INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Efficient Low Rank Approximation via Alternate Least Squares for Scalable Kernel Learning.

by

Piyush Bhardwaj

13111041

A thesis submitted in partial fulfillment for the degree of Master of Technology

in the

Department of Computer Science and Engineering

July 13, 2015

Certificate

It is certified that the work contained in the thesis entitled, **Efficient Low Rank Approximation via Alternate Least Squares for Scalable Kernel Learning** , by **Piyush Bhardwaj** (**13111041**), has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Prof. Harish Karnick
Department of Computer Science and Technology,
Indian Institute of Technology,
Kanpur.

Date:

Dr. Prateek Jain
Researcher, Microsoft Research
Adjucnt Faculty,
Indian Institute of Technology,
Kanpur.

Date:

Contents

Abstract

Kernel methods have demonstrated huge success in modelling non- linearities in real world data. A major challenge in using these methods for large scale data is the high cost involved in computation and storage of the kernel matrix. Kernel approximation is an effective way of dealing with these issues and scaling up kernel machines. Although a long track of work has been done in finding a low rank approximation of the kernel matrix, little effort has been made in leveraging algorithms designed for approximation of general matrices. A major obstacle in extending general matrix approximation schemes to the kernel matrix is their inability to respect the symmetry and the positive semi-definite nature of the kernel matrix. In this work, we take inspiration from the Leverage Element Low rank Approximation (LELA) algorithm, a recent development in general matrix approximation and propose an $O(nr^3 \log n)$ time algorithm for rank r approximation of the kernel matrix. The proposed algorithm is based on random sampling of the entries of the kernel matrix followed by a matrix completion step using alternating least squares (ALS). Empirically, our method shows better performance than the state-of-the-art kernel approximation methods on several standard real life datasets. Theoretically, we show that under certain assumptions on the nature of the kernel matrix, previous results in low rank matrix completion literature can be extended to our proposed scheme, thus, showing convergence to the optimal solution.

Acknowledgement

I am indebted to many people, without whose help this work would have been impossible. First and foremost, I would like to express my gratitude towards Prof. Karnick for supporting me in all the stages of the thesis. Right from introducing me to the subject of machine learning to proof-reading this thesis, his role and influence has been enormous. I would like to thank Dr. Prateek Jain for introducing us to the problem and for his consistent guidance. I would like to thank him for taking out time for the weekly calls and answering my naive queries. I would like to thank Dr. Purushottam Kar for his support and advice. It has been a pleasure working under their guidance and they have been a great influence on my professional and personal life.

Lack of progress in one's work can at times be disheartening. In such times, I was fortunate to share my disappointments with Pooja, Kratika and Subba Rao. Their, own narratives of lack of progress helped me maintain my calm and taught me the value of patience in research. I would like to make a special mention of Subba Rao and Mirza, for discussing some of the material of the thesis in excruciating detail.

I would like to acknowledge the support of the department of computer science and engineering at IITK for providing a conducive environment for research. Coming from a different engineering stream, I was apprehensive about my abilities to understand computer science. I want to thank the faculty of CSE for their excellency in teaching. In particular, I want to thank Prof. Surender Baswana, whose contagious sincerity and humbleness has left a life long impact. Thank you, Sir.

During my last two years at IITK, several new friendships were forged and old ones were strengthened. Their indirect role in giving shape to this thesis is invaluable. In particular, I would like thank Pranay, Animesh, Manish, Sagar, Rustam, T, Deepak and Gautam for lending an ear to my unending whines and also for sharing their unique perspective towards research. A big shout out to all the members of 'the basketball diaries' for providing a constant source of rejuvenation.

Finally, I want to thank my family for their support. Last two years were tough on us and without their support it was impossible to complete this work. I specially want to thank my brother, Late Dr. Prashant Sharma and his family for their understanding, love and inspiration. I wish, I could have been more available when you needed my support.

Piyush Bhardwaj

Chapter 1

Introduction

In the last two decades, kernel methods have garnered a lot of attention in the machine learning community. One of the main reasons for the popularity of these methods is their ability to give good performance on machine learning tasks while providing a firm theoretical understanding. The famous "kernel trick" is a slick and easy way to introduce non-linearity in linear algorithms. It is no wonder that kernelized versions of several classical algorithms have become popular in the last two decades. These include Support Vector Machine (SVM) [1, 2], Kernel Ridge Regression (KRR) [3], Kernel Principal Component analysis (KPCA) [4] and Kernel Canonical Correlation Analysis (KCCA) [5, 6], among others. In this chapter we will introduce the kernel function and highlight the advantages they provide in machine learning tasks. Further, we discuss the challenges associated with kernel learning and use this to motivate the need for a low rank approximation of kernel matrix as a solution to scaling kernel machines. The problem of finding a low rank approximation of a symmetric positive semi-definite (SPSD) kernel matrix is the subject matter of this thesis.

1.1 Kernels

In mathematics and statistics, the word "kernel" can take several different meanings. In machine learning, kernel function (K) can be, crudely, understood as a similarity measure between the members of a set. A kernel function can simply be defined as a real valued bivariate function, $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ is some set. For example, inverse of distance between points x, y in euclidean space ($K(x, y) = -||x - y||^2$) is a natural similarity measure.

Though, kernel functions can be seen as a similarity measure, a more interesting class of kernel functions is the class of functions which represent inner product in some Hilbert space. This class of kernels is known by different terms in the machine learning literature including Mercer kernel, support vector kernel, admissible kernel, non-negative definite kernel, positive definite kernel, symmetric positive semi-definite kernel (SPSD) etc. This class has several properties which make them very interesting for theoretical analysis. In this work we will restrict ourselves to this class of kernels.

In a machine learning setting, the learning algorithm is often provided with some input

data from some set $\mathcal{X}$. Let $\mathcal{D} = \{x_1, \ldots, x_n\}$ be the input data where $x_i \in \mathcal{X}$, $\forall i \in [n]$ are the individual observations belonging to the set $\mathcal{X}$. Let $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a SPSD kernel. As SPSD kernels define an inner-product in some Hilbert space $\mathcal{H}$,

$$K(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathcal{H}}$$

where $\phi(.)$ defines a mapping from input set $\mathcal{X}$ to feature space $\mathcal{H}$,

$$\phi : \mathcal{X} \to \mathcal{H}$$

Though, the definition requires no restriction on the set $\mathcal{X}$, we will see that all the datasets that we consider in our experiments have $\mathcal{X} \subseteq \mathbb{R}^d$.

Given n input points in $\mathcal{X}$ and a kernel function K, we can define the corresponding kernel matrix as $n \times n$ matrix of pairwise inner-products in $\mathcal{H}$ i.e. $K_{ij} = K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$. This matrix is of much practical importance in kernel based learning algorithms. Therefore, it is important to understand some of the properties of this matrix. Observe that, K is a symmetric and positive semi-definite matrix. Symmetry of matrix K follows from the commutative nature of the inner-product, while positive semi-definiteness of K follows from:

$$K = [\phi(x_1) \ldots \phi(x_n)]^T [\phi(x_1) \ldots \phi(x_n)]$$

and for any $y \in \mathbb{R}^n$,

$$\begin{aligned} y^T K y &= \left([\phi(x_1) \ldots \phi(x_n)]\, y \right)^T \left([\phi(x_1) \ldots \phi(x_n)]\, y \right) \\ &= || \left([\phi(x_1) \ldots \phi(x_n)] y \right)^T ||^2 \geq 0 \end{aligned} \tag{1.1}$$

Note that the above property holds irrespective of the value of n i.e. given any n points from the input set, the corresponding kernel matrix K turns out to be SPSD. Therefore, any principal sub-matrix (defined by removing some rows and corresponding columns) of a kernel matrix is also SPSD. In fact, this result holds for any general SPSD matrix. As a result, the diagonal elements $K_{ii} \geq 0$, $\forall i$.

1.1.1 Kernel trick

The popularity of kernel methods can be attributed to a simple trick which provides an easy way to introduce non-linearity in linear algorithms. Kernel functions represent an inner product in feature space, $\mathcal{H}$, whose dimensions depend upon the kernel function itself. Usually, $\mathcal{H}$ is a high dimensional space. For example, the dimensionality of $\mathcal{H}$ for the commonly used Gaussian kernel is infinite. Explicitly working in such a high dimensional space is computationally infeasible. However, kernels provide an inexpensive way to compute the inner product in this space. For input data points in $\mathbb{R}^d$, most of the kernel functions can be computed in $O(d)$ time. The kernel trick exploits this fact by substituting the inner product with the kernel function in the formulation of the learning

algorithm. If in an machine learning algorithm, input data interacts solely based on inner products, kernels provide an inexpensive bridge to a high dimensional space.

Several algorithms in machine learning are classically known to learn linear functions. Kernelized versions of these algorithms are linear in the mapped feature space but in the original input space the learnt function is non-linear. As an example, SVM without kernels learns a linear separator in the input space, given by

$$h(x) = \sum_{i=1}^{n} \alpha_i x_i^T x \tag{1.2}$$

for $x_1, \ldots, x_n, x \in \mathcal{X} \subseteq \mathbb{R}^d$. Using the kernel trick, SVM can inexpensively create a linear separator in the mapped feature space, $\mathcal{H}$.

$$h(x) = \sum_{i=1}^{n} \alpha_i \phi(x_i)^T \phi(x) \tag{1.3}$$

$$= \sum_{i=1}^{n} \alpha_i K(x_i, x) \tag{1.4}$$

As the feature map associated with the kernel can be non-linear, linear classifiers in feature space corresponds to a non-linear classifier in the input space $\mathcal{H}$.

1.2 Challenges in Kernel Learning

Most of the kernel based learning algorithms require the computation of the entire kernel matrix corresponding to the training data set. For training set of size n belonging to $\mathbb{R}^d$, this will typically take $O(n^2 d)$ time. In today's world of ubiquitous data, anything quadratic in n is prohibitive for large scale learning. Additionally, storing the kernel matrix requires $O(n^2)$ space, which hinders scalability as well. To get a sense of the issue at hand, note that Tyree et al.[7] report training time of over 12 days on the MNIST8M dataset using a kernelized SVM. This is a little disappointing given the appealing theoretical understanding of these methods when compared to deep neural networks. Training and testing time notwithstanding, the performance of kernel methods have been reported to match those of deep neural networks on some specific problems [8, 9].

1.2.1 Overcoming scalability issues

A common way of tackling the problem of scalability is to use an approximate sketch of the kernel matrix. Generally, these sketches are obtained as low rank approximations of the kernel matrix. William and Seeger [10] have shown the low rank nature of the kernel matrix for several artificial and real life datasets. This motivates the construction of low rank approximations for the kernel matrix without compromising much on the performance. Once a low rank approximation of the form $K \approx UU^T$ is obtained, it can be utilized by

learning algorithms for speed ups. In terms of space complexity, low rank sketches can be stored effciently. Storing a rank r approximation requires storing the $n \times r$ size matrix U, thus reducing the space complexity to linear in n. A major challenge is to efficiently come up with good approximations for the kernel matrix within time and space constraints.

1.3 Objective and Scope

In this work we propose an approach to construct a low rank approximation of the kernel matrix efficiently. Given n input points $x_1, \ldots, x_n$ in input space $\mathcal{X} \subseteq \mathbb{R}^d$ and a SPSD kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ the kernel matrix is a $n \times n$ matrix with $K_{ij} = K(x_i, x_j)$. The problem of a rank r approximation for K can be stated as finding $U_{n \times r}$ such that

$$\min_{U \in \mathbb{R}^{n \times r}} \sum_{i,j} \left(K_{ij} - e_i^T U U^T e_j\right)^2 \tag{1.5}$$

where e_i is the $n \times 1$ column vector with 1 at i^{th} coordinate and 0 at other coordinates.

Note that the factorization of the sketch as UU^T ensures that the sketch has rank less than or eual to r. The challenge is to find U efficiently without looking at all the n^2 entries of the kernel matrix. This is interesting because every entry of K contributes to the objective function in 1.5. The major contributions of this work are summarized below:

- We provide an $O(n \log n r^3)$ time algorithm to compute a rank r approximation to K.
- We theoretically show the convergence of a simplified version of our algorithm.
- We empirically show the improvement in approximation error when compared to the state-of-the-art methods to solve this problem.
- We show a comparison of different low rank approximation schemes on kernel ridge regression problem

1.4 Structure of the thesis

The thesis is structured in the following manner:

Chapter 2 presents a literature review of methods that try to approximate the kernel matrix. We also look at some algorithms for low rank approximation of a general matrix.

Chapter 3 covers the details of our algorithm. Here, we introduce several sampling schemes and initialization strategies to solve the optimization problem. Finally, we elaborate the details of our algorithm.

Chapter 4 presents the theoretical analysis of our algorithm. We show that if enough number of entries are sampled, convergence to the true optimum will occur with high probability.

Chapter 5 gives empirical evidence for the performance of our algorithm as compared to other kernel approximation algorithms. We also show an application of kernel approximation to the kernel ridge regression problem.

In Chapter 6 limitations of the algorithm are presented and the scope for future work is discussed.

Chapter 2

Literature Review

In this chapter a brief overview of the past research relevant to the thesis is provided. Existing algorithms that try to provide a low rank approximation of the kernel matrix are discussed and their difference from the algorithm proposed in this thesis is pointed out. Research in the field of low rank approximation of general matrices is vast and only relevant algorithms are presented in this chapter.

2.1 Singular Value Decomposition (SVD)

For any general matrix $M \in \mathbb{R}^{n \times m}$, the problem of finding the best rank r can be posed as:

$$\min_{rank(\hat{M}) \leq r} ||M - \hat{M}|| \tag{2.1}$$

where $||.||$ is the spectral norm. It can be shown that the top r singular vectors can be used to construct the optimal rank r approximation, M_r [11]. If the complete SVD of M is given by,

$$M = U_{n \times m} \Sigma_{m \times m} V^T_{m \times m} \tag{2.2}$$

then by choosing the top-r left and right singular vectors, an approximation can be constructed as

$$M_r = U_{n \times r} \Sigma_{r \times r} V^T_{r \times m} \tag{2.3}$$

It can also be shown that the optimal spectral and Frobenius error is given by:

$$||M - M_r||_2 = \sigma_{r+1} \tag{2.4}$$

$$||M - M_r||_F = \left(\sum_{i=r+1}^{m} \sigma_i^2 \right)^{\frac{1}{2}}$$

Computing the top-r singular vectors requires computation of partial SVD which takes $O(nmr)$ time and $O(nm)$ space. For large datasets this is prohibitive in both space and time, specially due to the requirement of computing the entire kernel matrix. The aim of any low rank approximation algorithm is to get errors as close to the optimal SVD errors as possible while doing better than SVD in both space and time complexity. In chapter 3, we compare the error of our proposed algorithm with the optimal SVD error.

2.2 Nystrom Approximations

The first attempt to approximate the kernel matrix were made by Williams and Seeger in their seminal work [10]. They introduce a kernel approximation algorithm based on the theory of linear integral operator defined using kernels.

2.2.1 Integral operators based on kernel function

From our discussion in chapter 1, we know that the kernel matrix defined by a symmetric positive semi-definite (SPSD) kernel function is SPSD. As the entries of this matrix define an inner-product in $\mathcal{H}$, there must be some restriction on the kernel function which generated these entries. Obviously, not all functions can satisfy $K(x, y) = \phi(x)^T\phi(y)$. What kinds of restrictions does such a condition put upon the function K? The question was answered long back in 1909 by Mercer [12] while discussing some theories of integral equations. The result is known as the well known Mercer's theorem in machine learning.

Mercer's theorem takes us to a more general understanding of kernels where they are associated with linear transforms. By associating integration with a kernel function a linear transformation can be defined. Integral transformation T_k of function $f : \mathcal{X} \to \mathbb{R}$ is defined as

$$(T_k f)(x) = \int_{\mathcal{X}} K(x, y)f(y)dy$$

This is a linear transformation as

$$\begin{aligned}(Tf_1 + f_2)(x) &= \int_{\mathcal{X}} K(x, y)(f_1(y) + f_2(y))dy \\ &= \int_{\mathcal{X}} K(x, y)f_1(y)dy + \int_{\mathcal{X}} K(x, y)f_2(y)dy \\ &= Tf_1(x) + Tf_2(x)\end{aligned}$$

where f_1 and f_2 have the same domain. Like matrices, integral operators also have eigenvalues and eigenfunctions. Functions in L_2 which satisfy

$$(T_k\psi)(x) = \int_{\mathcal{X}} K(x, y)\psi(y)dy = \lambda\psi(x) \tag{2.5}$$

are eigenfunctions with λ as eigenvalue. Positive semi-definiteness of T_k implies the following:

$$\int_{\mathcal{X}}\int_{\mathcal{X}} K(x, y)f(x)f(y)dxdy \geq 0, \forall f \in L_2 \tag{2.6}$$

PSD of integral operator implies non-negative eigenvalues and orthonormal eigenfunctions. If $\psi_j(x)$ are the orthonormal eigenfunctions of T_K and λ_j be the corresponding

eigenvalue then:

$$K(x,y) = \sum_{i=1}^{N_{\mathcal{H}}} \lambda_i \psi_i(x)\psi_i(y) \tag{2.7}$$

$$\lambda_i \geq 0 \tag{2.8}$$

$$\int_{\mathcal{X}} \psi_i(x)\psi_j(x)dx = \delta_{ij} \tag{2.9}$$

Using (2.7) we get a decomposition in terms of kernel functions for a positive definite integral operator as

$$K(x,y) = \sum_{i=1}^{N_{\mathcal{H}}} \lambda_i \psi_i(x)\psi_i(y) \tag{2.10}$$

$$= [\sqrt{\lambda_1}\psi_1(x), \dots \sqrt{\lambda_{N_{\mathcal{H}}}}\psi_{N_{\mathcal{H}}}(x)][\sqrt{\lambda_1}\psi_1(y), \dots \sqrt{\lambda_{N_{\mathcal{H}}}}\psi_{N_{\mathcal{H}}}(y)]^T \tag{2.11}$$

$$= \phi(x)^T\phi(y) \tag{2.12}$$

where $N_{\mathcal{H}} \leq \infty$. This defines the restriction that we talked about. For any kernel function to be SPSD kernel, the associated integral operator should be SPSD. Moreover, this gives an insight into the feature map ϕ. From the point of view of the kernel approximation, if λ_is decay, then the kernel function can be approximated by the first few ψ [13, Proposition 1.3.2].

2.2.2 Theory of Nystrom approximations

The Nystrom method is a numerical method to find the eigenfunctions corresponding to an integral operator by replacing the integral with a weighted sum [14]. For finding eigenfunction and eigenvalues of the kernel matrix a natural choice of weights comes from the iid assumption.

$$\int_{\mathcal{X}} K(x,y)\psi(x)p(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} K(x_i,y)\psi(x_i) \tag{2.13}$$

$$= \lambda\psi(y)$$

For $y = x_k$, $k = 1 \dots n$, the above turns into an eigenproblem for matrix $K_{n\times n}$.

$$K\bar{\psi} = n\lambda\bar{\psi} \tag{2.14}$$

where $\bar{\psi} = [\psi(x_1), \dots, \psi(x_n)]^T$. Solving the eigenvalue problem for K, we obtain $u_i^{(n)}$ and $\lambda_i^{(n)}$ as the eigenvectors and eigenvalues of $K_{n\times n}$. From equation 2.14 and the above solution, we can get an estimate for the i^{th} eigenvalue of K as $\lambda_i \approx \frac{\lambda_i^{(n)}}{n}$ and $\sqrt{n}u_i^{(n)} = [\psi_i(x_1) \dots \psi_i(x_n)]^T$. Putting this back in (2.13) gives an approximation of the eigenfunctions:

$$\psi_i(y) = \frac{1}{n\lambda_i}\sum_{j=1}^{n} K(x_j,y)\psi(x_j) = \frac{\sqrt{n}}{\lambda_i^{(n)}} K^{(n)}(y)^T u_i^{(n)} \tag{2.15}$$

for $i = 1 \ldots n$, where $K^{(n)}(y) = [K(x_1, y), \ldots, K(x_n, y)]^T$

To find a low rank approximation of the kernel matrix, we are looking for eigenvectors $u_i^{(n)}$ and eigenvalues $\lambda_i^{(n)}$. In the standard Nystrom approximations proposed by Williams and Seeger [10], $u_i^{(n)}$ is approximated by the eigenvectors of a smaller uniformly sampled kernel matrix. They propose taking a uniform random sample of m points, finding the eigenvectors of the corresponding $m \times m$ kernel matrix as $u_i^{(m)}$ and using them to approximate $u_i^{(n)}$.

$$\psi_i(y) = \frac{1}{m} \sum_{j=1}^{m} K(x_j, y)\psi(x_j) = \frac{\sqrt{m}}{\lambda_i^{(m)}} K^{(m)}(y)^T u_i^{(m)} \tag{2.16}$$

and $\lambda_i = \frac{\lambda_i^{(m)}}{m} = \frac{\lambda_i^{(n)}}{n}$ and therefore $\forall i = 1, \ldots, m$,

$$\lambda_i^{(n)} = \frac{n}{m} \lambda_i^{(m)} \tag{2.17}$$

By equating (2.15) and (2.16) we have:

$$\frac{\sqrt{n}}{\lambda_i^{(n)}} K^{(n)}(y)^T u_i^{(n)} = \frac{\sqrt{m}}{\lambda_i^{(m)}} K^{(m)}(y)^T u_i^{(m)}$$

for $i = 1, \ldots, m$. Putting $y = x_1 \ldots, y = x_n$

$$\frac{\sqrt{n}}{\lambda_i^{(n)}} K_{n \times n} u_i^{(n)} = \frac{\sqrt{m}}{\lambda_i^{(m)}} C_{n \times m} u_i^{(m)} \tag{2.18}$$

where $C_{n \times m}$ represents the sampled kernel matrix i.e $C_{n \times m} = K_{n \times n} S_{n \times m}$ and $S_{n \times m}$ is a column selection matrix.

$$\begin{aligned} \frac{\sqrt{n}}{\lambda_i^{(n)}} \lambda_i^{(n)} u_i^{(n)} &= \frac{\sqrt{m}}{\lambda_i^{(m)}} C_{n \times m} u_i^{(m)} \\ u_i^{(n)} &= \sqrt{\frac{m}{n}} \frac{1}{\lambda_i^{(m)}} C_{n \times m} u_i^{(m)} \end{aligned} \tag{2.19}$$

The computational complexity of the whole process is $O(m^3)$ for solving the $m \times m$ eigenproblem and $O(nmr)$ for finding $u_i^{(n)}$, $i = 1, \ldots r$ for a rank r approximation. Hence, the overall time complexity of the Nystrom Approximation is $O(m^3 + nmr)$. Using equation (2.19) and simple algebra, a rank m approximation using Nystrom method can be written as:

$$\hat{K}_m = C_{n \times m} \, (W)_{m \times m}^{-1} \, C_{n \times m}^T \tag{2.20}$$

where $W = S^T K S$ is a matrix formed by intersection of selected rows and columns of K. Note that $u_i^{(n)}$ as defined in equation 2.19 is just an approximation to the eigenvectors

of K and is not necessarily orthonormal to $u_j^{(n)}$. The eigenvectors and eigenvalues of the Nystrom approximation will in general differ from that defined in equation 2.17 and 2.19.

It seems one can easily construct kernels where the spectrum does not decay using (2.7). A natural question to ask here is whether kernel matrices really have a low rank structure. The question was partially answered by Williams and Seeger [15], where they demonstrate on multiple real life and artificial datasets that the spectrum decays rapidly for commonly used Gaussian kernels.

2.2.3 Nystrom extensions

William and Seeger's work was extended by Drineas and Mahoney [16] to handle cases where W is not invertible. Their extensions used a non-uniform sampling distribution based on the input matrix. They also give theoretical error bounds for their approximations. If the number of sampled columns (m) is greater than $4c\varepsilon^{-2}$ then with high probability,

$$||K - \hat{K}_r||_2 \leq ||K - K_r||_2 + \varepsilon \sum_{i=1}^{n} K_{ii}^2 \tag{2.21}$$

where c is a constant depending upon the high probability parameter. Above bounds aren't very interesting for the Gaussian kernel matrix where $\sum_{i=1}^{n} K_{ii}^2 = n$. In general, for any kernel matrix with bounded diagonal terms i.e. $K_{ii} \leq c$, we get the following high probability guarantee,

$$||K - \hat{K}_r||_2 \leq ||K - K_r||_2 + O\left(\frac{n}{\sqrt{m}}\right) \tag{2.22}$$

The importance sampling based approach of Drineas et al. is expensive because non-uniform sampling probabilities have to be computed.

Zhang et al. [17] proposed k-means clustering to select the columns rather than using a random sampling scheme. Despite its simplicity, their method has shown better performance than other Nystrom extensions. The approximation guarantees they give are based upon the error in the clustering process, e_c and the Frobenius norm of W^{-1}. The right hand side of the bound is minimum when the clusters are of the same size. In that case the bound is:

$$||K - \hat{K}_r||_F \leq 4\frac{n}{m}\sqrt{Cne_c} + Cne_c||W^{-1}||_F \tag{2.23}$$

where C is a constant based on type of kernel. Note that, if W is approximately low rank then $||W^{-1}||_F$ blows up and the bound becomes trivial.

Kumar et al. [18] proposed an extension of Nystrom based on taking the weighted average of several runs of standard Nystrom. Their algorithm is easily parallelizable and the parallel version can be run in approximately the same time as the standard Nystrom. They also improve upon the error bounds of Drineas et al. by using tighter concentration inequalities. Other attempts in the direction of improving error bounds of the Nystrom method have been made by [19, 20, 21, 22]

2.3 Random Feature based Approximations

Random features for kernel approximations were introduced by Recht and Rahimi in their seminal work [23]. Their results are based upon Bochner's theorem, a classical theorem from harmonic analysis which states that a shift invariant kernel if properly scaled can be written as a Fourier transform of a non-negative measure.

$$K(x,y) = K(x-y) = \int_{\mathbb{R}^d} p(w) e^{jw^T(x-y)} dw \tag{2.24}$$

This non-negative measure, $p(w)$, interpreted as a probability distribution can be used to generate random features which approximate the kernel function in expectation.

$$\begin{aligned} K(x-y) &= \int_{\mathbb{R}^d} p(w) e^{jw^T(x-y)} dw \\ &= E_w \left[z_w(x) z_w(y) \right] \end{aligned} \tag{2.25}$$

where $z_w(x) = cos(w^T x)$ and $z_w(y) = cos(w^T y)$ are the random features with w drawn from the distribution $p(w)$. $p(w)$ can be computed as the inverse Fourier transform of the kernel function.

Note that the above result is in expectation and to reduce the variance in approximation one needs to sample multiple random features. However, the number of features required to get a good approximation with high probability is rather large. For ε accuracy in approximating $K(x,y)$ one needs $D = O\left(d\varepsilon^{-2} \log \frac{1}{\varepsilon^2}\right)$ number of random features. Lopez-Paz et al. [24] show bounds on the spectral norm of the kernel approximation using D random features by leveraging the matrix Bernstein inequalities [25, 26]. In particular they show,

$$E\left[||K - \hat{K}_D|| \right] \leq \sqrt{\frac{3n^2 \log D}{D}} + \frac{2n \log n}{D} \tag{2.26}$$

This can be easily extended to a high probability bound.

Lemma 1. *Let $\hat{K}_D$ be an approximation of K formed by using D random Fourier features. If $D \geq \frac{4n^2+2n\varepsilon}{\varepsilon^2} \log \frac{2n}{\delta}$ then with probability greater than $1-\delta$,*

$$||K - \hat{K}_D|| \leq \varepsilon$$

Proof of lemma 1 is provided in appendix A. Note that the above bounds seem to be very loose. For ε accuracy, the number of random Fourier features required is $O(n^2)$. Computing this many random features defeats the purpose of approximation.

For a rank r approximation, one needs to compute r random Fourier features which require $O(nrd)$ time, where d is the dimension of the input space. The number of parameters required to store a rank r approximation are $O(nr)$.

Random Fourier features approximate a shift-invariant kernel in expectation. Random feature maps for several other classes of kernels have been proposed. These include [27, 28, 29] homogeneous, radial basis exponential and dot product kernels respectively.

2.4 Memory Efficient Kernel Approximation

Memory efficient kernel approximation (MEKA) [30] is a method to approximate shift invariant kernel matrices. MEKA works on the observation that the kernel matrix becomes low rank or high rank depending on the value of the kernel bandwidth (σ) parameter. For example, for the Gaussian kernel function (also known as Radial Basis Function (RBF)), given by,

$$K(x, y) = \exp\left(-\frac{||x - y||^2}{2\sigma^2}\right) \tag{2.27}$$

if $\sigma \to \infty$ then the kernel matrix is rank 1, on the other hand if $\sigma = 0$ kernel matrix is full rank. Note that, a full rank kernel matrix is an identity matrix and hence sparse. Entries of the Gaussian kernel are exponentially decaying scaled distances which implies that decreasing the σ parameter will gradually take the off-diagonal terms to 0.

MEKA exploits this fact by clustering the data, $X_{n\times d}$ in the input space and approximating the kernel values corresponding to each cluster by using some kernel approximation technique. Standard Nystrom is taken as the default approximation scheme in the original paper. Inter-cluster values in the kernel matrix are then approximated based on the inter-cluster distance. If the inter-cluster distance is higher than a threshold then the corresponding kernel value is set to 0, otherwise it is approximated by solving a convex optimization problem.

The time complexity of MEKA is the sum of the time complexity of clustering (T_c), the time complexity of approximating inter-cluster blocks (T_L) and the time complexity of approximating intra-cluster blocks. For c clusters and a rank k approximation of each block, the time complexity for approximating the intra-cluster blocks is the sum of $O(nmd)$ time for computing the kernel values and $O(nmk + cm^3)$ time for a rank k Nystrom approximation of each cluster, where m is the number of columns sampled to construct the Nystrom approximation. Thus, the overall time complexity for a rank $r = ck$ approximation is $O(nm(d + k) + cm^3 + T_L + T_c)$.

The space complexity of a rank r approximate, constructed using c clusters depends on the rank approximation (k_i) of the corresponding i^{th} cluster. Overall, the number of parameters required to construct a rank r approximation is given as

$$\sum_{i=1}^{c} n_i k_i + \sum_{i,j} k_i k_j \tag{2.28}$$

where $\sum_{i=1}^{c} k_i = r$ and n_i is the number of points in the i^{th} cluster, returned by the clustering process.

If each block is approximated by a rank k matrix, then $k_i = k, \forall i \in [c]$. Hence, The number of parameters required to store a rank $r = ck$ approximation is $O(nk + (ck)^2)$. Note that the above computation assumes that $n_i \geq k$, if $n_i < k$, then the i^{th} block can have a maximum rank of n_i and hence, $r < ck$.

The Guarantees of MEKA's approximations are based on those of standard Nystrom. If the gap between the $k+1^{st}$ and the $ck+1^{st}$ singular values is large and the inter cluster entries are small then MEKA has better approximation error than Nystrom which uses the same space.

In most applications σ is tuned using cross-validation. MEKA claims to work irrespective of the value of σ. Though an interesting idea, we found the performance of this method to be unstable (high variance) and difficult to tune, in our experiments. This behaviour has been reported by other researchers as well [31, 32]

2.5 Leveraged Element Low Rank Approximation

Low rank approximation of a matrix is a fundamental problem in several scientific communities including machine learning. The problem of low rank approximation with limited number of passes over the matrix was first introduced by Frieze et al. in [33]. There has since been a long track of work in this direction. The current state of the art algorithm, known as Leverage Element Low Rank Approximation (LELA), was given by Bhojanapalli et al. in [34]. Their algorithm runs in input sparsity time ($O(nnz(M))$ and gives guarantees in spectral norm. This is in contrast with the previous best algorithm by Clarkson and Woodruff [35], which also ran in input sparsity time but gave guarantees in the weaker Frobenius norm.

LELA works by non-uniformly sampling a few entries of the matrix and then doing a weighted alternating least square to perform matrix completion. Random samples are chosen based on an weighted distribution computed using the column norms of the matrix and the absolute value of entries of the matrix. Let $M_{n\times d}$ be the matrix to be approximated. Every element of M is sampled using a non-uniform distribution given by:

$$q_{ij} = m\left(0.5\frac{||M_i||^2 + ||M^j||^2}{(n+d)||M||_F^2} + 0.5\frac{|M|_{ij}}{||M||_{1,1}}\right) \tag{2.29}$$

where m is the desired sample size, M_i denotes a vector of elements of the i^{th} row of M, M^j denotes a vector of elements of the j^{th} column of M and $||M||_{1,1} = \sum_{i,j}|M_{ij}|$. This kind of distribution is carefully chosen so that the $L1$ term gives importance to the heavier element of the matrix, thus accounting for the possible high rank nature of the matrix.

LELA uses the sampled entries to perform a matrix completion step. It utilizes alternating least square minimization for matrix completion. Let $R_\Omega(M)$ be the sampled matrix with m sampled entries and Ω be the set of sampled entries, then LELA solves the following optimization problem over factors $U_{n\times r}$ and $V_{d\times r}$:

$$\min_{U,V}\sum_{\Omega}\frac{1}{q_{ij}}\left(M_{ij} - e_i^T U V^T e_j\right)^2 \tag{2.30}$$

The problem is inherently non-convex in U and V but becomes convex if one of U or V is fixed. LELA takes advantage of this by solving the problem alternatively. The authors

give a provable algorithm which converges to the optimal SVD solution if enough number of entries of M are sampled. In fact, it can be shown that the update step using alternative least square is same as the update step of the power method.

One of the key elements of LELA is the initialization of U and V by an SVD of sparse $R_\Omega(M)$. It can be shown that this is a good initialization point by showing its closeness to the optimal subspaces, U^*, V^*, obtained by SVD. For $M_{n\times d}$, LELA returns a rank r approximation in $O(nnz(M) + mr^2)$ time, where m is the number of randomly sampled entries of M. The number of parameters required to store the approximation is $O\left((n+d)\,r\right)$.

Chapter 3

Kernel Approximation using ALS

In this chapter, we will discuss our proposed algorithm for a low rank approximation of the kernel matrix. This method can be seen as a modification of the LELA algorithm, discussed in section 2.5. We start with the problem formulation and define some notation. We then discuss the challenges in extending LELA for kernel matrix approximation. This is followed by a description of our algorithm and a discussion on different initialization and sampling strategies. In chapter 4 we discuss the theoretical guarantees of our method, followed by empirical results in chapter 5.

3.1 Mathematical Notation

For the rest of the thesis following notation is used: A matrix is represented by the capital letter (M) and a vector is represented by the small letter (u). M_{ij} represents the $(i,j)^{th}$ element of the matrix M, M_i represents the i^{th} column of M and M^j represent j^{th} row of M transposed. M^T and M^{-1} represent the transpose and inverse of M, respectively. $nnz(M)$ denotes the count of the number of non-zero entries in the matrix M. $||M||$ and $||M||_F$ represent the 2-norm and the Frobenius norm of the matrix M. $||u||_p$ gives the p-norm of vectors u. At times, we mention the dimension of the matrix ($M_{n\times n}$) as a subscript for clarity.

$X_{n\times d}$ represents the data with n observations in d dimensional space. K is used to represent the $n \times n$ kernel matrix corresponding to X. Ω is used to represent the set of indices corresponding to the sampled entries of the matrix. $R_\Omega(K)$ denotes the sampled version of K, with $R_\Omega(K)_{ij} = K_{ij}$ if $(i,j) \in \Omega$ and 0 otherwise. $m = |\Omega|$, denotes the number of samples.

The problem formulation was done in section 1.3. Here it is presented for the sake of completeness. Given n input points $x_1, \ldots, x_n$ in input space $\mathcal{X} \subseteq \mathbb{R}^d$ and an SPSD kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, the associated kernel matrix is a $n \times n$ matrix with $K_{ij} = K(x_i, x_j)$. The problem of rank r approximation of matrix K can be stated as finding $U_{n\times r}$ such that

$$\min_{U\in\mathbb{R}^{n\times r}} \sum_{i,j} \left(K_{ij} - e_i^T U U^T e_j\right)^2 \tag{3.1}$$

where e_i is the $n \times 1$ column vector with 1 at i^{th} coordinate and 0 at other coordinates.

3.2 Extending LELA

LELA was discussed in section 2.5 as an approximation method for the low rank approximation of any general matrix. As discussed earlier, the time complexity for running LELA is $\left(O(nnz(K) + mr^2\right)$. Owing to the possibly dense nature of the kernel matrix, $nnz(K) = O(n^2)$. Clearly, this leads to scaling issues for large data sets. The major bottleneck in LELA is the importance sampling step which requires computation of the row and column norms of the matrix and therefore forces computation of the complete kernel matrix (see equation 2.29). In section 3.4, we propose and compare several sampling strategies, which can be computed efficiently.

The second computational bottleneck in extending LELA comes from the initialization step. The initialization step of LELA requires SVD of the sparse matrix $R_\Omega(K)$. Numerically, special iterative methods have been designed to find few large eigenvalues and eigenvectors of a large sparse matrix. However, if the number of eigenvectors required is large then these methods don't give good empirical performance. We propose and compare several efficient initialization strategies to overcome this obstacle in section 3.5.

As LELA is a general matrix approximation algorithm it provides an approximation of the form UV^T. However, for SPSD kernel matrix we need factorization of the form UU^T. This issue is tackled by solving the optimization problem for U and V and then taking average of U and V. We show in our results that this strategy leads to low errors.

Once sampling is done the problem is solved by minimizing least square error over the sampled entries (equation 2.30). The hope is that the learned U,V matrix will provide good approximation for the unsampled entries as well. Though, ALS solves a convex optimization problem in each iteration, it involves solving n least square problems in each iteration. Therefore, the time complexity of running each iteration is $O(nr^3)$ which can become challenging for large r.

3.3 Basic ALS Algorithm

Algorithm 1 shows the basic skeleton of our method. Further details are later filled in based on the empirical evidence. In step 1, $R_\Omega(K)$ is the sparse matrix created based on some sampling scheme. The solution is initialized as $\hat{U}^{(0)}$ following some initialization strategy in step 2. Step 4 and 5 are the iterative steps of ALS.

Note that, step 4 solves a weighted least square problem with weight of each sampled entry being the inverse of the probability of sampling that entry i.e. $w_{ij} = \frac{1}{p_{ij}}$, where p_{ij} is the probability of sampling the $(i,j)^{th}$ entry. One advantage of taking a weighted objective function is that in expectation the error value is same as that of non weighted objective function created by sampling all the entries. For the uniform distribution this probability based weight can be ignored. Also, note that the problem solved in step 4 of algorithm 1 is a regularized least square problem and hence a convex optimization problem. For ease

Algorithm 1 Basic Algorithm with ALS

Input: data X, target rank r

Output: $\hat{U}_{n\times r}$

1: Sample K to create $R_\Omega(K)$

2: Initialize $\hat{U}^{(0)}$

3: **for** $t = 0, \ldots, T-1$ **do**

4: $\quad \hat{V}^{(t+1)} \leftarrow \text{argmin}_{V_{n\times r}} \sum_{(i,j)\in\Omega} w_{ij} \left(K_{ij} - e_i^T \hat{U}^{(t)} V^T e_j\right)^2 + \lambda ||V||_F^2$

5: $\quad \hat{U}^{(t+1)} \leftarrow \frac{1}{2}\left(\hat{V}^{(t+1)} + \hat{U}^{(t)}\right)$

of computation it can be sub-divided into n independent least square problems as:

$$\min \sum_{j=1}^{n} \left(\sum_{i:(i,j)\in\Omega} w_{ij} \left(K_{ij} - e_i^T \hat{U}^{(t)} V^j\right)^2 + \lambda ||V^j||^2 \right) \tag{3.2}$$

Each of these n problems can be solved independently for $V^j,\ j = 1, \ldots, n$. This lets us use a parallel solver.

Note that we use a regularized version of problem 2.30. This is to prevent over fitting. As the optimization objective function in 2.30 focuses solely on the sampled entries, there is no check over the approximation of unsampled entries. If the number of entries sampled are not sufficient, this may cause the optimization process to take the unsampled entries to large values. Regularization ensures that the unsampled entries don't grow unrestrained. The λ parameter can be tuned using croos-validation or a test set.

Solving each of the n optimization problems in 3.2 takes $O(|\Omega_j|r^2 + r^3)$ time, where $|\Omega_j|$ is the number of entries sampled from the j^{th} column of K. Thus, overall time complexity of running 1 iteration of the *for* loop in step 3 of the above algorithm is $O(|\Omega|r^2 + nr^3)$.

3.4 Sampling Schemes

In this section we describe several sampling schemes which can be computed efficiently. The results of the sampling schemes are discussed later in section 5.2.1.

1. **L1 sampling**: Probability of sampling $(i,j)^{th}$ element is given by

$$p_{ij} = \min\left(1, \frac{m|K_{ij}|}{||K||_1}\right) \tag{3.3}$$

where m is the requisite number of samples. This kind of sampling gives more importance to the heavier elements of the matrix, ensuring that the sampled sparse matrix takes care of the arbitrary high rank nature of the original matrix. Computationally, 3.3 requires calculating the entire kernel matrix and therefore is infeasible for large datasets. Note that the expected number of samples using the above distribution is m.

2. **L2 sampling**: Probability of sampling $(i,j)^{th}$ element is given by

$$p_{ij} = \min\left(1, \frac{m\left(||K^i||^2 + ||K_j||^2\right)}{2||K||_F^2}\right) \tag{3.4}$$

This requires computation of the entire kernel matrix and is therefore infeasible to compute for large datasets. Note that the expected number of samples using the above distribution is m.

3. **Approximate L2 sampling**: We can use the random Fourier features [23] to approximate the kernel matrix and then use this approximated matrix to further approximate row and column norms. Let $X_{n\times d}$ be the data matrix, then the random Fourier features corresponding to i^{th} data point x_i is given by

$$\phi(x_i) = \sqrt{\frac{2}{D}}\left[\cos(\omega_1^T x_i + b_1) \ \ldots \ \cos(\omega_D^T x_i + b_D)\right] \tag{3.5}$$

where D is the total number of random features and w_i random vectors are drawn from a distribution based on the inverse Fourier transform of kernel function K. For shift-invariant kernels $K(x_i, x_j) = E\left[\phi(x_i)^T\phi(x_j)\right]$.

In order to compute 3.4, we need to approximate $||K_i||$.

$$||K_i||^2 \approx \sum_{j=1}^{n}\left(\phi(x_i)^T\phi(x_j)\right)^2 \tag{3.6}$$

$$= \sum_{j=1}^{n}\left(\phi(x_i)^T\phi(x_j)\phi(x_j)^T\phi(x_j)\right) \tag{3.7}$$

$$= \phi(x_i)^T\left(\sum_{j=1}^{n}\phi(x_j)\phi(x_j)^T\right)\phi(x_i) \tag{3.8}$$

Computing D random Fourier features, as in equation 3.5, requires $O(ndD)$ time, while computing the column norm via equation 3.8 takes $O(nD^2)$ time. Typically $D \gg d$, thus, we can calculate row leverage scores in $O(nD^2)$ time for all $i \in [n]$, by calculating and storing $\sum_{j=1}^{n}\phi(x_j)\phi(x_j)^T$. The same follows for the column leverage scores.

4. **Uniform**: Entries are sampled randomly uniformly from the matrix. This is the most efficient way of sampling as it doesn't require computation of any distribution.

5. **Uniform+ Diagonal ($U+D$)**: Similar to uniform clustering except diagonal terms are added deterministically.

6. **Clustering**: In this method k-means clustering of input data, $X_{n\times d}$, is performed. Kernel values corresponding to the data point closest to the centroid of each cluster

are added deterministically to the sample. For each cluster center x_{c_i}, kernel value corresponding to interaction of c_i with rest of the data points is added to the sample.

$$\Omega \leftarrow \Omega \cap \{(c_i, 1), (c_i, 2) \ldots, (c_i, n),\ (1, c_i), (2, c_i), \ldots, (n, c_i)\}$$

Number of clusters to be created depends on number of elements to be sampled.

7. **Uniform+clustering($U + C$)**: Half the entries are chosen by uniform sampling and rest of the entries are sampled using the clustering procedure.

8. **Uniform+clustering+diagonal ($U + C + D$)** Similar to $U + C$ but diagonal term of the kernel matrix are added deterministically to the sample.

Section 5.2.1 shows an empirical comparison of the above sampling schemes. In summary, we found that $U + C + D$ performs better than other sampling schemes across a range of kernel matrices.

3.5 Initialization Strategies

Initialization of iterate $\hat{U}^{(0)}$ in step 2 of algorithm 1 is an important part of our method. A good initialization step ensures low errors in a few iterations. In this section, we compare several initialization strategies viz. random, Nystrom, kNystrom and sparse SVD. Following is a brief explanation of the initialization strategies.

1. **Random**: Each element of $\hat{U}^{(0)}$ is chosen uniformly randomly from $[0, 1]$.

2. **Nystrom**: This method selects $4r$ columns uniformly and performs Nystrom approximation. Results of Nystrom approximations are passed as an initialization to ALS.

3. **KNystrom**: This method selects $4r$ columns based on k-means clustering of the data in the input space. Results of k-means Nystrom are passed as an initial iterate to ALS.

4. **Sparse SVD**: Sparse SVD of the sampled kernel matrix $R_\Omega(K)$ is done to find the top-r scaled singular vectors.

An empirical comparison of different initialization strategies is shown in section 5.2.2. We found that kNystrom based initialization gives the best results.

3.6 ALS for kernel approximation

Using empirical evidence from section 3.4 and 3.5, we are now ready to give the final version of our algorithm. In this version, we have filled in the following details about sampling and initialization.

1. Sampling using a mixture of uniform sampling, clustering and deterministically selecting the diagonal terms is done.

2. K-means based Nystrom is used for initialization.

Algorithm 2 gives the final algorithm.

Algorithm 2 ALS based algorithm for kernel low rank approximation

Input: data X, target rank r, sample size m

Output: $\hat{U}_{n\times r}$

1: $\Omega \leftarrow$ Sample $\frac{m}{4}$ elements $(i,j) \in [n] \times [n]$
2: **for** each $(i,j) \in \Omega$ **do**
3: $\quad \Omega \leftarrow \Omega \cup (j,i)$
4: $\Omega \leftarrow \Omega \cup (i,i),\ \forall i \in [n]$
5: Create $\frac{m}{2n}$ clusters using $kmeans(X)$
6: **for** each cluster center i **do**
7: $\quad \Omega \leftarrow \Omega \cup (i,j),\ \forall j \in [n]$
8: Initialization: $\hat{U}^{(0)} = KNystrom(X, 4r)$
9: **for** $t = 0, \ldots, T-1$ **do**
10: $\quad \hat{V}^{(t+1)} \leftarrow \text{argmin}_{V_{n\times r}} \sum_{(i,j)\in\Omega} \left(K_{ij} - e_i^T \hat{U}^{(t)} V^T e_j\right)^2 + \lambda ||V||_F^2$
11: $\quad \hat{U}^{(t+1)} \leftarrow \frac{1}{2}\left(\hat{V}^{(t+1)} + \hat{U}^{(t)}\right)$

Step 1 to 3 sample half the required number of elements uniformly. Uniform sampling is done by sampling both i and j uniformly from $[0, n]$ and rounding them to the next higher integer. Note that the sampling is done in a way to ensure that the sampled matrix is symmetric. In step 4, diagonal entries are added deterministically. Clustering in step 5 is done on standardized zero mean and unit variance data. As shown in equation 3.2, the optimization problem in step 10 of algorithm 2 is broken down into n independent least square problems. Each of these sub-problems is solved in parallel using a conjugate gradient descent based iterative solver. The advantage of using an iterative solver over classical matrix factorization techniques is that it allows for a warm start for the solution leading to quicker convergence. For example, in the $(t+1)^{th}$ iteration, we can pass $U^{(t)}$ as an initial estimate of $V^{(t+1)}$.

3.6.1 Time and space complexity

In terms of time complexity:

- Step 1 to 4, take $O(|\Omega|)$ time for uniform sampling.
- If the number of clusters is c, each iteration of k-means clustering takes $O(ndc)$ time. For us, $c = \frac{|\Omega|}{2n}$ and thus the time for clustering in step 5 is $O(|\Omega| d T_C)$, where T_C is the number of iterations of k-means.

- K-means Nystrom initialization in step 8 takes $O(r^3+nr^2+rdT_c)$ time for performing k-means with $4r$ clusters and computing the approximation.
- The ALS procedure in step 9 to 11 takes $O(|\Omega|r^2+nr^3)$ in each iteration. Note that by solving the optimization problem approximately by using an iterative solver like conjugate gradient method, the r^3 term can be avoided.

Thus the overall time complexity of our method is

$$O\left(|\Omega|dT_C + rdT_c + \left(|\Omega|r^2 + nr^3\right) T_{ALS}\right)$$

where T_{ALS} is the number of iterations of ALS. In our experiments we have found, T_{ALS} of around 3 is enough to give good results. Also typically, $\left(|\Omega|r^2 + nr^3\right) T_{ALS}$ will dominate other terms and time complexity will reduce to $O\left(|\Omega|r^2 + nr^3\right)$. In our experiments, we have found that number of sampled elements scale as $O(rn \log n)$, therefore, practically our algorithm has a time complexity of $O(n \log nr^3)$ for a rank r approximation.

In terms of space complexity, for storing a rank r approximation we require $O(nr)$ space. Table 3.1 shows the time complexity of other kernel approximation algorithms with almost the same space constraints. Note that, m is the number of columns sampled in Nystrom, c_m is the input number of clusters in MEKA and T_{Lm} and T_{Cm} are the time required for approximating off-diagonal blocks and clustering, respectively in MEKA.

Algorithm	Space complexity	Rank	Time Complexity
Nystrom	$O(nr)$	r	$O(m^3 + nmr) \approx O(nr^2)$
SVD	$O(nr)$	r	$O(n^2 r)$
ALS	$O(nr)$	r	$O\left(\|\Omega\|r^2 + nr^3\right) \approx O(n \log nr^3)$
MEKA	$O(nr + c_m^2 r^2)$	$c_m r$	$O(nr^2 + c_m m^3) + T_{Lm} + T_{Cm}$

Table 3.1: Comparison of time and space complexities of different algorithms for kernel approximation

Clearly, SVD is the slowest among these algorithm. MEKA seems slower than Nystrom, but note that it gives a higher rank approximation with almost the same number of parameters. ALS is slower than most of the algorithms, but as we will see in section 5.2.3 it has lower approximation errors. An empirical comparison between the algorithms is shown in section 5.2.3. We also do a comparison of kernel approximation to kernel ridge regression in section 5.3

Chapter 4

Theoretical Analysis

In this chapter we show theoretical results for the approximation of kernel matrix using ALS. The structure of the proofs is directly borrowed from the low rank matrix completion proofs in [36].

4.1 Preliminaries

This section introduces the definitions and the notation used in the following sections. Let M be any general matrix. Let U_M and V_M denote the left and right singular vectors of M such that $M = U_M \Sigma_M V_M^T$. Let u_i and v_i be the i^{th} column of U and V, respectively. The best rank r approximation of M is denoted by M_r and is formed by the top-r left and right singular vectors of M, $M_r = U_M^{(r)} \Sigma_M^{(r)} V_M^{(r)T}$, where $\Sigma_M^{(r)} \in \mathbb{R}^{r \times r}$ is a diagonal matrix formed by taking top-r diagonal entries of Σ_M in the original order. Similarly, $\Sigma_{M(r)} \in \mathbb{R}^{r \times r}$ is a diagonal matrix formed by taking bottom-r diagonal entries of Σ_M in the original order. For $rank(M) = k$, the pseudo-inverse is defined as $M^\dagger = \sum_{i=1}^{k} \frac{1}{\sigma_i} u_i v_i^T$. $||M||$ and $||M||_F$ denote the spectral and the Frobenius norm of matrix M, respectively.

For rank r approximation, $\hat{U}^t \in \mathbb{R}^{n \times r}$ denotes the solution returned by ALS at the end of the t^{th} iteration. U^t denotes an orthonormal basis spanning the column space of $\hat{U}^t$. The optimal rank r approximation of K is $K_r = U^* \Sigma_K^{(r)} U^{*T}$, where $U^* = U_K^{(r)}$.

The notion of matrix coherence measures the uniformity of entries of a matrix. Loosely speaking, incoherence is a measure of closeness of the orthonormal basis spanned by the column space of a matrix to the canonical basis. Formally, incoherence can be defined as following:

Definition 1. *[37] A matrix $M \in \mathbb{R}^{n \times m}$ is incoherent with parameter μ_r if:*

$$||u^{(i)}|| \leq \frac{\mu_r \sqrt{r}}{\sqrt{n}}, \forall i \in [n], \; ||v^{(j)}|| \leq \frac{\mu_r \sqrt{r}}{\sqrt{n}}, \forall j \in [m], \tag{4.1}$$

where $M = U \Sigma V^T$ is the SVD of M and $u^{(i)}$,$v^{(j)}$ denote the i^{th} row of U and the j^{th} row of V respectively.

Incoherence parameter μ_r is upper bounded by $\sqrt{\frac{n}{r}}$ when the top r singular vectors are exactly same as the canonical basis vectors. It is lower bounded by 1 when all the entries

are equal to $\frac{1}{\sqrt{n}}$.

$$1 \leq \mu_r \leq \sqrt{\frac{n}{r}} \tag{4.2}$$

Previous literature on matrix completion shows that highly coherent matrices are difficult to recover by random sampling and matrix completion [37].

Principal angles between subspaces is a common way to quantify the distance between subspaces.

Definition 2. *[11] Given two matrices $\hat{U}$, $\hat{W} \in \mathbb{R}^{n \times r}$, the principal angle based distance between the subspaces spanned by the columns of $\hat{U}$ and $\hat{W}$ is given by:*

$$dist(\hat{U}, \hat{W}) = ||U_\perp^T W|| = ||W_\perp^T U|| \tag{4.3}$$

where U and W are orthonormal bases spanning the column space of $\hat{U}$ and $\hat{W}$ respectively. $U_\perp$ and $W_\perp$ denote the orthonormal basis spanning the subspace perpendicular to $\hat{U}$ and $\hat{W}$ respectively.

In fact, $dist(\hat{U}, \hat{W}) = \sin\Theta = ||U_\perp^T W||$, where Θ is the largest principal angle between subspaces U and W. We use the following basic inequality from linear algebra in our proofs.

$$d_{\min}||A|| \leq ||AD|| \leq d_{\max}||A|| \tag{4.4}$$

$$||A^T|| = ||A|| \tag{4.5}$$

for any matrix A and any diagonal matrix D with $d_{\min}$ and $d_{\max}$ as the lowest and the highest entries of matrix D, respectively.

4.1.1 Proof summary

We make assumptions regarding the coherence of the kernel matrix, which is a standard assumption in matrix completion literature. The proof proceeds by showing that the subspaces spanned by the solution of ALS gets iteratively closer to the optimal subspace spanned by the top-r singular vectors of the original kernel matrix. We use the following inductive structure for the proof:

1. Base case: Show that U^0 is close to U^* and U^0 is incoherent

2. Inductive hypothesis: Assume U^t is close to U^* and U^t is incoherent

3. Inductive step: Show that U^{t+1} is close to U^* and U^{t+1} is incoherent

4.2 Initialization

In this section we show that the distance between the initial iterate, $\hat{U}^0$ returned by the Nystrom approximation is close to U^*. As the initialization is done using the Nystrom approximation, we take a closer look at the structure of the Nystrom approximation.

As K is SPSD we can factorize it as

$$K = X^T X \tag{4.6}$$

$$\text{where, } X = \Sigma_K^{\frac{1}{2}} U_K^T \tag{4.7}$$

Let $S \in \mathbb{R}^{n \times m}$ be a column selection matrix, then $S^T K S \in \mathbb{R}^{m \times m}$ is the matrix formed by the intersection of the selected m rows and the m columns of K. The rank r Nystrom approximation is of the form

$$\hat{K} = KS(S^T K S)_r^\dagger S^T K \tag{4.8}$$

This can be simplified in terms of $B = XS = U_B \Sigma_B V_B^T$.

$$\begin{aligned}
\hat{K} &= X^T X S (S^T X^T X S)_r^\dagger S^T X^T X && (4.9)\\
&= X^T B (B^T B)_r^\dagger B^T X \\
&= X^T U_B \Sigma_B V_B^T \left(V_B \Sigma_B U_B^T U_B \Sigma_B V_B^T\right)_r^\dagger V_B \Sigma_B U_B^T X \\
&= X^T U_B \Sigma_B V_B^T \left(V_B \Sigma_B^2 V_B^T\right)_r^\dagger V_B \Sigma_B U_B^T X \\
&= X^T U_B \Sigma_B V_B^T \left(V_B^{(r)} {\Sigma_B^{(r)}}^{-2} V_B^{(r)T}\right) V_B \Sigma_B U_B^T X \\
&= X^T U_B^{(r)} U_B^{(r)T} X \\
&= X^T U_{B'} {U_{B'}}^T X && (4.10)\\
&= X^T P_{B'} X && (4.11)
\end{aligned}$$

where in equation 4.10, $U_{B'} = U_B^{(r)}$. Note that, $P_{B'} = U_{B'} U_{B'}^T$ is the projection matrix into the column space spanned by the top r columns of XS. From equation 4.7, it can be observed that the column space of X is $\mathbb{R}^n$. If S samples all the columns of X and $r = m$ then $\mathcal{C}(B') = \mathcal{C}(X)$ and $P_{B'} = I_n$. This leads to the exact recovery of the kernel matrix. However, S can sample at most m columns of K and $P_{B'}$ can at most be rank r. If S is such that $P_{B'} = \begin{pmatrix} I_r & 0 \\ 0 & 0 \end{pmatrix}$ then,

$$\begin{aligned}
\hat{K} &= X^T P_{B'} X \\
&= U_K \Sigma_K^{\frac{1}{2}} \begin{pmatrix} I_r & 0 \\ 0 & 0 \end{pmatrix} \Sigma_K^{\frac{1}{2}} U_K^T \\
&= U_K \begin{pmatrix} \Sigma_K^{(r)} & 0 \\ 0 & 0 \end{pmatrix} U_K^T \\
&= U_K^{(r)} \Sigma_K^{(r)} U_K^{(r)T} = K_r && (4.12)
\end{aligned}$$

Thus, setting $P_{B'} = \begin{pmatrix} I_r & 0 \\ 0 & 0 \end{pmatrix}$ gives the optimal rank r approximation. This observation gives us the intuition that the distance between K and $\hat{K}$ may be directly related to the distance between $P_{B'}$ and $\begin{pmatrix} I_r & 0 \\ 0 & 0 \end{pmatrix}$. Further we show the relationship of this distance with K and $\hat{K}$ in the lemma below.

Lemma 2. *For rank r Nystrom approximation, the distance between the optimal subspace, U^* and the subspace spanned by the columns of $\hat{K}$, $U_{\hat{K}}$ is bounded by:*

$$\sqrt{\frac{\sigma_n(K)}{\sigma_1(\hat{K})}}||U_{B'(n-r)}|| \leq dist(U^*, U_{\hat{K}}) \leq \sqrt{\frac{\sigma_{r+1}(K)}{\sigma_r(\hat{K})}} \tag{4.13}$$

where $U_{B'(n-r)} \in \mathbb{R}^{n-r \times r}$ is the last $n-r$ rows of orthonormal matrix $U_{B'} \in \mathbb{R}^{n \times r}$.

Proof. Let SVD of $\hat{K}$ be $\hat{K} = U_{\hat{K}} \Sigma_{\hat{K}} U_{\hat{K}}^T$ and SVD of K be $K = U_K \Sigma_K U_K^T$, then by equation 4.10

$$U_{\hat{K}} \Sigma_{\hat{K}} U_{\hat{K}}^T = U_K \Sigma_K^{\frac{1}{2}} U_{B'} U_{B'}^T \Sigma_K^{\frac{1}{2}} U_K^T \tag{4.14}$$

Multiplying by $U_\perp^*$ on both sides, taking norm and using $||A^T A|| = ||A||^2$

$$||U_\perp^{*T} U_{\hat{K}} \Sigma_{\hat{K}}^{\frac{1}{2}}|| = ||U_\perp^{*T} U_K \Sigma_K^{\frac{1}{2}} U_{B'}|| \tag{4.15}$$

As $U_\perp^*$ is the last $n-r$ columns of U_K, RHS above can be manipulated to get

$$\begin{aligned} U_\perp^{*T} U_K \Sigma_K^{\frac{1}{2}} U_{B'} &= [\mathbf{0}\ I_{n-r}] \begin{pmatrix} \Sigma_K^{(r)\frac{1}{2}} & \mathbf{0} \\ \mathbf{0} & \Sigma_K^{(n-r)\frac{1}{2}} \end{pmatrix} U_{B'} \\ &= \Sigma_K^{(n-r)\frac{1}{2}} U_{B'(n-r)} \end{aligned} \tag{4.16}$$

Using equation 4.15 and 4.24 with inequality 4.4 and definition of distance between subspaces, we get the following result:

$$\sqrt{\frac{\sigma_n(K)}{\sigma_1(\hat{K})}}||U_{B'(n-r)}|| \leq \sin\Theta \leq \sqrt{\frac{\sigma_{r+1}(K)}{\sigma_r(\hat{K})}}||U_{B'(n-r)}|| \tag{4.17}$$

Next, we argue that $||U_{B'(n-r)}|| \leq 1$. As $\mathcal{C}(B') \subseteq \mathcal{C}(X)$, using proposition 8.5 from [38]

$$||P_{B'} M|| \leq ||P_X M|| = ||M||,\ \forall M \tag{4.18}$$

where the last equality follows from equation 4.7. Consider $M = \begin{pmatrix} 0 \\ I_{n-r} \end{pmatrix}$,

$$||P_{B'}M|| = ||U_{B'}U_{B'}^T \begin{pmatrix} 0 \\ I_{n-r} \end{pmatrix}|| = ||U_{B'}^T \begin{pmatrix} 0 \\ I_{n-r} \end{pmatrix}|| = ||U_{B'(n-r)}^T|| = ||U_{B'(n-r)}||$$
$$\leq || \begin{pmatrix} 0 \\ I_{n-r} \end{pmatrix}|| = 1 \tag{4.19}$$

The second equality follows from the unitary invariance of the spectral norm, the fourth equality follows from equation 4.5 and the last inequality follows from equation 4.18. This completes the proof. □

As a consequence of the above theorem, if the r^{th} eigenvalue of K and $\hat{K}$ are close and the ratio of consecutive eigenvalues $\frac{\sigma_r(K)}{\sigma_{r+1}(K)}$ is large then the Nystrom approximation is "good". This is reminiscent of the bounds derived by Yang et al. in [39]. They show that for a large eigengap $(\sigma_r - \sigma_{r+1})$, spectral error can be bounded with high probability as

$$||K - \hat{K}|| \leq ||K - K_r|| + O\left(\frac{n}{r}\right) \tag{4.20}$$

Indeed, by assuming large eigengap in the spectrum of K and using theorem 6 from Yang etal. [40], we get the following result:

Corollary 1. *Let* $\Delta = \frac{\sigma_r(K)}{\sigma_{r+1}(K)}$ *be the ratio of consecutive eigenvalues of* K*. If*

$$\sigma_r(K) - \sigma_{r+1}(K) \geq \frac{12\ln(2/\delta)}{\sqrt{r}}$$

then with probability $1 - \delta$

$$dist(U^*, U_{\hat{K}}) \leq \sqrt{\frac{3}{1 + 2\Delta}} \tag{4.21}$$

Proof. By theorem 6 (Yang et al. [40]), we have for $\sigma_r(K) - \sigma_{r+1}(K) \geq \frac{12\ln(2/\delta)}{\sqrt{r}}$ with probability $1 - \delta$,

$$\sigma_r(\hat{K}) \geq \frac{2}{3}\sigma_r(K) + \frac{1}{3}\sigma_{r+1}(K) \tag{4.22}$$

Substituting in lemma 2 we get the stated result. □

Next, we show the incoherence of the initial iterate $\hat{U}^{(0)}$. We assume U^* to be μ-incoherent. We show that coherence of $U_{\hat{K}}$ is $C\mu$, where C depends on the condition number of the kernel matrix.

Lemma 3. *Suppose U^* is incoherent with parameter μ and $U_{\hat{K}}$ is the orthonormal basis of the rank r Nystrom approximation $\hat{K}$, then $\hat{K}$ is incoherent with parameter $\sqrt{\frac{\sigma_1(K)}{\sigma_r(\hat{K})}}\mu$. Further, if*

$$\sigma_r(K) - \sigma_{r+1}(K) \geq \frac{12\ln(2/\delta)}{\sqrt{r}}$$

then with probability greater than $1-\delta$, $\hat{K}$ is incoherent with parameter $\sqrt{1.5\mathcal{K}}\mu$ where $\mathcal{K} = \frac{\sigma_1(K)}{\sigma_r(K)}$ is the condition number of best rank r approximation of K.

Proof. Using equation 4.10 and definition of SVD, we get

$$U_{\hat{K}}\Sigma_{\hat{K}}U_{\hat{K}}^T = U_K\Sigma_K^{\frac{1}{2}}U_{B'}U_{B'}^T\Sigma_K^{\frac{1}{2}}U_K^T \tag{4.23}$$

Multiplying by e_i^T on both sides, taking norm and using $||A^TA|| = ||A||^2$

$$\begin{aligned} ||e_i^TU_{\hat{K}}\Sigma_{\hat{K}}^{\frac{1}{2}}|| &= ||e_i^TU_K\Sigma_K^{\frac{1}{2}}U_{B'}|| && (4.24)\\ &\leq ||e_i^TU_K\Sigma_K^{\frac{1}{2}}||||U_{B'}|| \\ &\leq \sqrt{\sigma_1(K)}||e_i^TU_K|| && (4.25)\end{aligned}$$

where last inequality follows from equation 4.4 and the fact that orthonormal matrices have unit norm. Also,

$$\begin{aligned} \sqrt{\sigma_r(\hat{K})}||e_i^TU_{\hat{K}}|| \leq ||e_i^TU_{\hat{K}}\Sigma_{\hat{K}}^{\frac{1}{2}}|| \leq \sqrt{\sigma_1(K)}||e_i^TU_K|| && (4.26)\\ \Rightarrow\ ||e_i^TU_{\hat{K}}|| \leq \sqrt{\frac{\sigma_1(K)}{\sigma_r(\hat{K})}}||e_i^TU_K|| \\ \mu_r(\hat{K}) \leq \sqrt{\frac{\sigma_1(K)}{\sigma_r(\hat{K})}}\mu(K) && (4.27)\end{aligned}$$

This completes the first part of the proof. The second part follows directly from the first part and equation 4.22. □

4.3 Iterations

The inductive hypothesis is the coherence of $\hat{U}^{(t)}$ and closeness of $\hat{U}^{(t)}$ to U^*, in step 10 of algorithm 2. We show that each step of the iteration geometrically reduces the distance to the optimal subspace while preserving the coherence of the iterate. This result is borrowed directly from theorem 5.1 of Jain et al. [36]. Note that, like Jain et al., we also assume uniform sampling through out the proof.

Theorem 1 (Theorem 2.5 and 5.1 from [36])**.** *In the sampling step, let every entry of K be sampled uniformly and independently with probability,*

$$p \geq C\frac{\left(\frac{\sigma_1(K)}{\sigma_r(K)}\right)^2\mu^2r^{2.5}\log n\log\frac{r||K||_F}{\varepsilon}}{n\delta_{2r}^2} \tag{4.28}$$

where $\delta_{2r} \leq \frac{\sigma_r(K)}{12r\sigma_1(K)}$ *and* $C \geq 0$ *is a global constant. The* $(t+1)^{th}$ *iterate* $\hat{V}^{t+1}$ *satisfies the following with high probability:*

$$dist\left(\hat{V}^{(t+1)}, U^*\right) \leq \frac{1}{4} dist\left(\hat{U}^{(t)}, U^*\right) \tag{4.29}$$

As a corollary to the above theorem, we show that step 11 of algortihm 2, doesn't cause much change in the distance.

Corollary 2. *Under the conditions of theorem 1,*

$$dist\left(\hat{U}^{(t+1)}, U^*\right) \leq \frac{5}{8} dist\left(\hat{U}^{(t)}, U^*\right) \tag{4.30}$$

Proof. By step 11 of algorithm 2 we have,

$$\hat{U}^{(t+1)} = \frac{1}{2}\left(\hat{V}^{(t+1)} + \hat{U}^{(t)}\right)$$

Multiplying by $U_\perp^*$ and taking norm,

$$\begin{aligned} ||U_\perp^* \hat{U}^{(t+1)}|| &= \left|\left|\frac{1}{2}\left(U_\perp^* \hat{V}^{(t+1)} + U_\perp^* \hat{U}^{(t)}\right)\right|\right| \\ &\leq \frac{1}{2}\left(||\, U_\perp^* \hat{V}^{(t+1)}\,|| + ||\, U_\perp^* \hat{U}^{(t)}\,||\right) \\ &= \frac{1}{2}\left(dist\left(\hat{V}^{(t+1)}, U^*\right) + dist\left(\hat{U}^{(t)}, U^*\right)\right) \\ &\leq \frac{5}{8} dist\left(\hat{U}^{(t)}, U^*\right) \end{aligned} \tag{4.31}$$

where the last inequality follows from theorem 1 □

Finally, we show the incoherence of $\hat{U}^{(t+1)}$. The incoherence of $\hat{V}^{(t+1)}$ follows directly from lemma 5.5 from Jain et al. [36].

Lemma 4 (Lemma 5.5 from [36])**.** *Let* $\hat{U}^{(t)}$ *be* μ_1 *incoherent. Then with probability at least* $1 - \frac{1}{n^3}$*, iterate* $\hat{V}^{(t+1)}$ *is also* μ_1 *incoherent.*

Using lemma 4, we can show the incoherence of iterate $\hat{U}^{(t+1)}$.

Corollary 3. *Let* $\hat{U}^{(t)}$ *be* μ_1 *incoherent. Then with probability at least* $1 - \frac{1}{n^3}$*, iterate* $\hat{U}^{(t+1)}$ *is also* μ_1 *incoherent.*

Proof. By step 11 of algorithm 2 we have,

$$\hat{U}^{(t+1)} = \frac{1}{2}\left(\hat{V}^{(t+1)} + \hat{U}^{(t)}\right) \tag{4.32}$$

By lemma 4 and the definition of incoherence we have,

$$||e_i^T \hat{U}^{(t)}|| \leq \mu_1 \sqrt{\frac{r}{n}}, \ \forall i \in [n] \tag{4.33}$$

$$||e_i^T \hat{V}^{(t+1)}|| \leq \mu_1 \sqrt{\frac{r}{n}}, \ \forall i \in [n] \tag{4.34}$$

Using in equation 4.32,

$$\begin{aligned}\left|\left|e_i^T\hat{U}^{(t+1)}\right|\right| &= \frac{1}{2}\left|\left|\left(e_i^T\hat{V}^{(t+1)} + e_i^T\hat{U}^{(t)}\right)\right|\right| \\ &\leq \frac{1}{2}\left(\left|\left|e_i^T\hat{V}^{(t+1)}\right|\right| + \left|\left|e_i^T\hat{U}^{(t)}\right|\right|\right) \\ &\leq \mu_1\sqrt{\frac{r}{n}},\ \forall i \in [n] \end{aligned} \tag{4.35}$$

□

Chapter 5

Empirical Results

5.1 Exploring the Nature of the Gaussian Kernel Matrix

A kernel matrix can be high rank or low rank, coherent or incoherent depending on the kernel function. For shift invariant kernel functions, these properties are controlled by the scale parameter. For the Gaussian kernel function described in equation 5.1, these properties are controlled by the bandwidth parameter, σ. In this section, we will empirically show the effect of the bandwidth parameter on the nature of the Gaussian kernel matrix.

$$K(x, y) = \exp\left(-\frac{||x - y||^2}{2\sigma^2}\right) \tag{5.1}$$

5.1.1 Rank versus bandwidth parameter

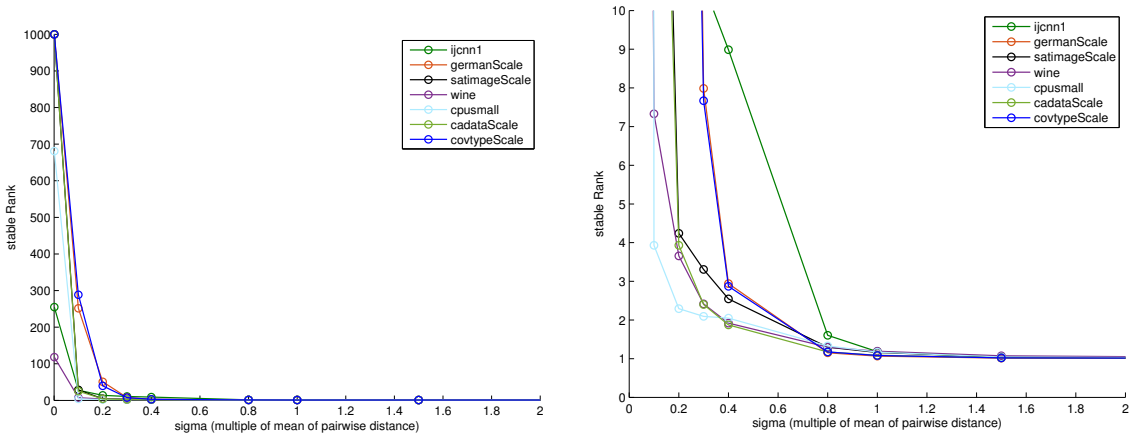
The nature of the Gaussian kernel matrix changes from high rank to low rank as the sigma parameter is increased from 0 to ∞. This behaviour can be observed by studying the variation of the stable rank of the kernel matrix with σ for several real life datasets . The stable rank of a matrix M is defined as:

$$R_s = \frac{||M||_F^2}{||M||^2} \tag{5.2}$$

$$= 1 + \frac{\sigma_2^2}{\sigma_1^2} + \ldots + \frac{\sigma_n^2}{\sigma_1^2} \tag{5.3}$$

where σ_i are the singular vectors. The stable rank of a matrix is a lower bound on the true rank of the matrix. It gives an idea of how quickly the spectrum decays relative to σ_1.

Figure 5.1 shows the variation of stable rank with the sigma parameter for several datasets. For uniformity, we consider the 'small' version of all the datasets which is composed by sampling 1000 observations uniformly from the dataset. Note in figure 5.1a that the stable rank falls rapidly from 1000 to 1 as the sigma parameter is increased. The quick descent to stable rank 1 indicates that as sigma is increased, the largest eigenvalue starts dominating the other eigenvalues. Figure 5.1b shows that for sigma based on the mean of pairwise distance between data (σ_{mean}), the kernel matrix has a low rank nature as stable rank approaches 1. This is important as we have noticed in our experiments that the tuned σ value for kernel ridge regression is close to σ_{mean} (see table 5.13). This kind of behaviour suggests the low rank nature of the kernel matrix at the tuned values and motivates the use of a low rank approximation.



(a) (b)

Figure 5.1: stable rank vs σ for different datasets. Left: Results for sigma based on 0 to 2 times of mean of pairwise distance. Right: Magnified view, results for sigma based on 0 to 2 times of mean of pairwise distance.

5.2 Kernel Approximation via ALS

5.2.1 Comparison of sampling schemes

In section 3.4, several sampling schemes were discussed. In this section we give an empirical comparison between these schemes. Table 5.1 to table 5.5 shows the comparison of different sampling schemes across datasets. The tables report the mean spectral error ($||K - \hat{K}||$) over 10 independent runs of ALS. Note that kNystrom is used as the initialization step here. The datasets we compare here are: 'ijcnn small', 'german scaled' and 'satimage scaled'. For a description of the datasets see appendix B. The best sampling scheme for each sigma parameter is highlighted in bold.

It is worthwhile to discuss the nature of the kernel matrix for extreme values of the sigma parameter. At the lower end of the spectrum, the kernel matrix is nearly an identity matrix. If we use a distribution similar to equation 3.3 for such matrices, the probability of sampling off-diagonal elements is very low and these are almost never sampled. This leads to the problem of not being able to sample the requisite number of samples. A work around is to raise the value of m, leading to a rise in the probability of sampling off-diagonal elements. However, such a method gives very high weights to the off-diagonal terms, leading to high errors. At the other end of the spectrum are the kernel matrices for high values of sigma. For very high sigma values, the kernel matrix is a rank 1 matrix with all entries as 1. For such matrices, $L1$, $L2$ or uniform sampling will behave in the same way as can also be seen from the simulations.

Table 5.1 shows the errors for 'ijcnn1 small' dataset for a rank 50 approximation.

Approximately $49n \log n$ samples were taken for each run. The uniform sampling scheme

sigma	L1	L2	L2 approx	Uniform	Clustering	$U + C$	$U + D$	$U + C + D$
0.212	**5.871**	28780	27152	13.96	949.6	13.13	251.2	20.87
0.424	12121	27.25	29.35	**5.218**	9.942	5.271	5.895	6.241
0.848	4.898	2.205	2.107	**2.030**	3.900	2.314	2.085	2.212
1.061	1.449	1.226	1.238	**1.171**	2.121	1.368	1.209	1.340
1.591	0.362	0.355	**0.354**	0.363	0.631	0.398	0.357	0.392
3.182	0.030	0.030	0.030	**0.030**	0.050	0.033	0.031	0.033
5.303	0.004	0.004	0.004	**0.004**	0.007	0.005	0.004	0.004

Table 5.1: Comparison of different sampling schemes across sigma parameter for 'ijcnn1 small' dataset rank 50 approximation. Sample size $\approx 49n \log n$

performs better than any other sampling scheme for most of the sigma values. Note that the performance of $L2$ and $L2$ approximation are almost similar, indicating that the approximation using Fourier features is not bad. Table 5.2 shows the errors for 'ijcnn1 small' dataset for a rank 200 approximation. Here as well, the performance of $L2$ and $L2$ approximation is similar. Interestingly, $U+C+D$ performs better than any other sampling scheme for most of the moderately low values of sigma parameter.

sigma	L1	L2	L2 approx	Uniform	Clustering	$U + C$	$U + D$	$U + C + D$
0.212	**1.603**	1477	141.4	3.979	6.635	1.766	3.087	2.423
0.424	21.62	1.332	1.417	0.980	1.197	0.876	0.959	**0.870**
0.848	0.362	0.219	0.214	0.181	0.183	0.161	0.164	**0.155**
1.061	0.118	0.094	0.097	0.078	0.0761	0.072	0.075	**0.069**
1.591	0.014	0.016	0.016	0.014	0.0134	0.014	0.013	**0.013**
3.182	5.9e-04	6.1e-04	5.8e-04	5.8e-04	5.8e-04	5.7e-04	5.8e-04	**5.6e-04**
5.303	6.8e-05	6.9e-05	7.0e-05	6.8e-05	7.0e-05	6.9e-05	7.0e-05	**6.8e-05**

Table 5.2: Comparison of different sampling schemes across sigma parameter for 'ijcnn1 small' dataset, rank 200 approximation. Sample size $\approx 120n \log n$

Similar observations are made for the 'german scaled' dataset in table 5.3 and table 5.4. For low rank approximation (rank 50), uniform and $L2$ sampling out perform other sampling schemes. Uniform is either better or very close in error to the $L2$ sampling scheme. For higher rank approximation (rank 200), $U + C$ or $U + C + D$ does better than uniform. For 'satimge scaled' dataset, table 5.5 reports the errors for rank 100 approximation. For this dataset $U + C + D$ performs better of close to the the best scheme.

Based on these observations, we choose $U + C + D$ as the preferred sampling scheme when the requisite rank approximation is greater than 50. For rank approximation less than 50, sampling is done using the 'uniform' distribution.

sigma	L1	L2	L2 approx	Uniform	Clustering	$U + C$	$U + D$	$U + C + D$
0.649	18951	795	339.4	303.0	7476	**139.1**	1676	1844
1.299	6389	5.721	4.912	**3.668**	6.140	4.262	4.924	9.485
2.598	6.613	**3.067**	3.140	3.071	3.531	3.131	3.098	3.324
3.247	2.388	2.114	**2.065**	2.094	2.376	2.170	2.114	2.163
4.871	0.725	0.719	0.714	**0.712**	0.798	0.739	0.717	0.732
9.742	0.064	**0.063**	0.063	0.063	0.072	0.066	0.065	0.065
16.236	0.009	0.010	**0.009**	0.009	0.010	0.009	0.009	0.009

Table 5.3: Comparison of different sampling schemes across sigma parameter for 'german scaled' dataset, rank 50 approximation. Sample size $\approx 49n \log n$

sigma	L1	L2	L2 approx	Uniform	Clustering	$U + C$	$U + D$	$U + C + D$
0.649	1373	4914	276.8	137.1	**102.7**	184.6	230.2	502.2
1.299	3.857	5.164	2.762	1.953	1.687	**1.626**	1.841	2.143
2.598	1.049	0.754	0.771	0.635	0.591	**0.566**	0.568	0.569
3.247	0.439	0.380	0.394	0.302	0.289	0.288	0.299	**0.282**
4.871	0.067	0.073	0.079	0.067	0.065	0.063	0.064	**0.061**
9.742	4.1e-03	4.2e-03	4.3e-03	4.1e-03	4.1e-03	4.1e-03	4.1e-03	**4.0e-03**
16.24	5.3e-04	5.3e-04	5.7e-04	5.4e-04	5.4e-04	5.3e-03	**5.2e-03**	5.3e-03

Table 5.4: Comparison of different sampling schemes across sigma parameter for 'german scale' dataset, rank 200 approximation. Sample size $\approx 120n \log n$

5.2.2 Comparison of initialization strategies

Several initialization strategies were discussed in section 3.5. In this section, the empirical error for different strategies is reported. For comparison, we run 10 independent runs of ALS and report the mean spectral error. For each independent run 3 iterations of ALS are performed. Using the observations from the previous section, we choose the sampling distribution as $U + C + D$ for $r \geq 50$ and uniform sampling otherwise.

Table 5.6 shows the performance of different initialization schemes across a range of sigma parameters for 'ijcnn1 small' dataset. K-means Nystrom performs better than any other initialization scheme across the range of sigma values except for small sigma values.

sigma	L1	L2	L2 approx	Uniform	Clustering	$U+C$	$U+D$	$U+C+D$
0.465	4434	92.05	10.70	**2.172**	7.592	2.198	2.227	2.852
0.929	204.68	2.237	2.267	1.548	2.388	1.596	**1.474**	1.552
1.859	2.167	0.632	0.605	0.412	0.391	0.392	**0.346**	0.358
2.324	0.820	0.280	0.302	0.213	0.209	0.207	0.195	**0.189**
3.485	0.064	0.062	0.064	0.057	0.053	0.056	0.050	**0.048**
6.972	3.7e-03	4.0e-03	4.0e-03	4.1e-03	3.5e-03	4.0e-03	**3.5e-03**	3.7e-03
11.620	5.6e-04	5.4e-04	5.5e-04	5.5e-04	5.08e-04	4.9e-04	**4.4e-04**	4.9e-04

Table 5.5: Comparison of different sampling schemes across sigma parameter for 'satimage scale' dataset, rank 100 approximation. Sample size $\approx 151n \log n$

Sigma	Random	Nystrom	KNystrom	SVD
0.21	193.00	23.71	20.14	**6.04**
0.42	187.08	5.06	**4.96**	6.70
0.85	147.98	2.14	**2.01**	5.04
1.06	124.97	1.23	**1.22**	5.34
1.59	89.55	0.37	**0.36**	5.55
3.18	67.70	0.03	**0.03**	6.88
5.30	65.48	0.004	**0.004**	4.01

Table 5.6: Performance of different sampling strategies across sigma parameter for 'ijcnn1 Small'

Note that for small sigma, sparse SVD is better or comparable to Nystrom based methods. One reason for such behaviour is the lack of performance of sampling based methods for coherent matrices. For Nystrom methods this behaviour has been studied earlier by Talwalkar et al. [22]. Table 5.7 shows similar results for 'german scale' dataset. Table 5.8 shows the comparison of initialization strategies for different datasets. Sigma parameter in table 5.8 is chosen as the mean of pairwise distance between the data points (equation 5.4).This is a common technique used for Gaussian kernels [41, 42].

$$\sigma_{mean} = \sqrt{\frac{1}{2}\sum_{i,j} ||x_i - x_j||^2} \tag{5.4}$$

K-means Nystrom performs better on all the data sets. Performance of Nystrom is close to k-means Nystrom and is better than sparse SVD and random initialization. Sparse SVD, though better than random is much worse than Nystrom based approximations. Moreover, empirically the time taken by sparse SVD is much more than the time taken by Nystrom based methods.

Sigma	Random	Nystrom	KNystrom	SVD
0.65	**195.70**	1027.88	439.70	1390.21
1.30	192.18	3.94	**3.71**	5.11
2.60	166.28	3.18	**3.01**	6.56
3.25	139.94	2.20	**2.05**	6.16
4.87	96.45	0.75	**0.72**	6.99
9.74	67.98	0.068	**0.063**	5.93
16.24	64.94	0.010	**0.010**	4.19

Table 5.7: Performance of different sampling strategies across sigma parameter for 'german Scale'

Dataset	n	r	$\|\Omega\|$	Random	Nystrom	KNystrom	SVD
ijcnn1 Small	1000	50	$49n \log n$	125.95	1.23	**1.21**	5.38
german Scale	1000	50	$49n \log n$	141.26	2.16	**2.09**	6.45
satimage Scale	4435	100	$150n \log n$	348.81	0.21	**0.19**	2.20
wine	6497	100	$160n \log n$	322.09	1.06	**0.20**	5.12
cpusmall	8192	100	$163n \log n$	214.79	1.16e-05	**2.78e-06**	3.54

Table 5.8: Performance of different sampling strategies

5.2.3 Performance of ALS for kernel approximation

In this section we show comparisons of our method with other state of the art techniques for low rank approximation of kernel matrix. The algorithms that we compare include standard Nystrom, k-means Nystrom, MEKA, random Fourier features (RFF) and ALS. In past research, K-means Nystrom has shown better performance than other variations of Nystrom [30, 17] and therefore we do not consider other variations of Nystrom here. We use our own implementation of Nystrom, k-means Nystrom and RFF. For MEKA we use code provided by the original authors at `http://www.cs.utexas.edu/~ssi/meka/`. In the experiments done by Si et al. in [30], the rank approximation of each block is fixed to be constant. We use the same strategy here. Note that, in the code provided by the original authors, rank approximation of each block is computed adaptively in proportion to the number of points in the cluster. We also evaluated this strategy, but found the former to yield better results. The results for the adaptive strategy can be found in appendix **??**. Moreover, the original authors fixed the number of iterations of k-means clustering in MEKA to 10. We perform at most 500 iterations of kmeans to ensure convergence. We also compare the performance of all the algorithms against SVD's optimal errors.

5.2.3.1 Spectral error at fixed number of output parameters

Table 5.10 shows the comparison of performance of various algorithms. For each dataset the spectral error is computed over a randomly sampled subset of 1000 data points. For each dataset, the number of parameters required to store the sketch are kept the same across the algorithms. As it is difficult to match the exact same number of parameters for MEKA and other methods (see table 3.1), we choose MEKA's parameters such that the number of parameters is slightly greater than that used by the other algorithms. Bandwidth parameter is selected by either using equation 5.4 or by tuning the sigma parameter for kernel ridge regression problem. Results of tuning sigma using kernel ridge regression are provided in table 5.13.

Now we provide details of the parameters used for different algorithms:

- For standard Nystrom and k-means Nystrom, the number of columns sampled is $4r$.
- For ALS, we take $O(rn \log n)$ samples. Table 5.9 shows the details of the number of samples for ALS. Number of iterations of ALS is kept fixed at 3 for all datasets except for 'cadata', for which 4 iterations are run.
- For RFF, exactly r features are sampled for a rank r approximation.

We run 10 independent trials of each algorithm and report the mean and the standard deviation of the spectral error.

From table 5.10, it is clear that ALS leads to lower error than other state of the art algorithms. As the number of parameters used to store the sketch are the same for all the

Dataset	n	σ_{RBF}	r	$\lvert\Omega\rvert$
ijcnn1 Small	1000	1.06	50	$49n \log n$
german Scale	1000	3.25	50	$49n \log n$
satimage Scale	4435	2.32	100	$102n \log n$
wine	6497	32	100	$160n \log n$
cpusmall	8192	2×10^6	100	$163n \log n$
cadata Scale	20400	4	200	$430n \log n$
ijcnn1Scale	49990	4.70	200	$387n \log n$

Table 5.9: Number of samples taken from different datasets for each run of ALS. Table 5.10 shows the performance for these input parameters

Dataset	$\#param$	stdNys	kNys	MEKA	RRF	ALS	SVD
ijcnn Small	50000	2.28 ± 0.29	1.79 ± 0.20	2.06 ± 0.70	81.89 ± 0.34	$\mathbf{1.21 \pm 0.08}$	0.9178
german Scale	50000	3.36 ± 0.29	2.92 ± 0.11	7.00 ± 2.64	103.39 ± 33.17	$\mathbf{2.09 \pm 0.07}$	1.67
satimage Scale	443500	0.72 ± 0.28	0.30 ± 0.04	1.73 ± 0.24	57.42 ± 11.53	$\mathbf{0.20 \pm 0.01}$	0.12
wine	649700	1.12 ± 0.23	0.26 ± 0.04	1.16 ± 0.79	65.40 ± 22.73	$\mathbf{0.17 \pm 0.08}$	0.004
cpusmall ($\times 10^{-6}$)	819200	191.6 ± 211.1	4.64 ± 2.05	875.0 ± 735.5	$62.51 \pm 38.69 \times 10^6$	$\mathbf{1.28 \pm 0.56}$	0.004
cadata	4128000	0.67 ± 0.30	0.60 ± 0.15	0.072 ± 0.02	45.65 ± 12.67	$\mathbf{0.067 \pm 0.04}$	0.0003
ijcnnScale	9998000	0.79 ± 0.07	0.57 ± 0.06	1.03 ± 0.45	37.90 ± 6.46	$\mathbf{0.38 \pm 0.02}$	0.1307

Table 5.10: Comparison of different algorithms for low rank approximation of kernel matrix

algorithms, ALS shows the best performance given the sketch size. The good performance of ALS can in part be attributed to the good initialization solution provided by k-means Nystrom. In fact, k-means Nystrom is the second best algorithm among the suite of algorithms that we consider. Recently, MEKA was shown to have better performance than any other kernel approximation method [30]. We use the parameters as advised by the original authors and also do some hand tuning but we find the performance of MEKA unstable as can be seen from the high standard deviation of errors. This behaviour has been reported elsewhere [32, 31]. Interestingly, the performance of MEKA is considerably better for larger datasets and for high rank approximation. RRF shows the worst performance among the methods we compare. This is expected, as the number of features sampled for rank r approximation are restricted to r. Usually, a very high number of random features are required to accurately approximate the kernel value.

From table 5.10, one may also observe that ALS is among the most stable methods, with very low standard deviation. It is more stable than the k-means Nystrom method which initializes its solutions. This indicates that the ALS optimization process is iteratively able to reduce the noise in the solution in some sense.

One may argue that the performance of standard Nystrom and k-means Nystrom may be improved by increasing the number of sampled columns. Table 5.11 shows the comparison of the mean spectral error of ALS and Nystrom for the same number of sampled entries. Though Nystrom performance improves on increasing the number of sampled rows and columns, ALS is still slightly better. Also, note that the initialization strategy for ALS utilizes the Nystrom approximation. Hence, the output of the Nystrom approximation, with a large number of sampled columns can be further used to improve the performance of ALS. Similarly, MEKA's performance can be improved by using ALS rather than Nystrom for approximating the diagonal blocks.

Dataset	$\lvert\Omega\rvert$	stdNys	kNys	ALS
ijcnn Small	$49n \log n$	1.47	1.41	**1.18**
german Scale	$49n \log n$	2.57	2.25	**2.12**
satimage Scale	$102n \log n$	0.41	0.22	**0.19**

Table 5.11: Comparison of ALS and Nystrom with same number of samples

5.2.4 Comparison of error with number of parameters

Figure 5.2 shows the variation of the error with the number of parameters required to store the solution. The number of parameters are controlled by the parameter r, the desired rank approximation. As the number of parameters increase the approximation error goes down for all the algorithms. This is expected as higher number of parameters mean a higher

rank approximation which leads to better approximation. This also shows the decaying nature of the spectrum of the kernel matrix.

As in the previous sections, the error here is the approximation over a smaller sub-sampled 1000×1000 kernel matrix. The small size of the sub-sampled matrix makes the computation of SVD feasible. Note that for lower number of parameters MEKA's performance surpasses that of SVD for ijcnn1 dataset. SVD's results are optimal given a fixed rank approximation. As MEKA does a block-wise rank approximation, it is able to perform a higher rank approximation than SVD for the same number of output parameters. This is a major plus for MEKA. However, in our experiments, the performance of MEKA becomes worse than that of SVD and ALS on increasing the number of parameters.

The performance of ALS is worse only compared to SVD, which is provably optimal. Interestingly, the nature of variation of error with number of parameters is very similar for ALS and SVD. This indicates that ALS is able to learn the decaying spectrum of the kernel matrix.

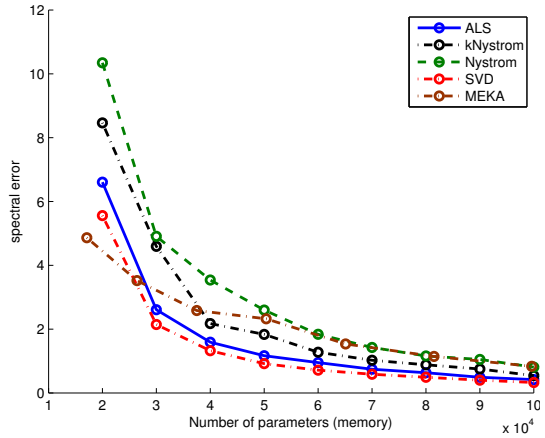
5.2.5 Run time comparison

Table 5.12 shows the average time taken (in sec) to get the approximation errors in table 5.10. We use a MATLAB based implementation of all the algorithms. All the experiments are run on intel core i7-4770 $3.4Ghz \times 4$ CPU with 31.1 GB memory. For each algorithm total time taken is computed as the time taken for the algorithm to return an approximation of the form $K \approx UU^T$.

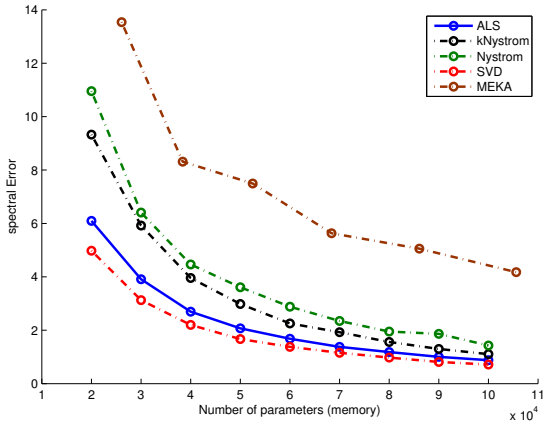
Clearly, ALS is the slowest among the compared algorithms and RRF is the fastest. Though the time complexity of ALS is almost linear in n, run time is a bottleneck for moderately high n. The slowness of ALS can be attributed to overheads in sampling and running the iterative least square solver.

Dataset	stdNys	kNys	MEKA	RRF	ALS
ijcnn Small	0.01	0.18	0.06	0.002	3.99
german Scale	0.01	0.23	0.04	0.002	3.47
satimage Scale	0.07	5.40	0.40	0.007	37.89
wine	0.08	2.70	0.23	0.009	81.07
cpusmall	0.08	5.68	0.39	0.010	42.84
cadata	0.38	17.07	1.74	0.05	3747

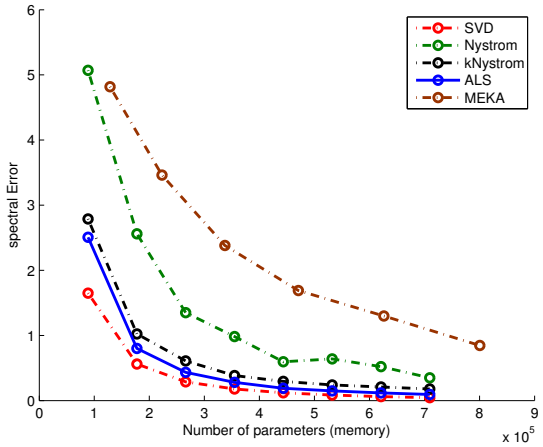
Table 5.12: Comparison of approximation time (in sec) using ALS algorithm for low rank approximation of kernel matrix



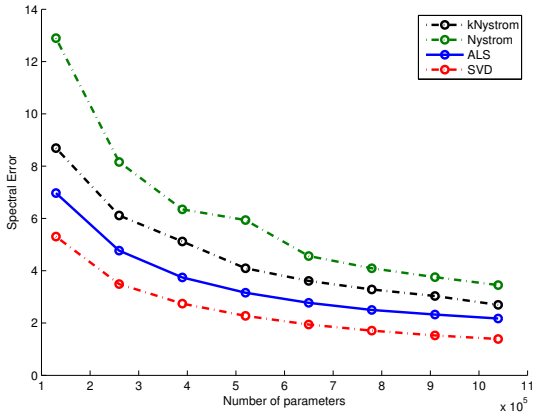
(a) ijcnn1Small: memory vs error



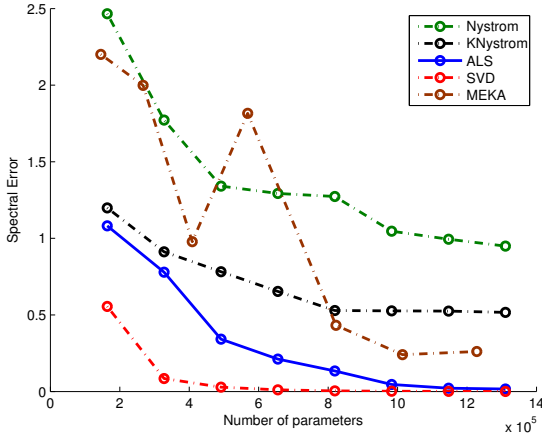
(b) german scale: memory vs error



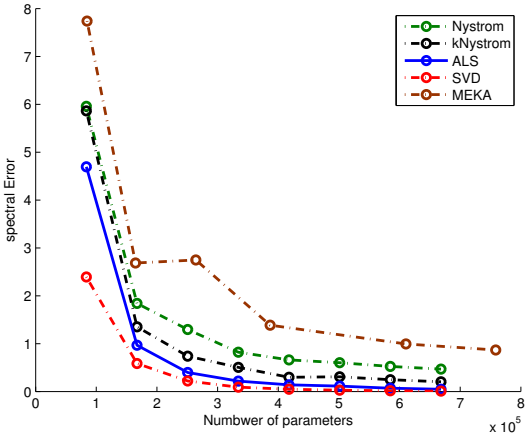
(c) satimage scale: memory vs error



(d) wineScale: memory vs error

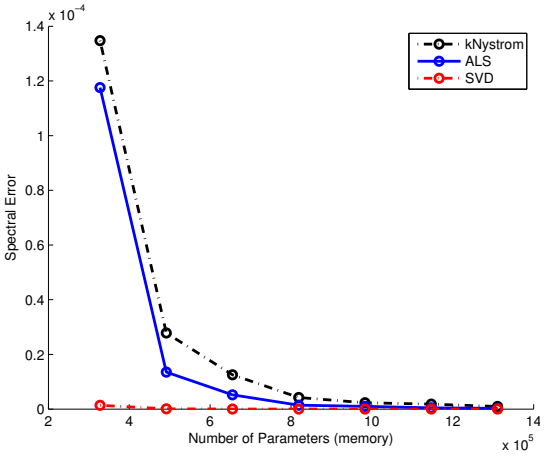


(e) cpusmallScale: memory vs error

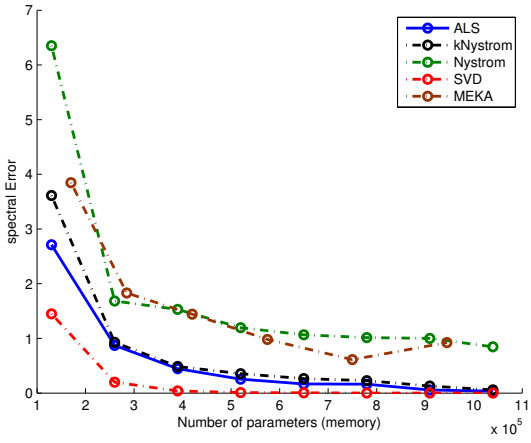


(f) abaloneScale: memory vs error

Figure 5.2: Number of parameters (Memory) vs Error for different datasets. Algorithms that have error greater than the maximum value on the given y-axis are not shown



(g) cpusmall: memory vs error



(h) wine: memory vs error

Figure 5.2: Number of parameters (Memory) vs Error for different datasets. Algorithms that have error greater than the maximum value on the given y-axis are not shown

5.3 Kernel Ridge Regression using Approximated Kernel

Kernel ridge regression (KRR) [3] performs linear regression in the mapped feature space. Let $\mathcal{L} = \{(x_1, y_1), \ldots, (x_L, y_L)\}$ be the learning set. Let $\phi : \mathcal{X} \to \mathcal{H}$ be the feature map from the input space to the feature space. In the mapped feature space the objective function to be minimized is:

$$\min_{w} \quad \lambda||w||^2 + \sum_{i=1}^{L}(w^T\phi(x_i) - y_i)^2 \tag{5.5}$$

where L is the size of learning set. The solution of above problem is given by

$$w = \frac{1}{2\lambda}\sum_{i=1}^{L}\alpha_i x_i$$
$$\text{where } \alpha = 2\lambda\left(K + \lambda I\right)^{-1} y \tag{5.6}$$

where $y = [y_1, \ldots, y_L]^T$, $\alpha = [\alpha_1, \ldots, \alpha_L]^T$ and K is the $L \times L$ kernel matrix corresponding to the learning set. For testing:

$$y_{predicted} = \frac{1}{2\lambda}\sum_{i=1}^{L}\alpha_i K\left(x_i, x_{test}\right) \tag{5.7}$$

The training step of KRR can be accelerated using a kernel approximation. The training step entails finding an inverse of the $n \times n$ matrix. Using an approximation of the form $K \approx UU^T$, this inversion can be computed by using the famous Woodbury formula [43, 11].

$$\lambda(K + \lambda I)^{-1} = I - U(\lambda I + U^T U)^{-1}U^T \tag{5.8}$$

Thanks to equation 5.8, the above inverse can be computed in $O(nr^2)$ time.

Root mean square error (RMSE) for cross validation is computed by evaluating mean square error for each fold and then taking the square root of its mean across the folds i.e. RMSE over the k-folds is

$$RMSE_{CV} = \sqrt{\frac{\sum_{j=1}^{k} \frac{\sum_{i=1}^{V_j}(y_i - \hat{y}_i)^2}{V_j}}{k}}$$

where V_j is the number of elements in the held out part of the j^{th} fold.

For the purpose of experiments all the datasets are split into 80-20 train-test ratio uniformly randomly. Tuning is performed on the training set. Parameters to be tuned include the bandwidth parameter of the kernel matrix and the regularizer of the regression problem. Tuning is done by performing a grid search over the parameter space to minimize the 4-fold cross validation error. The reported errors are 10-fold cross validation errors at the tuned value. Exact kernel computation is done for tuning. Table 5.13 shows the tuned parameters. σ_{mean} denotes the bandwidth parameter computed using the mean of pairwise distances between the data points. σ_{tune} denotes the bandwidth parameter after tuning. For the experiments we consider some datasets which are scaled to zero mean and unit variance and some unscaled datasets. For details about the datasets, see appendix B

Dataset	$\#obs$	σ_{mean}	σ_{tune}	λ	$RMSE_{CV10}$
abaloneScale	4177	2.83	2	0.0625	0.6541
Wine	6,497	59	32	.0039	0.7109
wineScale	6,497	3.32	1	0.5	0.7535
cpusmall	8,192	5×10^5	2×10^6	1.5×10^{-5}	5.3923
cpusmallScale	8,192	3.47	8	2^{-10}	0.1686
cadataScale	20,640	2.83	2	0.0078	0.4664

Table 5.13: Table showing tuned parameters for different datasets

Figure 5.3 shows the comparison of different datasets for kernel regression problem. Each point is an average of 10 random runs of all the algorithms. 'Exact' refers to using the original kernel matrix without any kind of approximation. From the plots, it can be inferred:

- As the number of parameters increase, RMSE goes down. As higher number of parameters mean better approximation to the true kernel, it shows that better approximation leads to better training.
- At the same time we observe that for a given number of parameters, all the algorithms perform similar to each other. This suggests that small differences in kernel may not be important for regression. This becomes more clear in the test plots.

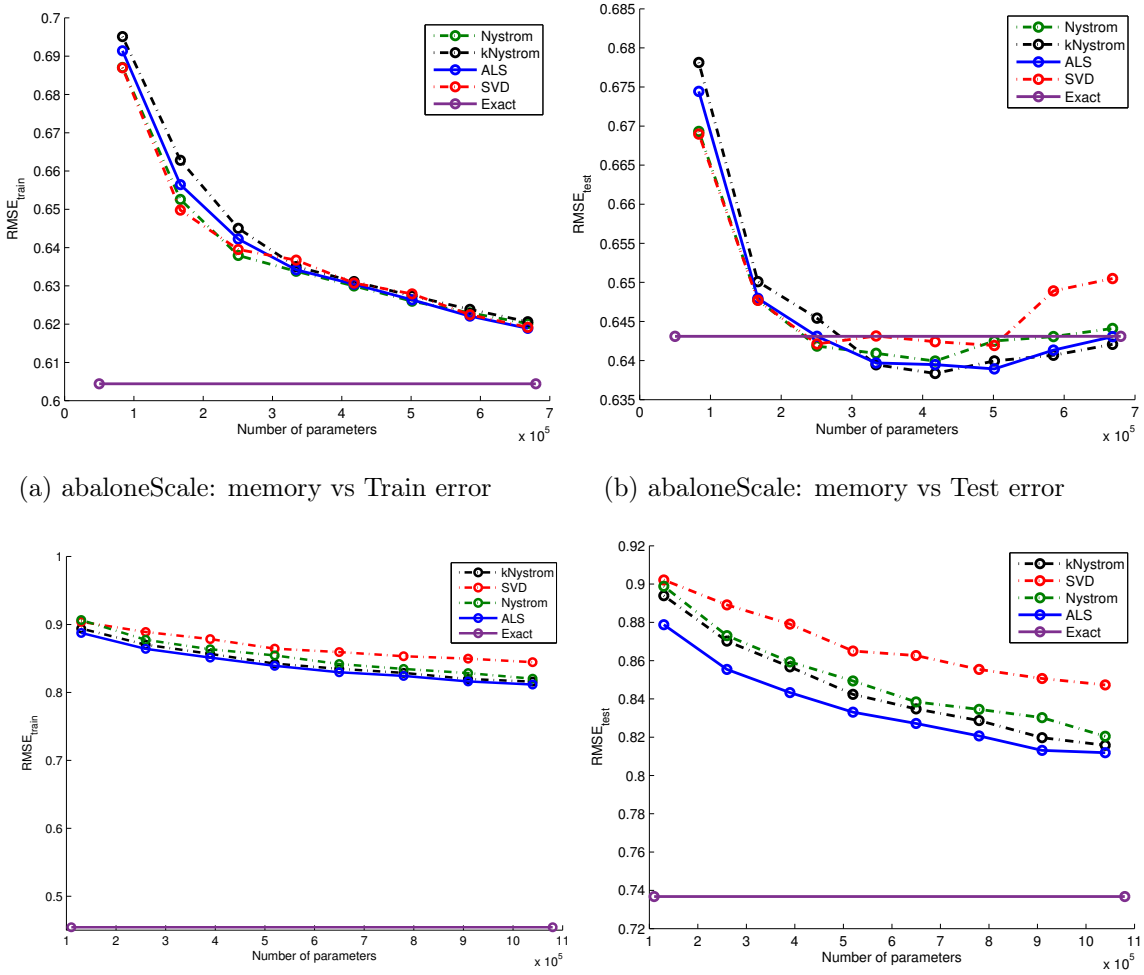
- In the test plots, for some datasets using an approximation may even be better than using the exact kernel. This may be attributed to low stable rank of these matrices as well. In figure 5.1b, we saw that the stable rank at which we are operating is very low. For a very low rank matrix, low number of parameters should be enough for approximation. In the test plots, the y-axis has a very small range, indicating that the difference between algorithms may not be statistically significant.

Note that, most of the plots in figure 5.3 do not show errors for MEKA. The reason is that MEKA has shown poor performance in our experiments and the mean errors go way higher than the shown y-axis. Though individual runs of MEKA may be competitive, we observe that the method is unstable and some runs may yield an ill-conditioned approximate leading to blowing up of α in equation 5.7. In the next section we present a comparison of condition number of approximates returned by different approximation schemes.

5.3.1 Condition number of approximate

For a rank r approximation, we measure the condition number of approximates as $\mathcal{K} = \frac{\sigma_1}{\sigma_r}$. Figure 5.4 shows the comparison of condition number for different rank approximations. All the entries are mean over 10 independent runs of each algorithm. Note that, for MEKA the rank approximation r is the cumulative of rank approximation of each diagonal block. Although we use the simple scheme where rank of each block is a fixed number, certain pathological cases may occur where the size of the cluster is less than the assigned rank approximation of that block. Thus, rank of each block depends on the clustering procedure, which begins with random initialization in each run. In figure 5.4, rank for MEKA is taken as the average over independent runs.

From figure 5.4, it is clear that the condition number of approximates rises with increase in the number of parameters. This is expected since the condition number rises with the decaying spectrum. Further, the condition number of ALS is consistently smaller than that of Nystrom and k-menas Nystrom. Also, the condition number for ALS is close to SVD's condition number, showing that spectrum of ALS approximate is similar to that of SVD. Interestingly, MEKA shows a very erratic behaviour in terms of condition number. Again, we observe the high variance behaviour for MEKA. Some of the independent runs yield a very high condition number, leading to high mean condition number. To illustrate the difference we show the condition number of MEKA with rank in figure 5.5 and the corresponding regression error in figure 5.6. We believe that the erratic behaviour of MEKA could be attributed to instability in k-means clustering which give different clusters in each random run.



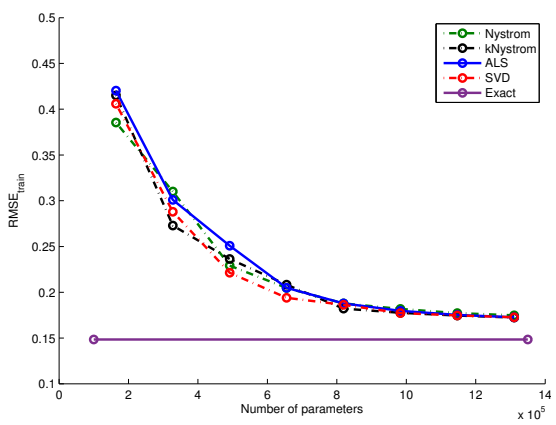
(a) abaloneScale: memory vs Train error

(b) abaloneScale: memory vs Test error

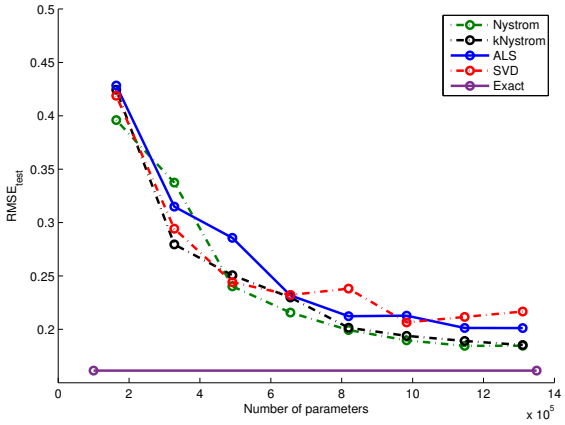
(c) wineScale: memory vs Train error

(d) wineScale: memory vs Test error

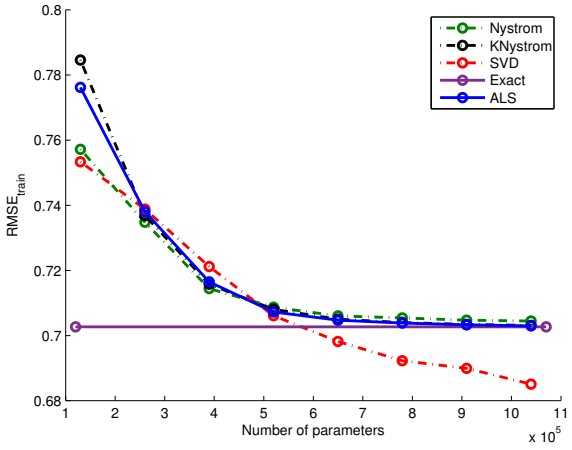
Figure 5.3: Number of parameters (Memory) vs Error for different datasets. Algorithms that have error greater than the maximum value on the given y-axis are not shown



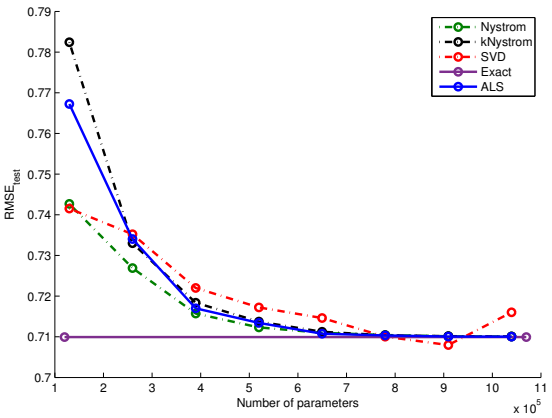
(e) cpusmallScale: memory vs Train error



(f) cpusmallScale: memory vs Test error

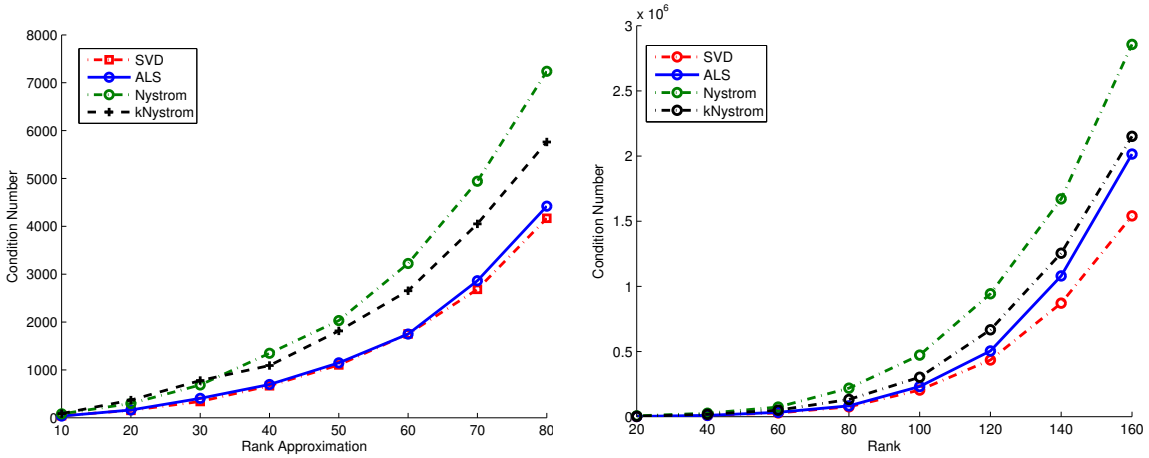


(g) wine: memory vs Train error

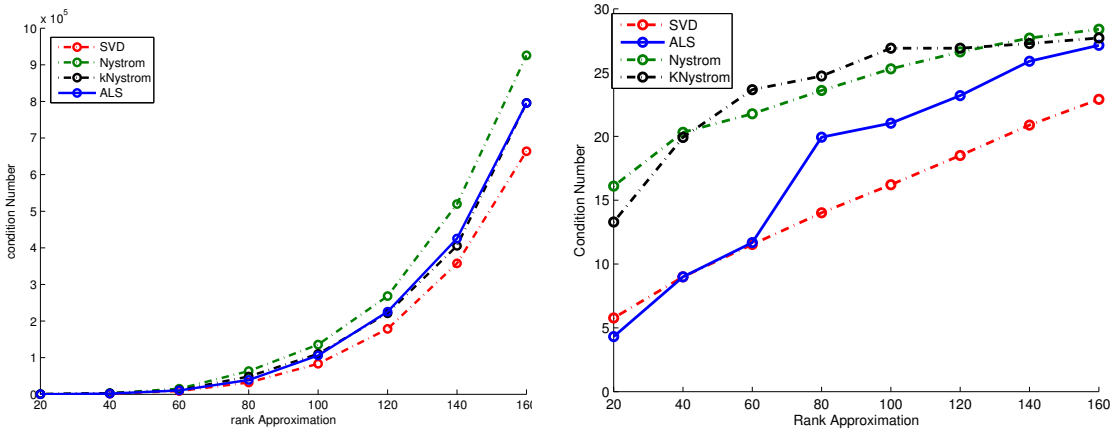


(h) wine: memory vs Test error

Figure 5.3: Number of parameters (Memory) vs Error for different datasets. Algorithms that have error greater than the maximum value on the given y-axis are not shown



(a) abaloneScale: Condition Number vs Rank (b) cpusmallScale: Condition Number vs Rank



(c) wine:Condition Number vs Rank (d) wineScale: Condition Number vs Rank

Figure 5.4: Condition number vs Rank for different datasets. Algorithms that have condition number greater than the maximum value on the given y-axis are not shown

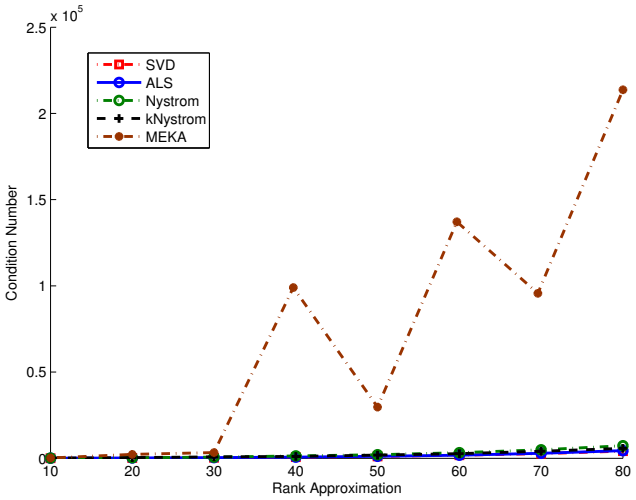
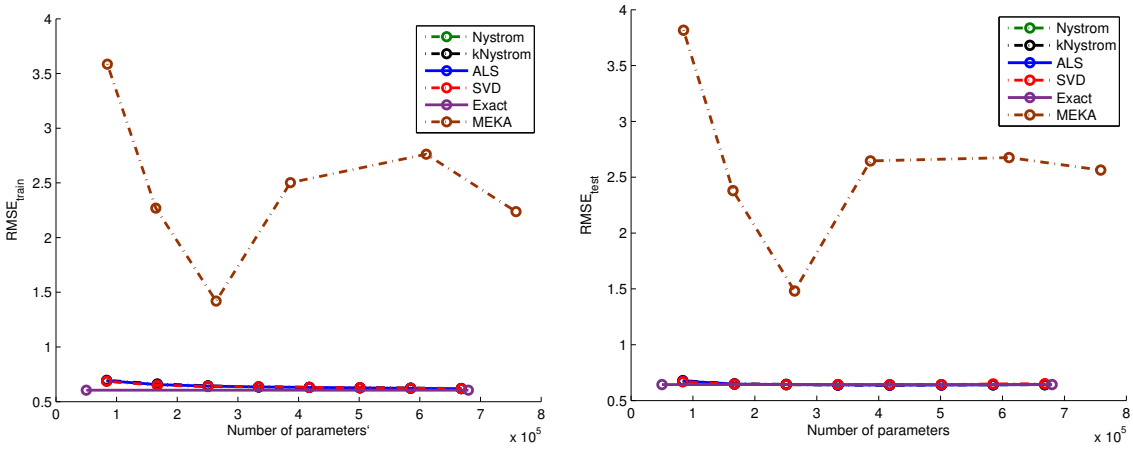


Figure 5.5: abaloneScale:Condition Number vs Rank



(a) abaloneScale:Train Error vs Number of parameters

(b) abaloneScale: Test Error vs Number of parameters

Figure 5.6: Train and Test Regression Error vs Number of parameters for different algorithms. The figure shows the difference between MEKA and other algorithms

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Kernel approximation is a popular and effective way of scaling up kernel machines. There has been a long line of work that has been done in this direction. Much work has also been done in the direction of general matrix low rank approximation. While these methods can be applied directly to kernel matrices, they do not generally respect the symmetry and positive semi-definite nature of the kernel matrix [19]. Leverage element low rank approximation (LELA) is a recent development for approximating general matrices. In this work we have extended the LELA algorithm to handle SPSD kernel matrices. The major bottlenecks in LELA, while dealing with dense SPSD matrices, included efficient sampling, good initialization and symmetric factorization. We make the following contributions in this regard:

- We propose an efficient sampling scheme and show empirical evidence of its superiority over other sampling schemes (section 3.4).
- We leverage the properties of SPSD matrices and propose an initialization scheme based on a variant of the Nystrom approximation method. We empirically show the advantages of using this initialization scheme over sparse SVD (section 3.5).
- We propose and implement a simple trick to construct a symmetric factorization of the kernel matrix. We show that previous theoretical results in the literature for non-symmetric factorization can be easily extended for the proposed symmetric factorization step as well (section 4.3).
- We provide theoretical analysis of a simplified version of our method and show its convergence (chapter 4).
- Our kernel approximation scheme based on ALS shows good empirical performance on several datasets when compared with other state-of-the-art algorithms in kernel approximation (section 3.6).
- We show a comparison of different low rank approximation schemes on kernel ridge regression problem (section 5.3).

6.2 Future work

The major drawback of our algorithm is its run time when compared to other state-of-the-art methods (Table 5.12). Our method runs faster than SVD but is substantially slower than other approximation schemes while providing a slightly better sketch. The major time consuming step in our algorithms is solving the least square problem iteratively. In the future we would like to explore different optimization algorithms which could give similar performance but run faster.

The current sampling is based on a mixture of uniform sampling and k-means clustering. Our theoretical analysis is based on the assumption of a simpler uniform sampling. In future we would like to explore more complicated but computationally efficient sampling schemes based on the properties of kernel matrix.

Lastly, our analysis of initialization using Nystrom approximation is based on the assumption of a large gap in the spectrum of the input matrix and is pretty loose. Indeed, we find that the bounds given in the literature on theoretical analysis of the Nystrom approximation are very loose when compared to its empirical performance. We intend to do a tighter analysis of the Nystrom method and in turn improve the bounds of our algorithm for kernel approximation.

Appendix A

Proof of lemma 1 is a simple extension of Theorem 3 in [24]. Before stating the proof we review some of the inequalities used in the proof.

Lemma 5. *(Matrix Bernstein inequality [25]) Let $X_1, \ldots, X_p$ be independent random matrices in $\mathbb{R}^{n\times n}$. Assume each matrix has bounded deviation from its mean:*

$$||X_i - E[X_i]|| \leq L, \forall i \in [p] \text{ with probability } 1 \tag{A.1}$$

Also, let variance be

$$\sigma^2 = \max\left\{\left|\left|E\left[\sum_{i=1}^{p}(X_i - E[X_i])(X_i - E[X_i])^T\right]\right|\right|, \left|\left|E\left[\sum_{i=1}^{p}(X_i - E[X_i])^T(X_i - E[X_i])\right]\right|\right|\right\}$$

Then,

$$P\left[\left|\left|\sum_{i=1}^{p}(X_i - E[X_i])\right|\right| \geq t\right] \leq 2n\exp\left(\frac{-t^2/2}{\sigma^2 + Lt/3}\right) \tag{A.2}$$

A partial ordering can be defined on symmetric matrices. $A \preceq B$ signifies that matrix $B - A$ is PSD. Also, if $A \preceq B$ then it follows that $||A|| \leq ||B||$.

Now, we state the proof of lemma 1

Proof. (Proof of lemma 1) Consider D random fourier features[23]

$$z_i(X) = \left[\sqrt{2}\cos(\omega_i x_1 + b_i), \ldots, \sqrt{2}\cos(\omega_i x_n + b_i)\right]^T, i = 1, \ldots D$$

Let $X_i = \frac{1}{D}\left(z_i z_i^T - K\right)$ be a random matrix. As $E\left[z_i z_i^T\right] = K$, $E[X_i] = 0$.

$$\begin{aligned} ||X_i|| &= \frac{1}{D}||z_i z_i^T - K|| \\ &\leq \frac{1}{D}\left(||z_i z_i^T|| + ||K||\right) \end{aligned} \tag{A.3}$$

$z_i z_i^T$ is a rank 1 matrix so, $z_i z_i^T = \frac{z_i}{||z_i||}||z_i||^2\frac{z_i^T}{||z_i^T||}$ and $||z_i z_i^T|| = ||z_i||^2 \leq 2n$. Also, $||K|| \leq tr(K) = n$. Using in equation A.3:

$$||X_i|| \leq \frac{3n}{D} \tag{A.4}$$

Variance $\sigma^2 = \max\{\left|\left|E\left[\sum_{i=1}^{D} X_i X_i^T\right]\right|\right|, \left|\left|E\left[\sum_{i=1}^{D} X_i^T X_i\right]\right|\right|\}$ As, X_i is symmetric here, this reduces to,

$$\begin{aligned}\sigma^2 &= \left|\left|E\left[\sum_{i=1}^{D} X_i^2\right]\right|\right| = \left|\left|\sum_{i=1}^{D} E\left[X_i^2\right]\right|\right| \\ &\leq \sum_{i=1}^{D} \left|\left|E\left[X_i^2\right]\right|\right| \end{aligned} \tag{A.5}$$

by linearity of expectation and triangle inequality. Further,

$$\begin{aligned} E\left[X_i^2\right] &= \frac{1}{D^2} E\left[||z_i||^2 z_i z_i^T - K z_i z_i^T - z_i z_i^T K + K^2\right] \\ &\preccurlyeq \frac{1}{D^2} E\left[2n z_i z_i^T\right] - K E\left[z_i z_i^T\right] - E\left[z_i z_i^T\right] K + K^2 \\ &= \frac{1}{D^2}\left(2nK - K^2\right) = \frac{1}{D^2} K\left(2nI - K\right) \preccurlyeq \frac{2nK}{D^2} \end{aligned} \tag{A.6}$$

$\left|\left|E\left[X_i^2\right]\right|\right| \leq \frac{2n||K||}{D^2} = \frac{2n^2}{D^2}$. Using this in equation A.5,

$$\sigma^2 \leq \frac{2n^2}{D}$$

Using above and equation A.4 in matrix bernstein inequality, we get the stated result.

□

Appendix B

Details of Datasets

B.1 Summary of Datasets used

Dataset	$\#obs$	$\#features$	σ_{mean}
abaloneScale	4,177	8	2.83
cadataScale	20,640	8	2.52×10^3
cpusmall	8,192	12	5×10^5
cpusmallScale	8,192	12	3.46
wine	6,497	11	59
wineScale	6,497	11	3.32
ijcnn1Small	1,000	22	1.06
germanScale	1,000	24	3.25
satimageScale	4,435	36	2.32

Table B.1: Summary of datasets used

B.2 Details of Datasets used

The scaled version of most of the datasets with test and train data points is available from project's website.

1. **abaloneScale**: Original source of abalone dataset is UCI. We download it from LIBSVM website. Original dataset is scaled to 0 mean and unit variance to create abaloneScale.

2. **cpusmall**: Original data available from Delve. We download it from LIBSVM website. **cpusmallScale** is the Scaled version of this dataset to 0 mean and unit variance.

3. **cadataScale**: Original source of cadata is statLib. We download it from LIBSVM website and scale it to 0 mean and unit variance.

4. **wine**: Original source is wine quality dataset from UCI. **wineScale** is the scaled version of this dataset with 0 mean and unit variance.

5. **ijcnn1Small** - ijcnn1 is the modified version of the data provided in a competition conducted as a part of International Joint Conference on Neural Networks 2001 (ijcnn1). The original data is time series data of $\approx 50,000$ observations of a 10 cylinder internal combustion engine. We take the pre processed data from LIBSVM website and uniformly randomly select $1,000$ observations.

6. **germanScale**: We take the scaled version of the german dataset from LIBSVM website. The data is scaled in $[-1, 1]$.

7. **satimageScale**: We take the scaled version of the satimage dataset from LIBSVM website.

Bibliography

[1] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992. (Cited on page 1.)

[2] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995. (Cited on page 1.)

[3] Craig Saunders, Alexander Gammerman, and Volodya Vovk. Ridge regression learning algorithm in dual variables. *(ICML-1998) Proceedings of the 15th International Conference on Machine Learning*, pages 515–521, 1998. (Cited on pages 1 and 44.)

[4] Bernhard Scholkopf, Alexander Smola, and Klaus Robert Muller. Kernel principal component analysis. In *Artificial Neural Networks-ICANN'97*, pages 583–588. Springer, 1997. (Cited on page 1.)

[5] Pei Ling Lai and Colin Fyfe. Kernel and nonlinear canonical correlation analysis. *International Journal of Neural Systems*, 10(05):365–377, 2000. (Cited on page 1.)

[6] Francis R Bach and Michael I Jordan. Kernel independent component analysis. *The Journal of Machine Learning Research*, 3:1–48, 2003. (Cited on page 1.)

[7] Stephen Tyree, Jacob R. Gardner, Kilian Q. Weinberger, Kunal Agrawal, and John Tran. Parallel support vector machines in practice. *arXiv preprint arXiv:1404.1066*, 2014. (Cited on page 3.)

[8] Po-Sen Huang, Haim Avron, Tara N Sainath, Vikas Sindhwani, and Bhuvana Ramabhadran. Kernel methods match deep neural networks on timit. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 205–209. IEEE, 2014. (Cited on page 3.)

[9] Zhiyun Lu, Avner May, Kuan Liu, Alireza Bagheri Garakani, Dong Guo, Aurélien Bellet, Linxi Fan, Michael Collins, Brian Kingsbury, Michael Picheny, et al. How to scale up kernel methods to be as good as deep neural nets. *arXiv preprint arXiv:1411.4000*, 2014. (Cited on page 3.)

[10] Christopher Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Proceedings of the 14th Annual Conference on Neural Information Processing Systems*, number EPFL-CONF-161322, pages 682–688, 2001. (Cited on pages 3, 8 and 10.)

[11] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996. (Cited on pages 7, 26 and 44.)

[12] J. Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 209(441-458):415–446, 1909. (Cited on page 8.)

[13] Bernhard Scholkopf. Support vector learning. 1997. (Cited on page 9.)

[14] E.J. Nyström. Uber die praktische auflosung von integralgleichungen mit anwendungen auf randwertaufgaben. *Acta Mathematica*, 54(1):185–204, 1930. (Cited on page 9.)

[15] Christopher Williams and Matthias Seeger. The effect of the input density distribution on kernel-based classifiers. In *Proceedings of the 17th International Conference on Machine Learning*, number EPFL-CONF-161323, pages 1159–1166, 2000. (Cited on page 11.)

[16] Petros Drineas and Michael W Mahoney. On the nyström method for approximating a gram matrix for improved kernel-based learning. *The Journal of Machine Learning Research*, 6:2153–2175, 2005. (Cited on page 11.)

[17] Kai Zhang, Ivor W Tsang, and James T Kwok. Improved nyström low-rank approximation and error analysis. In *Proceedings of the 25th international conference on Machine learning*, pages 1232–1239. ACM, 2008. (Cited on pages 11 and 39.)

[18] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. Ensemble nyström method. *Neural Information Processing Systems*, 2009. (Cited on page 11.)

[19] Alex Gittens and Michael W Mahoney. Revisiting the nyström method for improved large-scale machine learning. *arXiv preprint arXiv:1303.1849*, 2013. (Cited on pages 11 and 51.)

[20] Alex Gittens. The spectral norm error of the naive nyström extension. *arXiv preprint arXiv:1110.5305*, 2011. (Cited on page 11.)

[21] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. Sampling techniques for the nyström method. *Conference on Artificial Intelligence and Statistics*, pages 304–311, 2009. (Cited on page 11.)

[22] Ameet Talwalkar and Afshin Rostamizadeh. Matrix coherence and the nyström method. In *Conference on Uncertainty in Artificial Intelligence*, 2010. (Cited on pages 11 and 37.)

[23] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2007. (Cited on pages 12, 20 and 53.)

[24] David Lopez-Paz, Suvrit Sra, Alex Smola, Zoubin Ghahramani, and Bernhard Schölkopf. Randomized nonlinear component analysis. *arXiv preprint arXiv:1402.0119*, 2014. (Cited on pages 12 and 53.)

[25] Joel A Tropp. User-friendly tail bounds for sums of random matrices. *Foundations of Computational Mathematics*, 12(4):389–434, 2012. (Cited on pages 12 and 53.)

[26] Lester Mackey, Michael I Jordan, Richard Y Chen, Brendan Farrell, Joel A Tropp, et al. Matrix concentration inequalities via the method of exchangeable pairs. *The Annals of Probability*, 42(3):906–945, 2014. (Cited on page 12.)

[27] Andrea Vedaldi and Andrew Zisserman. Efficient additive kernels via explicit feature maps. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(3):480–492, 2012. (Cited on page 12.)

[28] Sreekanth Vempati, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Generalized rbf feature maps for efficient detection. In *BMVC*, pages 1–11, 2010. (Cited on page 12.)

[29] Purushottam Kar and Harish Karnick. Random feature maps for dot product kernels. *Arxiv preprint arXiv:1201.6530*, 2012. (Cited on page 12.)

[30] Si Si, Cho-Jui Hsieh, and Inderjit Dhillon. Memory efficient kernel approximation. In *Proceedings of The 31st International Conference on Machine Learning*, pages 701–709, 2014. (Cited on pages 13, 39 and 41.)

[31] Shusen Wang, Luo Luo, and Zhihua Zhang. The modified nyström method: Theories, algorithms, and extension. *arXiv preprint arXiv:1406.5675*, 2014. (Cited on pages 14 and 41.)

[32] Ruoxi Wang, Yingzhou Li, Michael W Mahoney, and Eric Darve. Structured block basis factorization for scalable kernel matrix evaluation. *arXiv preprint arXiv:1505.00398*, 2015. (Cited on pages 14 and 41.)

[33] Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, page 370. IEEE Computer Society, 1998. (Cited on page 14.)

[34] Srinadh Bhojanapalli, Prateek Jain, and Sujay Sanghavi. Tighter low-rank approximation via sampling the leveraged element. In *Proceedings of the Twenty-Sixth Annual*

ACM-SIAM Symposium on Discrete Algorithms, pages 902–920. SIAM, 2015. (Cited on page 14.)

[35] Kenneth L Clarkson and David P Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 81–90. ACM, 2013. (Cited on page 14.)

[36] Prateek Jain, Praneeth Netrapalli, and Sujay Sanghavi. Low-rank matrix completion using alternating minimization. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 665–674. ACM, 2013. (Cited on pages 25, 30 and 31.)

[37] Emmanuel J Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717–772, 2009. (Cited on pages 25 and 26.)

[38] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011. (Cited on page 28.)

[39] Tianbao Yang, Yu-Feng Li, Mehrdad Mahdavi, Rong Jin, and Zhi-Hua Zhou. Nyström method vs random fourier features: A theoretical and empirical comparison. In *Advances in neural information processing systems*, pages 476–484, 2012. (Cited on page 29.)

[40] Mehrdad Mahdavi, Tianbao Yang, and Rong Jin. An improved bound for the nystrom method for large eigengap. *arXiv preprint arXiv:1209.0001*, 2012. (Cited on page 29.)

[41] L. Song, B. Boots, S. M. Siddiqi, G. J. Gordon, and A. J. Smola. Hilbert space embeddings of hidden Markov models. In *Proc. 27th Intl. Conf. on Machine Learning (ICML)*, 2010. (Cited on page 37.)

[42] Bo Dai, Bo Xie, Niao He, Yingyu Liang, Anant Raj, Maria-Florina F Balcan, and Le Song. Scalable kernel methods via doubly stochastic gradients. In *Advances in Neural Information Processing Systems*, pages 3041–3049, 2014. (Cited on page 37.)

[43] MA Woodbury. Inverting modified matrices, memorandum rept. 42. *Statistical Research Group, Princeton University, Princeton, NJ*, 316, 1950. (Cited on page 44.)