEE conference on computer vision and pattern recognition* (pp. 3279–3286).

Indyk, P., & Motwani, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Annual ACM symposium on theory of computing*, ACM (pp. 604–613).

Jegou, H., Douze, M., & Schmid, C. (2011). Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 33*(1), 117–128.

Jiang, Q., &Li, W. (2015). Scalable graph hashing with feature transformation. In *International joint conference on artificial intelligence* (pp. 2248–2254).

Knuth, D. E. (1973). Sorting and searching. The art of computer programming 3:Ch–6.

Kong, W., & Li, W. J. (2012). Isotropic hashing. In *Annual conference on neural information processing systems* (pp. 1646–1654).

Kulis, B., & Darrell, T. (2009). Learning to hash with binary reconstructive embeddings. In *Annual conference on neural information processing systems* (pp. 1042–1050).

Liu, H., Wang, R., Shan, S., & Chen, X. (2016). Deep supervised hashing for fast image retrieval. In *IEEE conference on computer vision and pattern recognition* (pp. 2064–2072).

Liu, W., Wang, J., Kumar, S., & Chang, S. F. (2011). Hashing with graphs. In *International conference on machine learning* (pp. 1–8).

Liu, W., Mu, C., Kumar, S., & Chang, S. (2014). Discrete graph hashing. In *Annual conference on neural information processing systems* (pp. 3419–3427).

Makhoul, J., Roucos, S., & Gish, H. (1985). Vector quantization in speech coding. *Proceedings of the IEEE, 73*(11), 1551–1588.

Mitzenmacher, M., & Upfal, E. (2017). *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge University Press

Musco, C., & Musco, C. (2015). Randomized block krylov methods for stronger and faster approximate singular value decomposition. In *Annual conference on neural information processing systems* (pp. 1396–1404).

Musco, C., & Musco, C. (2017). Recursive sampling for the nystrom method. In *Advances in neural information processing systems* (pp. 3836–3848).

Pennington J., Socher R., & Manning C. D. (2014). GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014Conference on Empirical Methods in Natural Language Processing* (pp. 1532–1543).

Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). Squad: 100, 000+ questions for machine comprehension of text. In J. Su, X. Carreras, K. Duh (Eds.) *Proceedings of the 2016 conference on empirical methods in natural language processing* (pp. 2383–2392).

Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In: K. Inui, J. Jiang, V. Ng, X. Wan (Eds.) *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing*, (pp. 3980–3990).

Rudi, A., Camoriano, R., & Rosasco, L. (2015). Less is more: Nyström computational regularization. In *Annual conference on neural information processing systems* (pp. 1648–1656).

Samet, H. (1990). *The design and analysis of spatial data structures* (Vol. 85). Addison-Wesley.

Samet, H. (2006). Foundations of multidimensional and metric data structures

Sellis, T. K., Roussopoulos, N., & Faloutsos, C. (1997). Multidimensional access methods: Trees have grown everywhere. In *Proceedings of 23rd international conference on very large data bases* (pp. 13–14).

Shen, F., Shen, C., Liu, W., & Shen, H. T. (2015). Supervised discrete hashing. In *IEEE conference on computer vision and pattern recognition* (pp. 37–45).

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C.D., Ng, A.Y., & Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing* (pp. 1631–1642).

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2019). GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 7th international conference on learning representations*.

Warstadt, A., Singh, A., & Bowman, S. R. (2019). Neural network acceptability judgments. *Trans Assoc Comput Linguistics, 7,* 625–641.

Weiss, Y., Torralba, A., & Fergus, R. (2009). Spectral hashing. In *Annual conference on neural information processing systems* (pp. 1753–1760).

Weiss, Y., Fergus, R., & Torralba, A. (2012). Multidimensional spectral hashing. In *European conference on computer vision* (pp. 340–353).

Woodruff, D. P., et al. (2014). Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science, 10*(1–2), 1–157.

Xia, Y., He, K., Kohli, P., & Sun, J. (2015). Sparse projections for high-dimensional binary codes. In *IEEE conference on computer vision and pattern recognition* (pp. 3332–3339).

Yu, F. X., Kumar, S., Gong, Y., & Chang, S. (2014). Circulant binary embedding. In *Proceedings of the 31th international conference on machine learning* (pp. 946–954).