

A Kernel Vector Approximation File for Nearest Neighbor Search using Kernel Methods

Douglas R. Heisterkamp
Computer Science Department
Oklahoma State University
Stillwater, OK 74078
drh@ieee.org

Jing Peng
Electrical Engr. and Computer Science
Tulane University
New Orleans, LA 70118
jp@eecs.tulane.edu

1 Introduction

Traditional data partitioning approaches to indexing become inefficient as the number of dimensions increase, eventually taking longer time than a sequential scan of the data. A vector approximation file (VA-File) takes a signature or filter approach to indexing data [21, 7]. By sequentially processing a compressed approximation of the data, VA-File is able to filter most data vectors and need only retrieve a small fraction of the actual data. To be able to conduct the filtering, upper and lower bounds on the distance from the query to the data point needs to be calculated. This calculation, however, is invalid for kernel-based approaches. This paper introduces a novel KVA-File (kernel VA-File) that extends VA-File to kernel-based retrieval methods. An efficient approach to approximating vectors in an induced feature space is presented with the corresponding upper and lower distance bounds. Thus an effective indexing method is provided for kernel-based image retrieval methods.

The layout of the paper is as follows. A vector approximation file is presented in Section 2. Methods using kernel distance is reviewed in 3. The feature space distance bounds and the KVAFile is developed in Section 4. Creating the orthogonal basis and approximating data is presented in Section 5. Initial results on real world data is presented in Section 6.

2 Vector Approximation File

The vector-approximation file (VA-File) uses a signature file as a filter [21, 22]. The signature file is a compressed approximation of the original data file. Each data vector in the original data is stored in the approximation file as the bit encoding of the hypercube in which it lies. For example, in Figure 1, two bits per dimension were used to approximate the data vectors. Vector 3, $[0.25, 0.2]$, is represented by the bit pattern 0100, meaning bin 01 of first dimension and bin 00 of second dimension. Typically, the compressed file is 10% to 15% of the size of the original data file.

The distance of a point to an approximated vector is bounded by the maximum and minimum distances to the hypercube that the bit encoding of the approximation represents. The maximum and minimum distances of a point to the hypercube provides an upper and lower bounds on the distance between the query location and the original data point. For example, in Figure 1, the distance from the query to vector 3 is within the lower bound of l_3 and the upper bound of u_3 . Similarly, l_4 and u_4 are the lower and upper bounds of the distance of vector 4 from the query.

When using a VA-File, a K nearest neighbor search is a two phase processes. In phase one, a filtering of the possible K-NN is done by a sequential scan of the approximation file. The filtering process creates

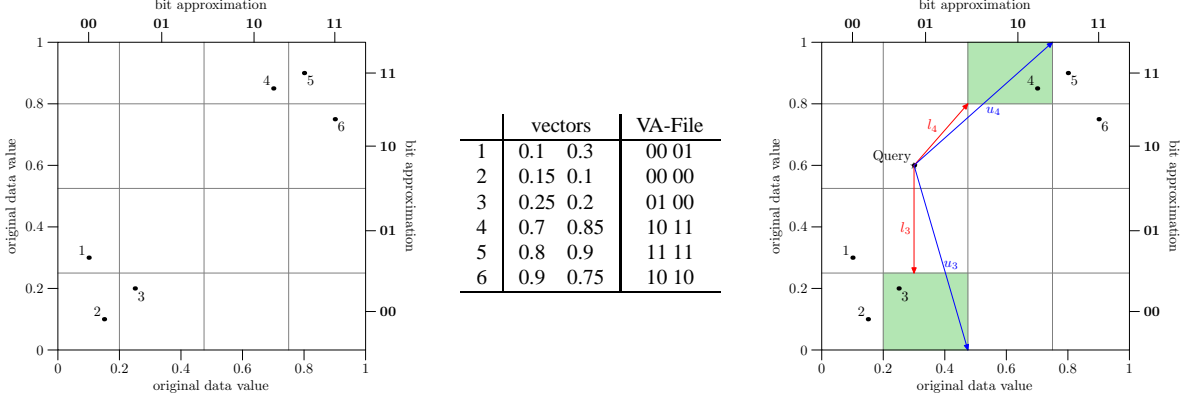


Figure 1: VA-File approximations

a candidate list. As each approximated vector is processed, if its lower bound is less than the current K th closest upper bound then it is added as a candidate for phase two. In phase two, the candidates are visited in ascending order of lower bound until the lower bound of the next candidate is greater than the actual distance to the current K th nearest neighbor.

3 Kernel Distance for Relevance Feedback Retrieval

The kernel trick has been applied to numerous problems [12, 17, 5, 13, 10]. The kernel allows an algorithm to work in a feature space. If $\phi(\mathbf{x})$ is a mapping of a point \mathbf{x} in the input space to the feature space

$$\mathbf{x} = (x_1, \dots, x_q)^t \rightarrow \phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \dots, \phi_n(\mathbf{x}))^t$$

then the kernel calculates the dot product in the feature space of the images of two points from input space, $k(\mathbf{a}, \mathbf{b}) = \langle \phi(\mathbf{a}), \phi(\mathbf{b}) \rangle$. Here q represents the dimensionality of the input space, and n the dimensionality of the feature space. Common kernels are Gaussian, $k(\mathbf{a}, \mathbf{b}) = e^{-\frac{\|\mathbf{a}-\mathbf{b}\|^2}{2\sigma^2}}$, and polynomial, $k(\mathbf{a}, \mathbf{b}) = (1 + \langle \mathbf{a}, \mathbf{b} \rangle)^d$. Distance in the feature space may be calculated by means of the kernel [20, 5]. With \mathbf{a} and \mathbf{b} in the input space, the squared feature space distance is

$$\text{dist}(\mathbf{a}, \mathbf{b})^2 = \|\phi(\mathbf{a}) - \phi(\mathbf{b})\|^2 = k(\mathbf{a}, \mathbf{a}) - 2k(\mathbf{a}, \mathbf{b}) + k(\mathbf{b}, \mathbf{b}). \quad (1)$$

Two known kernel distances in the relevance feedback literature are Adaptive Quasiconformal Kernel (AQK) [9] and One-Class SVM [4, 18]. We briefly present each approach.

Adaptive Quasiconformal Kernel (AQK) distance [9] combines the kernel distance (1) with a quasiconformal mapping [1, 2]

$$\tilde{k}(\mathbf{a}, \mathbf{b}) = c(\mathbf{a})c(\mathbf{b})k(\mathbf{a}, \mathbf{b}). \quad (2)$$

to create a new kernel distance:

$$\text{dist}(\mathbf{a}, \mathbf{b})^2 = \tilde{k}(\mathbf{a}, \mathbf{a}) - 2\tilde{k}(\mathbf{a}, \mathbf{b}) + \tilde{k}(\mathbf{b}, \mathbf{b}) = c(\mathbf{a})^2k(\mathbf{a}, \mathbf{a}) - 2c(\mathbf{a})c(\mathbf{b})k(\mathbf{a}, \mathbf{b}) + c(\mathbf{b})^2k(\mathbf{b}, \mathbf{b}) \quad (3)$$

where $c(\mathbf{a})$ is a positive real valued function of \mathbf{a} in the input space. One can select $c(\mathbf{a})$ from relevance feedback to expand the spatial resolution around irrelevant samples and contract the spatial resolution around relevant samples [9]. That is, distance to irrelevant samples from the query is increased and distance to relevant samples are decreased.

One-Class SVM kernel distance is the distance from a sample to the center of the smallest hypersphere that includes most of the relevant retrievals from the previous iterations [4, 18]. After finding the center, $\mathbf{c} = \sum_i \gamma_i \phi(\mathbf{x}_i)$, the one-class SVM kernel distance of vector \mathbf{z} to \mathbf{c} is

$$\text{dist}(\mathbf{z}, \mathbf{c})^2 = k(\mathbf{z}, \mathbf{z}) - 2 \sum_i \gamma_i k(\mathbf{x}_i, \mathbf{z}) + \sum_{i,j} \gamma_i \gamma_j k(\mathbf{x}_i, \mathbf{x}_j). \quad (4)$$

The γ_i 's are the Lagrangian multipliers from the solution of the quadratic programming problem

$$\min_{R, \Xi, c} R^2 + \frac{1}{vl} \sum_i \xi_i \quad \text{subject to} \quad \|\phi(\mathbf{x}_i) - c\|^2 \leq R^2 + \xi_i, \quad \xi_i \geq 0 \quad \text{for all } i.$$

4 Kernel Vector Approximation File

The approximation file is a reduced representation of the data that allows an efficient calculation of upper and lower distance bounds. To create an approximation, we use a reduced set of orthonormal basis vectors in the feature space. The feature space location of a data point is projected into the reduced space and used as the approximation. In addition, the magnitude of the error (the remaining component orthogonal to basis) is also used as part of the approximation.

Both Kernel-PCA [14, 15] and a Gram-Schmidt approach [6, 3] have been used to find an orthonormal basis of the feature space. The eigenvectors from Kernel-PCA that correspond to the largest eigenvalues would yield the best approximation of the locations in feature space. The problem with using Kernel-PCA is that the representation of the eigenvectors is not compact (it can be on the order of the number of vectors in the data set). For the work in this paper, we used a Gram-Schmidt approach similar to the efficient algorithm in [3] to find an orthonormal basis in the feature space. One of the sparse approaches for Kernel-PCA [19, 16] may also be useful.

A method to find an orthogonal basis and create data approximations is presented in Section 5. For now, we assume a reduced set of basis vectors, \mathbf{v}_t , of the feature space is available and that a point on feature space can be decomposed into a linear combination of basis vectors and a component, $\phi^\perp(\mathbf{x})$ that is orthogonal to the basis. With this decomposition, we can represent points \mathcal{P} and \mathcal{Q} as

$$\mathbf{P} = \phi(\mathbf{x}_p) = \phi^\perp(\mathbf{x}_p) + \sum_{t=0}^{d-1} \alpha_t \mathbf{v}_t \quad \text{and} \quad \mathbf{Q} = \phi(\mathbf{x}_q) = \phi^\perp(\mathbf{x}_q) + \sum_{t=0}^{d-1} \beta_t \mathbf{v}_t$$

where k is the number of basis vectors. Distance between \mathcal{P} and \mathcal{Q} can be expressed as

$$\begin{aligned} \text{dist}(\mathcal{Q}, \mathcal{P})^2 &= \|\phi(\mathbf{x}_q) - \phi(\mathbf{x}_p)\|^2 \\ &= \|\phi^\perp(\mathbf{x}_q) + \sum_{t=0}^{d-1} \beta_t \mathbf{v}_t - (\phi^\perp(\mathbf{x}_p) + \sum_{t=0}^{d-1} \alpha_t \mathbf{v}_t)\|^2 \\ &= \underbrace{\phi^\perp(\mathbf{x}_q)^T \phi^\perp(\mathbf{x}_q)}_{k(\mathbf{x}_q, \mathbf{x}_q)} + \sum_{t=0}^{d-1} \beta_t^2 + \underbrace{\phi^\perp(\mathbf{x}_p)^T \phi^\perp(\mathbf{x}_p)}_{k(\mathbf{x}_p, \mathbf{x}_p)} + \sum_{t=0}^{d-1} \alpha_t^2 - 2 \underbrace{\left(\phi^\perp(\mathbf{x}_q)^T \phi^\perp(\mathbf{x}_p) + \sum_{t=0}^{d-1} \alpha_t \beta_t \right)}_{k(\mathbf{x}_q, \mathbf{x}_p)} \\ &= \phi^\perp(\mathbf{x}_q)^T \phi^\perp(\mathbf{x}_q) + \phi^\perp(\mathbf{x}_p)^T \phi^\perp(\mathbf{x}_p) - 2 \phi^\perp(\mathbf{x}_q)^T \phi^\perp(\mathbf{x}_p) + \sum_{t=0}^{d-1} (\alpha_t - \beta_t)^2. \end{aligned} \quad (5)$$

We approximate a point in feature space by the basis coefficients, α , and the magnitude of the error,

$\sqrt{\boldsymbol{\phi}^\perp(\mathbf{x})^T \boldsymbol{\phi}^\perp(\mathbf{x})}$. With the approximation of points \mathcal{P} and \mathcal{Q} , the unknown term in (5) is $\boldsymbol{\phi}^\perp(\mathbf{x}_q)^T \boldsymbol{\phi}^\perp(\mathbf{x}_p)$. But this is also equal to

$$\boldsymbol{\phi}^\perp(\mathbf{x}_q)^T \boldsymbol{\phi}^\perp(\mathbf{x}_p) = \sqrt{\boldsymbol{\phi}^\perp(\mathbf{x}_q)^T \boldsymbol{\phi}^\perp(\mathbf{x}_q)} \sqrt{\boldsymbol{\phi}^\perp(\mathbf{x}_p)^T \boldsymbol{\phi}^\perp(\mathbf{x}_p)} \cos \theta$$

where θ is the angle between the two vectors. The angle is not represented in the approximation but the extremes can be used to generate bounds on the distance. Using the notation from Section 5 of $\hat{\mathbf{G}}_d(p, p) = \boldsymbol{\phi}^\perp(\mathbf{x}_p)^T \boldsymbol{\phi}^\perp(\mathbf{x}_p)$ and $\hat{\mathbf{G}}_d(q, q) = \boldsymbol{\phi}^\perp(\mathbf{x}_q)^T \boldsymbol{\phi}^\perp(\mathbf{x}_q)$, then the upper distance bounds, $\mathcal{U}(Q, P)$, and the lower distance bounds, $\mathcal{L}(Q, P)$, is

$$\mathcal{U}(Q, P) = \sqrt{\hat{\mathbf{G}}_d(q, q) + \hat{\mathbf{G}}_d(p, p) + 2\sqrt{\hat{\mathbf{G}}_d(q, q)}\sqrt{\hat{\mathbf{G}}_d(p, p)} + \sum_{t=0}^{d-1} (\alpha_t - \beta_t)^2} \quad (6)$$

$$\mathcal{L}(Q, P) = \sqrt{\hat{\mathbf{G}}_d(q, q) + \hat{\mathbf{G}}_d(p, p) - 2\sqrt{\hat{\mathbf{G}}_d(q, q)}\sqrt{\hat{\mathbf{G}}_d(p, p)} + \sum_{t=0}^{d-1} (\alpha_t - \beta_t)^2} \quad (7)$$

The ranges for each α_i and for $\hat{\mathbf{G}}_d(p, p)$ can be partitioned into bins and the bin locations represented by a bit encoding. Thus allowing two levels of compression: the number of basis dimensions and the number of bits per dimension.

4.1 Query as weighted combination of points

In One-Class SVM, [4], a query location is presented as a linear combination of data points:

$$\mathbf{Q} = \sum_i \gamma_i \boldsymbol{\phi}(\mathbf{x}_i) = \sum_i \gamma_i \left(\boldsymbol{\phi}^\perp(\mathbf{x}_i) + \sum_t^{d-1} \beta_{i,t} \mathbf{v}_t \right)$$

The same process as the previous section can be used to generate the distance equation

$$\begin{aligned} \text{dist}(\mathbf{Q}, \mathbf{P})^2 &= \left\| \left(\sum_i \gamma_i \left(\boldsymbol{\phi}^\perp(\mathbf{x}_i) + \sum_t^{d-1} \beta_{i,t} \mathbf{v}_t \right) \right) - \left(\boldsymbol{\phi}^\perp(\mathbf{x}_p) + \sum_{t=0}^{d-1} \alpha_t \mathbf{v}_t \right) \right\|^2 \\ &= \boldsymbol{\phi}^\perp(\mathbf{x}_p)^T \boldsymbol{\phi}^\perp(\mathbf{x}_p) + \sum_i \sum_j \gamma_i \gamma_j \boldsymbol{\phi}^\perp(\mathbf{x}_i)^T \boldsymbol{\phi}^\perp(\mathbf{x}_j) - 2\boldsymbol{\phi}^\perp(\mathbf{x}_p)^T \sum_i \gamma_i \boldsymbol{\phi}^\perp(\mathbf{x}_i) + \sum_{t=0}^{d-1} (\alpha_t - \sum_i \gamma_i \beta_{i,t})^2. \end{aligned}$$

The upper and lower bounds then follow:

$$\begin{aligned} \mathcal{U}(Q, P) &= \sqrt{\hat{\mathbf{G}}_d(p, p) + \sum_i \sum_j \gamma_i \gamma_j \hat{\mathbf{G}}_d(i, j) + 2\sqrt{\hat{\mathbf{G}}_d(p, p)} \sum_i \gamma_i \sqrt{\hat{\mathbf{G}}_d(i, i)} + \sum_{t=0}^{d-1} (\alpha_t - \sum_i \gamma_i \beta_{i,t})^2} \\ \mathcal{L}(Q, P) &= \sqrt{\hat{\mathbf{G}}_d(p, p) + \sum_i \sum_j \gamma_i \gamma_j \hat{\mathbf{G}}_d(i, j) - 2\sqrt{\hat{\mathbf{G}}_d(p, p)} \sum_i \gamma_i \sqrt{\hat{\mathbf{G}}_d(i, i)} + \sum_{t=0}^{d-1} (\alpha_t - \sum_i \gamma_i \beta_{i,t})^2} \end{aligned}$$

4.2 Adaptive quasi-conformal kernel distance

The Adaptive quasi-conformal kernel, [9], modifies the distance calculation by creating a new kernel, \tilde{k}

$$\tilde{k}(\mathbf{x}_a, \mathbf{x}_b) = c(\mathbf{x}_a)c(\mathbf{x}_b)k(\mathbf{x}_a, \mathbf{x}_b).$$

where $c(\mathbf{x})$ is a positive real value function of \mathbf{x} . The distance between points \mathbf{P} and \mathbf{Q} using the new kernel, is

$$\begin{aligned} \text{dist}(Q, P)^2 &= \left\| c(\mathbf{x}_q) \left(\boldsymbol{\phi}^\perp(\mathbf{x}_q) + \sum_{t=0}^{d-1} \beta_t \mathbf{v}_t \right) - c(\mathbf{x}_p) \left(\boldsymbol{\phi}^\perp(\mathbf{x}_p) + \sum_{t=0}^{d-1} \alpha_t \mathbf{v}_t \right) \right\|^2 \\ &= c(\mathbf{x}_q)^2 \boldsymbol{\phi}^\perp(\mathbf{x}_q)^T \boldsymbol{\phi}^\perp(\mathbf{x}_q) + c(\mathbf{x}_p)^2 \boldsymbol{\phi}^\perp(\mathbf{x}_p)^T \boldsymbol{\phi}^\perp(\mathbf{x}_p) - 2c(\mathbf{x}_q)c(\mathbf{x}_p) \boldsymbol{\phi}^\perp(\mathbf{x}_q)^T \boldsymbol{\phi}^\perp(\mathbf{x}_p) \\ &\quad + \sum_{t=0}^{d-1} (c(\mathbf{x}_p)\alpha_t - c(\mathbf{x}_q)\beta_t)^2. \end{aligned}$$

and the upper and lower bounds on the distance is

$$\begin{aligned} \mathcal{U}(Q, P) &= \left(c(\mathbf{x}_q)^2 \mathbf{K}(\mathbf{x}_q, \mathbf{x}_q) + 2c(\mathbf{x}_q)c(\mathbf{x}_p) \sqrt{\hat{\mathbf{G}}_d(q, q)} \sqrt{\hat{\mathbf{G}}_d(p, p)} \right. \\ &\quad \left. - 2c(\mathbf{x}_q)c(\mathbf{x}_p) \sum_{t=0}^{d-1} \alpha_t \beta_t + c(\mathbf{x}_p)^2 \hat{\mathbf{G}}_d(p, p) + c(\mathbf{x}_p)^2 \sum_{t=0}^{d-1} \alpha_t^2 \right)^{\frac{1}{2}} \end{aligned}$$

$$\begin{aligned} \mathcal{L}(Q, P) &= \left(c(\mathbf{x}_q)^2 \mathbf{K}(\mathbf{x}_q, \mathbf{x}_q) - 2c(\mathbf{x}_q)c(\mathbf{x}_p) \sqrt{\hat{\mathbf{G}}_d(q, q)} \sqrt{\hat{\mathbf{G}}_d(p, p)} \right. \\ &\quad \left. - 2c(\mathbf{x}_q)c(\mathbf{x}_p) \sum_{t=0}^{d-1} \alpha_t \beta_t + c(\mathbf{x}_p)^2 \hat{\mathbf{G}}_d(p, p) + c(\mathbf{x}_p)^2 \sum_{t=0}^{d-1} \alpha_t^2 \right)^{\frac{1}{2}}. \end{aligned}$$

$$\mathcal{U}(Q, P) = \sqrt{c(\mathbf{x}_q)^2 \hat{\mathbf{G}}_d(q, q) + c(\mathbf{x}_p)^2 \hat{\mathbf{G}}_d(p, p) + 2c(\mathbf{x}_q)c(\mathbf{x}_p) \sqrt{\hat{\mathbf{G}}_d(q, q)} \sqrt{\hat{\mathbf{G}}_d(p, p)} + \sum_{t=0}^{d-1} (c(\mathbf{x}_p)\alpha_t - c(\mathbf{x}_q)\beta_t)^2}$$

$$\mathcal{L}(Q, P) = \sqrt{c(\mathbf{x}_q)^2 \hat{\mathbf{G}}_d(q, q) + c(\mathbf{x}_p)^2 \hat{\mathbf{G}}_d(p, p) - 2c(\mathbf{x}_q)c(\mathbf{x}_p) \sqrt{\hat{\mathbf{G}}_d(q, q)} \sqrt{\hat{\mathbf{G}}_d(p, p)} + \sum_{t=0}^{d-1} (c(\mathbf{x}_p)\alpha_t - c(\mathbf{x}_q)\beta_t)^2}$$

For a query \mathbf{x}_q , we will have the actual data so we can calculate the exact $c(\mathbf{x}_q)$. But when processing the approximation file, we only have the approximations of the feature space point \mathcal{P} and thus we need to create bounds on the value of $c(\mathbf{x}_p)$. This depends on the form of the function c .

4.3 K Nearest Neighbor Search using KVAFile

The algorithm for K-NN search of the KVAFile is presented in Figure 2. Note that the phase one filtering requires no kernel evaluations beyond the approximation for the query vector in step 1.

Figure 2: K-NN Search of KVAFile

Input: Query \mathbf{x}_q , Data file \mathbf{D} , Approximation file \mathbf{A} where $a_i \in \mathbf{A}$ corresponding to $x_i \in \mathbf{D}$ is the result of the approximation algorithm (see Figure 5).

1. Approximate $\mathbf{x}_q \rightarrow \mathbf{b} = \beta, \hat{\mathbf{G}}_d(q, q)$
2. initialize empty Priority Queue and the K th smallest upper bound, KUB , to infinity.
3. *// Phase One Filtering*
for each $\mathbf{a}_i \in \mathbf{A}$ **do**
 if $\mathcal{L}(\mathbf{b}, \mathbf{a}_i) \leq KUB$ **then**
 push $\mathcal{L}(\mathbf{b}, \mathbf{a}_i), i$ onto Priority Queue
 update K th smallest upper bound, KUB
 end-if
end-for
 // Note: candidates are on the Priority Queue
4. initialize K-NN and K th smallest actual distance, KAD , to infinite distance.
5. *// Phase Two: Searching Data File*
while Queue not empty **and** Queue's front lower $\leq KAD$ **do**
 lower, i = pop front of Priority Queue
 read \mathbf{x}_i from \mathbf{D}^\dagger
 calculate $\text{dist}(\mathbf{x}_q, \mathbf{x}_i)$
 update K-NN and K th smallest actual distance, KAD
end-while
6. **return** K-NN

[†]In actual implementation, read block containing \mathbf{x}_i , measure distance to all records in block, mark block as visited so will not reread block if other records are candidates in the Priority Queue

5 Creating Orthogonal Basis and Approximating Data

5.1 Gram-Schmidt Orthogonalization

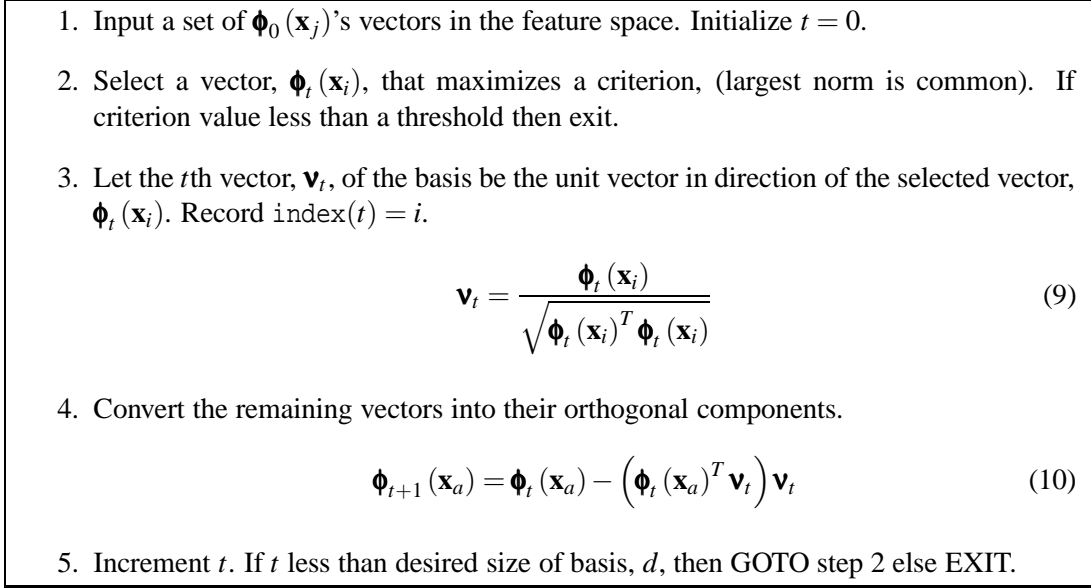
A Gram-Schmidt orthogonalization algorithm is presented in Figure 3. A basic summary of the approach is select a vector; convert all other vectors into their component that is orthogonal to the selected vector; repeat by selecting a new vector for the remaining set.

To implement the Gram-Schmidt algorithm in features space, we can only use the dot products. Given input vectors \mathbf{x}_i , with $0 \leq i \leq m$, the Gram matrix is

$$\mathbf{G}(a, b) = \phi(\mathbf{x}_a)^T \phi(\mathbf{x}_b) = k(\mathbf{x}_a, \mathbf{x}_b). \quad (8)$$

The changes in the Gram matrix that results from applying the algorithm in Figure 3 can be calculated from the entries of the Gram matrix. The orthogonalization algorithm using just the Gram matrix is presented in Figure 4.

Figure 3: Gram-Schmidt Orthogonalization Algorithm



At any iteration t , the mapping $\phi_t(\mathbf{x}_j)$ takes \mathbf{x}_j to the component of $\phi_0(\mathbf{x}_j)$ that is orthogonal to the set of basis vectors, \mathbf{v}_i , $0 \leq i \leq t-1$. The matrix, \mathbf{G}_t , is the Gram matrix of the set of $\phi_t(\mathbf{x}_j)$. The original feature space vector, $\phi_0(\mathbf{x}_j)$, is equal to its representation in the new basis plus the remaining component orthogonal to the new basis.

$$\phi_0(\mathbf{x}_j) = \phi^\perp(\mathbf{x}_j) + \sum_{t=0}^{d-1} \alpha_t \mathbf{v}_t \quad (11)$$

where

$$\alpha_t = \phi_t(\mathbf{x}_j)^T \mathbf{v}_t = \frac{\phi_t(\mathbf{x}_j)^T \phi_t(\mathbf{x}_{\text{index}(t)})}{\sqrt{\phi_t(\mathbf{x}_{\text{index}(t)})^T \phi_t(\mathbf{x}_{\text{index}(t)})}} = \frac{\mathbf{G}_t(j, \text{index}(t))}{\sqrt{\mathbf{G}_t(\text{index}(t), \text{index}(t))}} \quad (12)$$

and $\phi^\perp(\mathbf{x}_j)$ is the component orthogonal to the basis. Again, we don't need the explicit representation of this vector. All that we will need is the magnitude, which we can get from the final Gram matrix:

$$\phi^\perp(\mathbf{x}_j)^T \phi^\perp(\mathbf{x}_j) = \mathbf{G}_d(j, j) \quad (13)$$

5.2 Approximation of New Vectors

To approximate new data, the orthogonalization steps for that data point must be calculated. If \mathbf{x}_p was in the original data set, then it's Gram matrix entry would be updated by (14). Assuming that the data was p th data vector, then the values that we want are $\mathbf{G}_t(p, \text{index}(t))$, $t = 0 \dots d-1$, and $\mathbf{G}_d(p, p)$. By storing the \mathbf{G}_t components of the basis vectors, we can calculate these values. Denoting the column that we are calculating by $\hat{\mathbf{G}}_t$, then algorithm presented in Figure 5 decomposes \mathbf{x}_p .

Note: $\hat{\mathbf{G}}_{t+1}(p, p) = \hat{\mathbf{G}}_t(p, p) - \frac{\hat{\mathbf{G}}_t(t, p)\hat{\mathbf{G}}_t(t, p)}{\mathbf{G}_t(\text{index}(t), \text{index}(t))}$ is the same as $\hat{\mathbf{G}}_{t+1}(p, p) = \hat{\mathbf{G}}_t(p, p) - \alpha_t^2$. Thus we get the relation that we expect: $k(p, p) = \hat{\mathbf{G}}_n(p, p) + \sum_t \alpha_t^2$.

The storage requires for the Gram matrix is $O(n^2)$ where n is the size of the data set. Each entry is a scale versus the vector of the original data set. Even so, a moderate size data set would require external storage

Figure 4: Orthogonalization in terms of the Gram Matrix

1. Create the initial Gram matrix, \mathbf{G}_0 , from the initial mappings $\phi_0(\mathbf{x}_i)$. Initialize $t = 0$.
2. Select a sample i that maximizes a criterion, (such as largest $\mathbf{G}_t(i, i)$).
3. Let the t th vector, \mathbf{v}_t , of the basis be the unit vector in direction of the selected sample, $\phi_t(\mathbf{x}_i)$, see (9). Record $\text{index}(t) = i$. Note: we don't calculate the vector \mathbf{v}_t .
4. Update the Gram matrix. Converting the remaining vectors into their orthogonal components using (10) results in the following update to the Gram matrix:

$$\begin{aligned}
 \mathbf{G}_{t+1}(a, b) &= \phi_{t+1}(\mathbf{x}_a)^T \phi_{t+1}(\mathbf{x}_b) \\
 &= \left(\phi_t(\mathbf{x}_a) - \left(\phi_t(\mathbf{x}_a)^T \mathbf{v}_t \right) \mathbf{v}_t \right) \left(\phi_t(\mathbf{x}_b) - \left(\phi_t(\mathbf{x}_b)^T \mathbf{v}_t \right) \mathbf{v}_t \right) \\
 &= \mathbf{G}_t(a, b) - \frac{\mathbf{G}_t(a, i) \mathbf{G}_t(b, i)}{\mathbf{G}_t(i, i)}
 \end{aligned} \tag{14}$$

5. Increment t . If t less than desired size of basis, d , then GOTO step 2 else exit.

Figure 5: Approximation Algorithm

```

 $\hat{\mathbf{G}}_0(p, p) = k(\mathbf{x}_p, \mathbf{x}_p)$ 
for  $t=0$  to  $d-1$  do
     $\hat{\mathbf{G}}_0(t, p) = k(\mathbf{x}_t, \mathbf{x}_p)$ 
end-for
for  $t=0$  to  $d-1$  do
     $\alpha_t = \frac{\hat{\mathbf{G}}_t(t, p)}{\sqrt{\mathbf{G}_t(\text{index}(t), \text{index}(t))}}$ 
    for  $s = t+1$  to  $d-1$  do
         $\hat{\mathbf{G}}_{t+1}(s, p) = \hat{\mathbf{G}}_t(s, p) - \frac{\hat{\mathbf{G}}_t(t, p) \mathbf{G}_t(\text{index}(t), \text{index}(s))}{\mathbf{G}_t(\text{index}(t), \text{index}(t))}$ 
    end-for
     $\hat{\mathbf{G}}_{t+1}(p, p) = \hat{\mathbf{G}}_t(p, p) - \frac{\hat{\mathbf{G}}_t(t, p) \hat{\mathbf{G}}_t(t, p)}{\mathbf{G}_t(\text{index}(t), \text{index}(t))}$ 
end-for

```

of the Gram matrix. Instead of storing the components of the Gram matrix, only the current approximation of each vector is needed. The orthogonalization can be done by making k passes through the data file as oppose to k passes through a Gram matrix file. The algorithm is presented in Figure 6. This algorithm is equivalent to the incremental algorithm presented in [3]. Finding a basis for large data sets is feasible with the incremental algorithm.

6 Experimental Results

We performed experiments to measure the block IO of nearest neighbor search using kernel distance on the following two real data set.

LIRD: The Letter image database, taken from [11], is the **Letter Image Recognition data (LIRD)** data

Figure 6: Incremental Orthogonalization Algorithm

```

Select first basis vector corresponding to data vector  $\mathbf{x}_s$ .
for each data vector  $\mathbf{x}_p$  do
     $\alpha_0 = \frac{k(\mathbf{x}_p, \mathbf{x}_s)}{\sqrt{k(s, s)}}$ 
     $\hat{\mathbf{G}}_0(p, p) = k(p, p) - \alpha_0^2$ 
end-for
 $t = 0$ 
while need more basis vectors do
    Select  $\mathbf{x}_q$  such that  $\hat{\mathbf{G}}_t(q, q)$  is maximum for next basis vector.
    Let  $\beta_i$ 's and  $\hat{\mathbf{G}}_t(q, q)$  be the current approximation of  $\phi(\mathbf{x}_a)$ .
    for each data vector  $\mathbf{x}_p$  do
         $\alpha_{t+1} = \frac{k(\mathbf{x}_p, \mathbf{x}_q) - \sum_{i=0}^t \alpha_i \beta_i}{\sqrt{\hat{\mathbf{G}}_t(q, q)}}$ 
         $\hat{\mathbf{G}}_{t+1}(p, p) = \hat{\mathbf{G}}_t(p, p) - \alpha_{t+1}^2$ 
    end-for
    Record  $\mathbf{x}_q$ ,  $\hat{\mathbf{G}}_t(q, q)$ , and  $\beta_i$ 's for later use in approximating new vectors.
    increment  $t$ 
end-while

```

set. This data set consists of a large number of black-and-white rectangular pixel arrays as one of the 26 upper-case letters in the English alphabet. The characters are based on 20 Roman alphabet fonts. They represent five different stroke styles and six different letter styles. Each letter is randomly distorted through a quadratic transformation to produce a set of 20,000 unique letter images that are then converted into 16 primitive numerical features. Basically, these numerical features are statistical moments and edge counts.

Image Data: The Hemera Photo-Object image data set consists of 94800 images that are very heterogeneous and having annotated ground truth. To represent the images, a color histogram is created for each region. The histogram has 11 bins (or zones). They are: 0:Red, 1:Orange, 2:Yellow, 3:Skin Color, 4:Green, 5:Cyan, 6:Blue, 7:Purple, 8:Black, 9:Gray, 10:White. The thresholding of the color zones is done in the HVC space. The threshold locations were taken from [8]. The regions were created by a simple partitioning of the image [8]. A total of 14 regions of an image were used. The regions are centered in their respective partitioning scheme. They may overlap with adjacent regions if the image is small. Three scales of an image were also used. The scales used are: 1:1, 2:1, 4:1 (full size, one half size, one fourth size). This results in 462 features for each image.

Two hundred random samples were selected from each data set to used as query locations. The LIRD data was placed in a file with 31 records per block (block size of 1988 bytes). The Image Data was placed in a file with 12 records per block (block size of 22180 bytes). The search for the 10 nearest neighbors was performed for each of the query vectors. A Gaussian kernel was used in calculating the feature space distance.

The result of varying the number of basis vectors used in creating the KVAFfile of LIRD data is presented in Figure 7(a) and Figure 7(b). The results of using a bit approximation and varying the number of bits of the KVAFfile for a fixed number of basis vectors is presented in Figure 7(c) and Figure 7(c) for LIRD data and in Figure 8(a) and Figure 8(b) for Image data. As can be seen from the graphs, increasing the number of basis vectors decrease the average number of data blocks read by creating a better representation in feature space and thus a tighter upper and lower distance bounds. But it also increases the size of the approximation file. Decreasing the number of bits to represent a coefficient of a basis vector in an approximation increases the

average number of data blocks read. But it also decreases the size of the approximation file. For examples, using seven bits per α (basis coefficient) and one hundred basis vector for the Image data results in an approximation file 4.8% the size of the original data file and the ten nearest neighbor search processed, on average, 2.2% of the original data file. Using four bits per α and twenty-five basis vector for the LIRD data set results in an approximation file 20.4% the size of the original data file and the ten nearest neighbor search processed, on average, 6.4% of the original data file.

In addition to the saving in file I/O, the KVAFile using less computation than sequential search. Kernel evaluations are needed only for the actual data visited and the query location. The processing of the approximation file does not require kernel evaluations (beyond the single evaluation of the query).

7 Summary

This paper proposes a KVA-File as an extension of VA-File for kernel-based methods. An efficient approach to approximate vectors in feature space is presented with the corresponding upper and lower distance bounds. Thus an efficient indexing method is provided for kernel based image retrieval methods.

This approach provides two levels of data compression. The first is in the selection of the number of basis vectors. The second level of data compression is in the number of bits to represent the coefficients of an approximated vector. Both components depend on data distribution and the ability of the kernel to capture key components of that distribution. Experimental results on image data sets with high dimensionality demonstrated a computational and I/O efficiency improvement of nearest neighbor search using kernel distances.

References

- [1] S. Amari and S. Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 12(6):783–789, 1999.
- [2] G. D. Anderson, M. K. Vananamurthy, and M. K. Vuorinen. *Conformal Invariants, Inequalities, and Quasiconformal Maps*. Canadian Mathematical Society Series of Monographs and Advanced Texts. John Wiley & Sons, Inc., New York, 1997.
- [3] F. R. Bach and M. I. Jordan. Kernel independent component analysis. Technical Report UCB//CSD-01-1166, University of California, Berkeley, 2001.
- [4] Y. Chen, X. Zhou, and T. Huang. One-class svm for learning in image retrieval. In *Proceedings of IEEE International Conference on Image Processing, Thessaloniki, Greece*, pages 815–818, October 2001.
- [5] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, Cambridge, UK, 2000.
- [6] N. Cristianini, J. Shawe-Taylor, and H. Lodhi. Latent semantic kernels. In C. Brodley and A. Danyluk, editors, *Proceedings of ICML-01, 18th International Conference on Machine Learning*, pages 66–73, Williams College, US, 2001. Morgan Kaufmann Publishers, San Francisco, US.
- [7] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. Abbadi. Vector approximation based indexing for non-uniform high dimensional data sets. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, pages 202–209, November 2000.
- [8] Y. Gong. *Intelligent Image Databases: Toward Advanced Image Retrieval*. Kluwer, New York, 1998.
- [9] D. Heisterkamp, J. Peng, and H. Dai. An adaptive quasiconformal kernel metric for image retrieval. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Kauai Marriott, Hawaii*, pages 236–243, 2001.
- [10] K. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201, March 2001.
- [11] P. Murphy and D. Aha. UCI repository of machine learning databases. www.cs.uci.edu/~mlearn/MLRepository.html.
- [12] B. Scholkopf, C. J. C. Burges, and A. J. Smola, editors. *Advances in kernel methods : support vector learning*. MIT Press, Cambridge, MA, 1999.

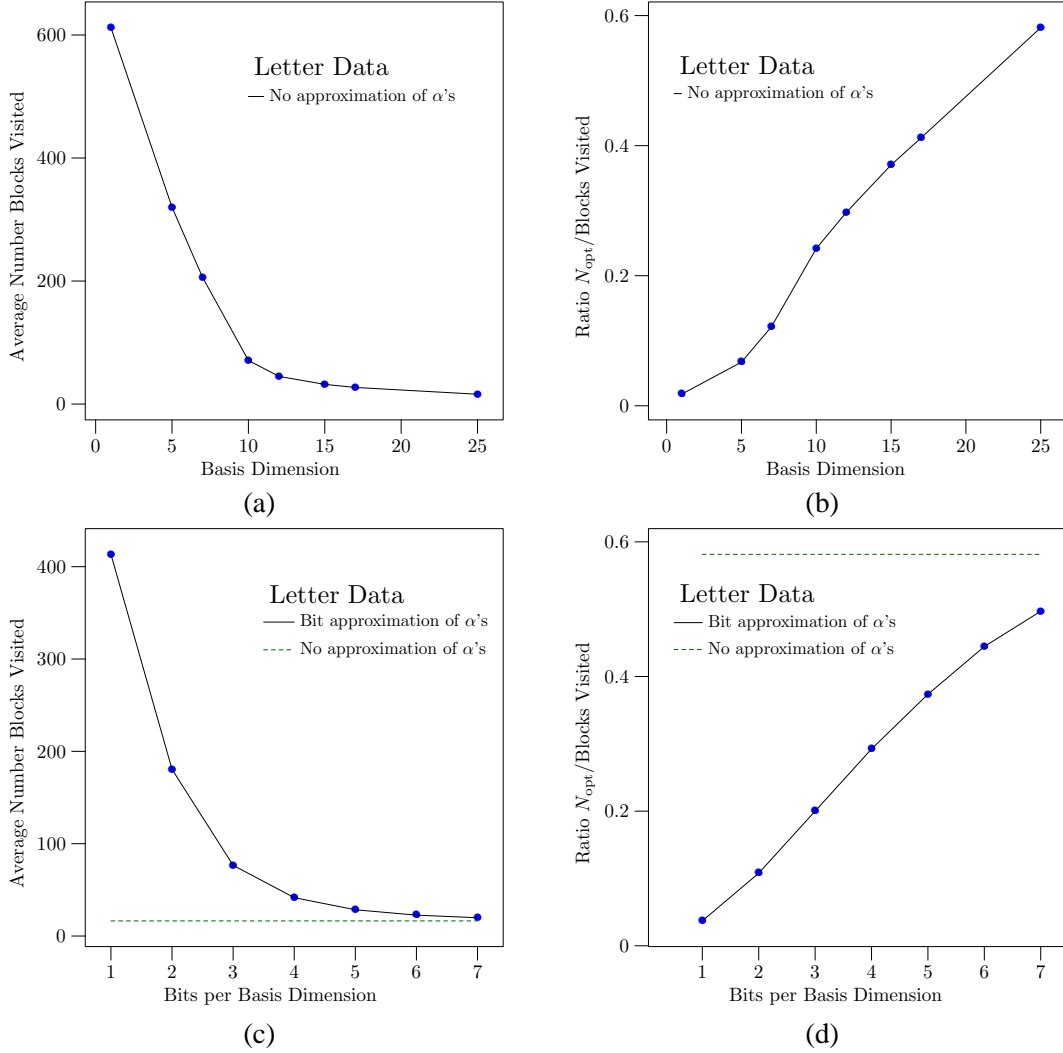
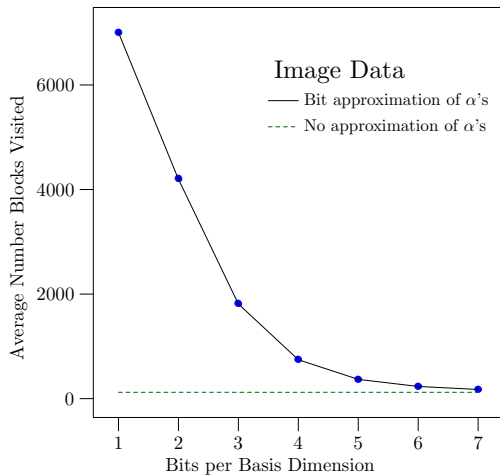
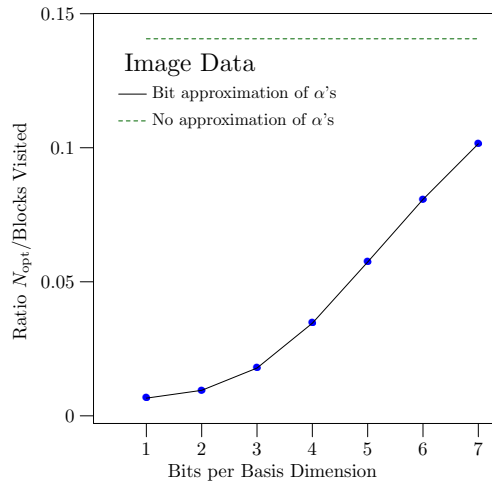


Figure 7: LIRD Data: Ten nearest neighbor search in feature space. The minimum number of blocks to retrieve the ten is N_{opt} . The number of basis dimensions were varied in (a) and (b) with no compression of the α values. The number of bits per basis dimension used in the compression of the α values was varied in (c) and (d). Twenty-five basis vectors are used in (c) and (d). Original input data dimension is 16.

- [13] B. Scholkopf and etal. Input space versus feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, 10(5):1000–1017, September 1999.
- [14] B. Scholkopf, A. Smola, and K.-R. Muller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.
- [15] B. Scholkopf, A. Smola, and K.-R. Muller. Kernel principal component analysis. In *Advances in Kernel Methods - Support Vector Learning*, pages 327–352, Cambridge, MA, 1999. MIT Press.
- [16] A. Smola, O. Mangasarian, and B. Scholkopf. Sparse kernel feature analysis. Technical Report Technical Report 99-03, Data Mining Institute, University of Wisconsin, Madison, 1999.
- [17] A. J. Smola, P. L. Bartlett, B. Scholkopf, and D. Schuurmans, editors. *Advances in Large Margin Classifiers*. MIT Press, Cambridge, MA, 2000.
- [18] D. Tax and R. Duin. Data domain description by support vectors. In *Proceedings of ESANN*, pages 251–256, 1999.
- [19] M. E. Tipping. Sparse kernel principal component analysis. In *Advances in Neural Information Processing Systems*, pages 633–639, 2000.



(c)



(d)

Figure 8: Image Data: Ten nearest neighbor search in feature space. The minimum number of blocks to retrieve the ten is N_{opt} . The number of bits per basis dimension used in the compression of the α values was varied in (a) and (b). One hundred basis vectors are used in (a) and (b). Original input data dimension is 462.

- [20] V. N. Vapnik. *Statistical learning theory*. Adaptive and learning systems for signal processing, communications, and control. Wiley, New York, 1998.
- [21] R. Webber, J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional space. In *Proceedings of the International Conference on Very Large Databases*, pages 194–205, August 1998.
- [22] R. Weber and S. Blott. An approximation-based data structure for similarity search. Technical Report 24, ESPRIT project HERMES (no. 9141), October 1997. Available at <http://www-dbs.ethz.ch/~weber/paper/HTR24.ps>.