UNIVERSIDAD NACIONAL DE COLOMBIA

Scalable Kernel Methods using Randomized Numerical Linear Algebra

Iván Yesid Castellanos Martínez

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, Colombia
2021

Scalable Kernel Methods using Randomized Numerical Linear Algebra

Iván Yesid Castellanos Martínez

Tesis presentada como requisito parcial para optar al título de:
Magister en Ingeniería de Sistemas y Computación

Director:
Ph.D., Fabio Augusto Gonzalez Osorio

Línea de Investigación:
Applied Computing
Grupo de Investigación:
Mindlab

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, Colombia
2021

Dedication

A mis padres Gilma y Jesús.

Acknowledgements

I want to give my sincere gratitude to everybody who has helped me during the development of this thesis, first I would like to thank my advisor Dr. Fabio Gonzalez for his support and motivation during challenging moments and his valuable feedback through all phases of this work. Also, I want to express my gratitude towards the Universidad Nacional de Colombia, which has been a second home for me and special thanks to all the professors, colleagues and students who have been somehow part of this path, particularly to Juan Camilo. Finally and most importantly, I would like to thank my parents Gilma and Jesús for their unconditional support and inspiration.

Abstract

Título en inglés: Scalable kernel methods using randomized numerical linear algebra

Kernel methods are a set of machine learning algorithms that make use of a kernel function in order to represent data in an implicit high dimensional space, where linear optimization systems lead to non-linear relationships in the data original space and therefore finding complex patterns in the data. The main disadvantage of these methods is their poor scalability, as most kernel based algorithms need to calculate a matrix of quadratic order regarding the number of data samples. This limitation has caused kernel methods to be avoided for large scale datasets and use approaches such as deep learning instead. However, kernel methods are still relevant to better understand deep learning methods and can improve them through hybrid settings that combine the best of both worlds.

The main goal of this thesis is to explore efficient ways to use kernel methods without a big loss in accuracy performance. In order to do this, different approaches are presented and formulated, from which, we propose the learning-on-a-budget strategy, which is presented in detail from a theoretical perspective, including a novel procedure of budget selection. This strategy shows, in the experimental evaluation competitive performance and improvements to the standard learning-on-a-budget method, especially when selecting smaller approximations, which are the most useful in large scale environments.

Keywords: Machine learning, Kernel methods, Budget method, Randomized numerical linear algebra, Distance based hashing, Approximated methods

Resumen

Título en español: Métodos de kernel escalables utilizando álgebra lineal numérica aleatorizada

Los métodos de kernel corresponden a un grupo de algoritmos de aprendizaje maquinal que hacen uso de una función de kernel para representar implicitamente datos en un espacio de alta dimensionalidad, donde sistemas de optimización lineal guíen a relaciones no lineales en el espacio original de los datos y por lo tanto encontrando patrones complejos dento de los datos. La mayor desventaja que tienen estos métodos es su pobre capacidad de escalamiento, pues muchos algoritmos basados en kernel requiren calcular una matriz de orden cuadrática respecto al numero de ejemplos en los datos, esta limitación ha provocado que los metodos de kernel sean evitados en configuraciones de datos a gran escala y utilicen en su lugar tecnicas como el aprendizaje profundo. Sin embargo, los metodos de kernel todavía son relevantes para entender mejor los métodos de aprendizaje profundo y ademas pueden mejorarlos haciendo uso de estrategias híbridas que combinen lo mejor de ambos mundos.

El principal objetivo de esta tesis es explorar maneras eficientes de utilizar métodos de kernel sin una gran pérdida en precisión. Para realizar esto, diferentes enfoque son presentados y formulados, dentro de los cuales, nosotros proponemos la estrategía de aprendizaje utilizando budget, la cual es presentada en detalle desde una perspectiva teórica, incluyendo un procedimiento novedoso para la selección del budget, esta estrategia muestra en la evaluación experimental un rendimiento competitivo y mejoras respecto al método estandar de aprendizaje utilizando budget, especialmente cuando se seleccionan aproximaciones mas pequeñas, las cuales son las mas útiles en ambientes de gran escala.

Palabras clave: Aprendizaje maquinal, Metodos de kernel, Método de budget, Álgebra lineal numérica aleatorizada, Hashing basado en distancias, Métodos aproximados

.

Esta tesis de maestría se sustentó el 06 de 08 de 2021 a las 08:00am,
y fue evaluada por los siguientes jurados:

Germán Jairo Hernández Pérez Ph.D.
Universidad Nacional de Colombia

Francisco Albeiro Gómez Jaramillo Ph.D.
Universidad Nacional de Colombia

List of Figures

List of Tables

Content

1 Introduction

1.1. Motivation

Machine learning is a field of computer science which consists mainly in automatically finding relations, patterns, regularities or some kind of structure from data sources, so that, from the obtained patterns we can make predictions or assertions over new data from the same sources. Currently, this field has become important to solve various problems with outstanding results in different domains such as: speech recognition, automatic text translations, identification of medical pathologies and many others [Ahuja and Angra, 2017].

Many of these applications need to be applied in a matter of seconds and within limited computational systems, i.g. mobile devices [Wang et al., 2018], IoT [Kumar et al., 2017] and others. Those restrictions may also apply for the time and memory resources needed to train any learning model to solve these problems. Because of this, it is important for machine learning methods to not only provide accurate results, but also to be efficient in terms of time and memory. In fact, for many of those cases it is more desirable to have high efficiency with a small loss in terms of prediction performance than improve accuracy at the expense of a very high computational cost.

There are many different techniques to approach machine learning applications. One of them are kernel machines, which are based on finding patterns depending on a similarity function of the data, however this approach has drawbacks in the computational cost during the training process for big datasets, which limits its application for current large scale problems [Bengio et al., 2005]. Another approach are deep learning methods, which are based on learning through neural networks using multiple layers, the success of some deep learning architectures to solve different problems and the ease of training using GPU, which has made this approach very popular for different applications [Kamilaris and Prenafeta-Boldú, 2018, Litjens et al., 2017].

In recent years, deep learning methods have increased their popularity over kernel methods to solve diverse machine learning problems, this is because the experimentally deep learning has outperformed other methods (including kernel methods) in many different applications [Baveye et al., 2015]. However, there are multiple reasons why kernel methods can still contribute to the solution of different machine learning issues. One of the main reasons is the ease of the understanding of kernel methods, particularly optimization using kernel methods is very well acknowledged due to its strong mathematical development, contrasted to deep learning methods which to this day remain poorly understood [Belkin et al., 2018]. This can be even more important in view of recent works which claim that any method using gradient descent, such as deep learning, can be seen as kernel machines with specific types of kernel function [Domingos, 2020], which is very useful considering that most kernel learning algorithms do not have restrictions over the type of kernel being used as long as it is a valid kernel. Therefore, deeper analysis over kernel methods may even impact our understanding of deep learning.

Another reason to still studying kernel methods is their employment in different hybrid methods. Currently there are various approaches combining both kernel and neural networks capabilities, which have been shown to perform significantly well solving different learning tasks [Mehrkanoon and Suykens, 2018, Wang et al., 2017]. In that regard, it is always desirable to understand better how to improve kernel methods, particularly how to solve their efficiency and scalability issues.

1.2. Problem identification

Kernel methods in machine learning have been used broadly to solve a large number of problems, because these methods can find explicit non-linear relations between data through the use of a similarity function, i.e. the kernel. Nevertheless, in general these methods use the so-called kernel matrix [Shawe-Taylor and Cristianini, 2004], which is a matrix containing the kernel measure between every pair of elements in the dataset; therefore, it has quadratic size in regard to the number of training samples. The use of this matrix is very problematic when applying the kernel methods for big datasets, because it needs extensive time and memory resources.

There are various ways to optimize the time and memory performance of kernel methods using different approximations. One of the most relevant in this context is the Nyström Method [Drineas and Mahoney, 2005], which is based on a numerical approximation of the matrix kernel, and even though this method has a strong theoretical background, the experimental accuracy is similar to other simpler methods like the Budget Method [Wang et al., 2012], which, instead of the full kernel matrix, makes formulations from a submatrix of the kernel matrix. However, even though this method works in practice, it is not very well developed theoretically. There is also the random Fourier features technique, which aims to approximate the kernel function instead of the kernel matrix through properties of shift invariant kernels. [Chitta et al., 2012]

Additionally, there is a recent area of applied mathematics consisting in the development of algorithms to resolve large-scale linear algebra problems named Randomized Numerical Linear Algebra [Drineas and Mahoney, 2016], which could be exploited considerably to solve efficiency issues in kernel methods. This thesis addresses the problem of developing efficient kernel methods using randomized numerical linear algebra ideas. To make this, we explore different current kernel approximated methods together with their formulations, and we connect that to the formal theory of randomized numerical linear algebra; according to this the research questions are:

- How to use algorithms inspired in randomized numerical linear algebra to formulate efficient kernel methods to solve machine learning problems?
- Which error bounds for approximated kernel methods can be improved according to randomized linear algebra theory?
- How to select a probability distribution to improve approximated kernel methods accuracy?
- What should be the size of the random sampling for approximated methods to make them efficient and accurate for large scale datasets?

1.3. Objectives

The main objective of this thesis is to design a strategy for implementing scalable kernel methods using ideas from randomized numerical linear algebra and apply it to the development of concrete kernel methods for both supervised and unsupervised learning.

The specific objectives of this thesis are:

- To design a general strategy to formulate approximated kernel methods using theory and algorithms from randomized numerical linear algebra
- To implement the designed strategy for at least one supervised learning algorithm and one unsupervised learning algorithm
- To test the developed methods with large scale datasets and compare them with the current approximated kernel methods

1.4. Contribution

This work presents several contributions to solve the problems described above about the scalability and efficiency of kernel methods with the following outline:

1.4.1. Approximated kernel methods

Relevant theoretical results regarding kernel approximations are presented in the first part of this thesis. Particularly, an analysis and comparison of different approximated kernel methods is shown in detail, as well as important remarks about the use of these methods. Parts of this work were published in:

- Castellanos Martinez, I. Y., Toledo Cortés, S. & Gonzalez, F. A., *Large Scale Learning Techniques for Least Squares Support Vector Machines*, Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, 2019, pages 3–11, ISBN:978-3-030-13469-3

1.4.2. Budget formulation

An extensive theoretical formulation of the budget approximation method is presented with various considerations regarding the selection of the budget including some error bounds obtained from randomized numerical linear algebra theory. Particularly the expected accuracy performance from the budget method is shown for random budgets considering any probability distribution.

Also, different strategies to select the budget were analyzed and formulated considering, not only the accuracy performance, but also the time and memory efficiency impact.

1.4.3. Machine learning implementations

Within the framework of this thesis, two supervised methods, one semi-supervised method and one unsupervised method using the budget approximation were formulated and implemented using Python 3 programming language and IPython notebooks. It also includes the different strategies to select the budget; the code can be found at `https://github.com/iycastellanosm/BudgetMethods`.

Experimentation setup and corresponding results over these implementations are also presented as part of this thesis.

1.5. Thesis structure

The next chapters are organized as follows: Chapter 2 provides the general background on kernel methods and some of the most relevant approaches to do approximations on these kind of methods. Then, in Chapter 3 the budget strategy to approximate kernel methods is described in detail, and different techniques to select the budget are presented. Chapter 4 describes how the budget strategy can be applied to different learning algorithms in supervised and unsupervised manners, including various experimental results. Finally, Chapter 5 discuses some conclusions and future work related to this research.

2 Background

In this chapter we present some motivational topics and relevant ideas that were used in the development of this thesis.

2.1. Kernel methods

Kernel methods are a set of data pattern analysis algorithms that makes use of a kernel function, which is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, such that for any given $x_i, x_j \in \mathcal{X}$ samples of data $k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$, where $\phi(x)$ is a mapping to a feature space $\mathcal{H}$ [Shawe-Taylor and Cristianini, 2004]. These methods are beneficial as the mapping $\phi(x)$ may be to any arbitrary Hilbert space with higher (possibly infinite) dimension than the original space of the data, leading to complex non-linear representation of that data. Moreover, the most common use of kernel methods is for learning algorithms based on the dot product of the data; therefore, using a kernel function k and avoiding the explicit mapping to the feature space $\mathcal{H}$: this is called kernel trick [Scholkopf, 2001].

2.1.1. Formulation

Formally, let $X \in \mathbb{R}^{d \times n}$ be a dataset of n samples $x_1, x_2, ..., x_n$ of dimension d with label values $y_1, y_2, ..., y_n$. For a regression problem or a binary classification problem, we need to calculate a weight vector $w \in \mathbb{R}^d$ and a real intercept $b \in \mathbb{R}$, such that, for any sample x, the prediction function $pred(x) = w^T x + b$ is as close as possible as the real value of the sample. In case of a binary classification problem, the labels are just positive ($y_i = 1$) or negative ($y_i = -1$), therefore when $pred(x) > 0$ the sample would be classified as positive and when $pred(x) \leq 0$ the sample would be classified as negative. In this case, finding the best w and b to solve this is a linear optimization problem.

Furthermore, in a learning algorithm problem, we would obtain w from the data samples $x_1, x_2, ..., x_n$, i.e. $w = \sum_i^n \alpha_i x_i$ where $\alpha_i \in \mathbb{R}$, here we can denote $\alpha \in \mathbb{R}^n$ as the vector $(\alpha_1, \alpha_2, ..., \alpha_n)$, in this case finding the best w is equivalent to find the best α considering the previous equality, which is still a linear optimization problem. However, the data may not fit very well with linear functions and exhibit non-linear patterns instead. A solution for this is to define a function $\phi : \mathbb{R}^d \to \mathcal{H}$ which maps the data to a Hilbert high-dimensional space where the data would show linear patterns. Thus, w now would be expressed as $w = \sum_i^n \alpha_i \phi(x_i)$ and the prediction function now would be $pred(x) = \sum_i^n \alpha_i \phi(x_i)^T \phi(x) + b$. As we are working in a Hilbert space we get that $\phi(x_i)^T \phi(x) = \langle \phi(x_i), \phi(x) \rangle$, where $\langle ., . \rangle$ is the inner product in $\mathcal{H}$. As a consequence $pred(x) = \sum_i^n \alpha_i c + b$. Finally, if we have a kernel function k corresponding to ϕ, i.e. $k(x_i, x) = \langle \phi(x_i), \phi(x) \rangle$, our problem is equivalent to find the best α and b to model the function $pred(x) = \sum_i^n \alpha_i k(x_i, x) + b$. Note that with the kernel formulation, all elements x_i do not need to be any more elements of $\mathbb{R}^d$, but just elements from a set $\mathcal{X}$ with a valid kernel function, which allows us to apply easily these learning algorithms to non-numerical data like text [Kudo and Matsumoto, 2003] and graphs [Tian et al., 2019]. Also note that to define the function k it is neither necessary to map the elements explicitly to $\mathcal{H}$ nor to calculate the function ϕ and the formulation is still valid, this property is called kernel trick.

Finally, considering that in a learning algorithm we are training over the dataset $X = \{x_1, x_2, ..., x_n\}$ and that the vector w is a linear combination of all elements from X, then we would need to eventually calculate $k(x_i, x_j)$ for all pairs $x_i, x_j \in X$. This calculation can be done once and be stored in a matrix K, called kernel matrix [Bousquet and Herrmann, 2003], which we would use during the corresponding kernel learning algorithm. Nevertheless, note that the calculation of K has $\Omega(n^2)$ time complexity and the storage of this matrix has $\Omega(n^2)$ space complexity, therefore any algorithm using these matrices has already these lower bounds by default. Now let the sets $X_1, X_2 \subseteq \mathcal{X}$ of sizes n_1 and n_2 respectively, we define $k(X_1, X_2) \in \mathbb{R}^{n_1 \times n_2}$ as the matrix with $k(X_1, X_2)_{i,j} = k(x_i, x_j)$ for $x_i \in X_1$ and $x_j \in X_2$. This notation is useful as we can consider the resulting matrix for different sets, in particular $K = k(X, X)$.

Ridge regression

When looking at a regression learning problem, if we were doing linear regression in the original space we can obtain an analytical solution of w as $w = (XX^T)^{-1}Xy$ and adding a regularization parameter we get $w = (XX^T + \lambda I)^{-1}Xy$. Also as we stated in the previous section, this means that $w = X\alpha$, a linear combination of elements from X, rearranging the terms in this case we get that $\alpha = (XX^T + \lambda I)^{-1}y$, this is called ridge regression. Moreover, we can use the kernel function as previously defined and considering that $\phi(X)\phi(X)^T = k(X, X)$ we could have an analytical solution of a regression problem calculating $\alpha = (K + \lambda I)^{-1}y$, however this not only forces us to have the full kernel matrix, but also to calculate the inverse of that matrix, which means this has a time complexity of $O(n^3)$ and space complexity of $O(n^2)$, making it suitable just to small datasets.

2.1.2. Applications

There are many algorithms which can find patterns from data using only the dot product from the space data is located on. All these algorithms are suitable to be kernelized, i.e. they can be formulated using kernels. One of the most well known is the support vector machines [Hearst et al., 1998] which separates data maximizing the gap between two classes. Over the past two decades kernel methods have managed to be popular in the industry due to their high flexibility, particularly when choosing the kernel function k. As an example of this, we may see applications on speech recognition [Ganapathiraju et al., 2004], chemistry [Li et al., 2009], signal processing [Rojo-Álvarez et al., 2018], face recognition [Yang, 2002], bioinformatics [Borgwardt, 2011] and many others.

Recently, many machine learning algorithms, including kernel methods, have been outperformed by deep learning in various applications [Wang et al., 2018, Min et al., 2017]. Aside from the experimental gains of accuracy in this methods, another advantage is the ease of training, as deep learning methods have been improved to be trained over GPUs, it is possible (and desirable) to train over datasets with millions of samples, as opposed to kernel methods, where their need of calculating a quadratic matrix limits that possibility. However, in contrary to kernel methods, deep learning is not very well understood, which has limited their potential of improvement and the general interpretability of results after training.

These factors have driven many of current works in kernel methods to be focused on their ability to improve deep learning instead of trying to outperform it. For instance, [Domingos, 2020] explains how any method based on stochastic gradient descent, like neural networks, can be formulated as kernel machines using a specific type of kernel, therefore improving knowledge over kernel methods can lead to improvements in deep learning as well. Other works include ways to combine advantages of deep learning and kernel methods to obtain the best of both worlds. One example is [Mehrkanoon et al., 2017] which shows how hybrid architectures using neural networks with some layers of kernel transformations can be developed to solve large scale learning problems. Another example is [Chen et al., 2020] which present convolutional kernel methods, which are essentially kernel methods in a multiple layer architecture instead of a shallow one to solve classification problem for graph structured data.

Distinctly, some of the recent works on kernel methods strive to explain specific cases where these methods can perform better or are equivalent to other techniques. For instance, [Schuld, 2021] presents how quantum machine learning models are essentially kernel methods using data-encoding quantum states. [Garriga-Alonso et al., 2018] shows how deep convolutional networks with an appropriate prior over the weights behave as a shallow Gaussian process kernel. As another example, [Pilario et al., 2020] explains how kernel methods can be used in feature extraction for nonlinear processes. Finally, other recent works on kernel methods try to reduce their efficiency limitations. [Sheikholeslami and Giannakis, 2017, Zhang et al., 2017, Chitta et al., 2014] show various approaches of working with kernels for large scale settings to solve specific problems. These approaches aim to obtain the advantages of kernel methods avoiding the construction of the kernel matrix.

2.2. Approximated kernel methods

Traditional approaches of kernel methods use the kernel matrix, which has evaluations of the kernel function for all pairs of samples in the data. Therefore, when we have n samples, only calculating the kernel matrix is $\Omega(n^2)$, which is impractical when handling with large datasets. To solve this, several approaches have been developed to make approximations for kernel methods avoiding the calculation of the whole kernel matrix, in the literature we found these approaches:

- *Low-rank approximation*: instead of calculating and using the $n \times n$ kernel matrix K, we could use a low-rank approximation, i.e. a matrix $\hat{K}$, such that, $\left\|K - \hat{K}\right\|$ is small and $rank(\hat{K}) < r$ where $r \ll n$. The most common procedure of this approach is the Nyström Method [Gittens and Mahoney, 2013].

- *Feature mapping approximation*: another common approach to avoid costly calculations in kernel methods is through an approximation of the kernel mapping to the implicit high dimensional feature space, i.e. a function $\hat{\phi}(x)$, such that, $\left\|k(x_i, x_j) - \langle\hat{\phi}(x_i), \hat{\phi}(x_j)\rangle\right\|$ is small for all pairs of data samples and $\hat{\phi}(x)$ is a low dimensional function. Therefore, the learning algorithms could be used with the explicit approximated function $\hat{\phi}(x)$ instead of the kernel function, avoiding the calculation of the kernel matrix. The most common procedure of this approach is through Random Fourier Features [Rahimi and Recht, 2007].

- *Budget restriction*: a third approach found in the literature is to include a restriction in the learning algorithms, in which, a relevant subset of $b \ll n$ samples is considered to be the basis of the learning space for all data points. Hence, instead of calculating a kernel matrix of size $n \times n$, we would have a matrix of size $n \times b$. Moreover, as b is a parameter, it can be tuned for large datasets. This is usually known as learning on a budget [Wang et al., 2012].

2.2.1. Nyström method

Let K be a symmetric positive semi-definite matrix, and $U\Delta U^T$ the eigendecomposition of K, i.e. U is an orthonormal matrix and $\Delta = diag(\lambda_i)$, where $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n \geq 0$ are the eigenvalues of K. Given some $p \ll n$ we can build U_p from the first columns of U and $\Delta_p = diag(\lambda_1, ..., \lambda_p)$, then $K = U\Delta U^T \approx U_p \Delta_p U_p^T$, this is a p-rank approximation of K. This decomposition is suitable for kernel matrices, as for any valid kernel function its corresponding matrix is always symmetric positive semi-definite, however the calculation of the eigendecomposition is $O(n^3)$, which is infeasible for large scale data.

Nevertheless, the Nyström procedure computes an approximation of this eigendecomposition [Williams and Seeger, 2001, Drineas and Mahoney, 2005]. Given m, with $p \leq m < n$, choose m indexes $j_1, ..., j_m$ from $1, ..., n$, and build matrix C choosing the columns $j_1, ..., j_m$ of K and scaling them, then build matrix W choosing the rows $j_1, ..., j_m$ of C and scaling them. Now calculate $W_p = \hat{U}_p \hat{\Delta}_p \hat{U}_p^T$, the p-rank approximation of W, with this eigencomposition the Moore-Penrose generalized inverse of W_p, noted as W_p^+ can be built [Courrieu, 2008]. Finally $\hat{(}K) = CW_p^+ C^T$, is the p-rank Nyström approximation of K, which is calculated in $O(m^3 + pmn)$.

Now for any kernel method that uses the kernel matrix K directly in the training and prediction formulation it suffixes to replace the matrix K with $\hat{K} = CW_p^+ C^T$. Moreover, when $m \ll \sqrt{n}$ then to calculate the Nyström approximation is much better than the exact $\theta(n^2)$ calculation of the full matrix, consequently, it is less expensive in time and additionally, it can be stored using less memory. For this reason, Nyström method has been used extensively as a way to improve kernel methods in various tasks like clustering [Choromanska et al., 2013, Wang et al., 2019], classification [Hoi et al., 2013] and regression [Trokicic and Todorovic, 2020].

Another important remark about this approximation of kernel methods is regarding the selection of the m column-indexes, which corresponds to a sampling of m data points. Selecting a more appropriate subset of the data, instead of just a uniformly random one, could improve significantly the accuracy of Nyström in most learning related tasks [Kumar et al., 2012, Wang and Zhang, 2013]. Finding an optimal selection of these samples is a relevant problem on its own [Paul et al., 2015].

2.2.2. Random features

Let $k : X \times X \to \mathbb{R}$ be a positive semi-definite function, Mercer's theorem [Sun, 2005] states that there is a map $\phi : X \to Y$ where Y is a Hilbert space such that k corresponds to the dot product in Y, i.e. $k(x, x') = \langle \phi(x), \phi(x') \rangle$. Furthermore, Y could be a very high, even possibly infinite dimensional space. Subsequently, often it is not possible to use in practice function ϕ explicitly, although we can consider to use a low dimensional approximation of the function ϕ instead.

Let $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a positive shift-invariant function, i.e. There exists a function $k' : \mathbb{R}^d \to \mathbb{R}$, such that $k(x, x') = k'(x - x')$. Bochner's theorem [Rudin, 1990] states that k is positive definite if and only if $k'(\delta)$ is the Fourier transform of a non-negative measure. As a result of this theorem, calculating the Fourier transform of a shift-invariant kernel $p(w) = \frac{1}{2\pi} \int e^{-iw'\gamma} k'(\gamma) d\Delta$ gives a probability function, which can be used to make the Random Fourier Features approximation $\hat{\phi}$ of the map ϕ by its Monte-Carlo estimate. In particular, choosing m iid samples $w_1, ..., w_m \in \mathbb{R}^d$ from $p(w)$ and m iid samples $b_1, ..., b_m \in \mathbb{R}$ uniformly from $[0, 2\pi]$, we build $\hat{\phi}(x) = \sqrt{\frac{1}{m}}(\cos(w_1^T x+b_1), ..., \cos(w_m^T x+b_m), \sin(w_1^T x+b_1), ..., \sin(w_m^T x+b_m))$. Then, $k(x, x') \approx \langle \hat{\phi}(x), \hat{\phi}(x') \rangle$ [Sriperumbudur and Szabo, 2015].

Additionally to Random Fourier Features, there are other ways of approximate ϕ such that $k(x, x') \approx \hat{\phi}(x)^T \hat{\phi}(x')$, approximations using this approach are named just Random Features (RF) [Rudi and Rosasco, 2017]. Other examples of RF are Random Binning Features [Wu et al., 2018] which partitions the input space using randomly shifted grids with different, randomly chosen resolutions and allocates every point in the input to the bin it falls using a binary string. Grids are chosen in such a way that the probability of two points x and x' being in the same bin is approximately proportional to $k(x, x')$. There is also orthogonal Random Features [Yu et al., 2016] which aims to approximate the Gaussian kernel imposing orthogonality on the matrix with a linear transformation randomly generated. Random Features have shown many capabilities, for instance [Sinha and Duchi, 2016] shows how to use random features in order to approximate the kernels directly instead of having to define one as input of the learning algorithm, this is feasible as RF are approximating the kernel function and therefore they may be capable of modifying it as needed.

Note that using these types of approximations make it possible for learning methods to use explicitly the mapping function $\hat{\phi}$ as this is low dimensional. Hence, the learning algorithms can be directly evaluated without using the kernel trick, mapping the data to the feature space generated by $\hat{\phi}$ and finding patterns there directly to solve different tasks like regression [Huang et al., 2013] and classification [Li and Marlin, 2015]. Furthermore, it is important to consider that Random Features are limited only to a particular set of kernels, which limits its potential applications.

2.2.3. Budget approximation

Learning on a budget strategy in kernel methods is motivated by support vector machines algorithm, which solves an optimization problem through the calculation of some support vectors. These vectors are a subset of the training data which is typically found during the learning process. The budget strategy basically consists in an a priori restriction on the number of support vectors in order to make the learning process much faster. In particular, we would restrict the algorithm to have $m < n$ support vectors [Wang et al., 2012].

Even though the budget notion is based on the support vector machines algorithm, it is possible to extend this strategy to other kernel algorithms, where the restriction corresponds to a representation of the data using only the subset defined by the budget, but we can still train the model using the whole dataset. Formally, let $x \in X$ be a training data sample and $B \subset X$ the budget subset of size m, we define the restriction over the learning algorithms such that the optimization problem is defined in terms of x and B, i.e. we have a loss function $L(x, B)$ instead of the traditional $L(x, X)$ in our training process. This approach in not only found in different versions of support vector machines [Jian et al., 2017, Schölkopf et al., 2007] but also in not supervised kernel-based algorithms [Vanegas et al., 2018].

It is important to remark about the budget strategy that it is very efficient in large scale datasets when 2 conditions are held: first, m should be very small compared to n, otherwise it would be close to the $O(n^2)$ traditional kernel matrix calculation, and second, the method to select the budget is also efficient in comparison to the training procedure. Nevertheless, those conditions are opposite to good results in the learning algorithms. The selection of the budget size and its elements is then a relevant problem on its own [Glasmachers and Qaadan, 2018].

3 Budget selection

In this chapter we present theoretical bounds regarding the formulation of budget approach for support vector machines and different budget selection strategies using randomized numerical linear algebra theory.

3.1. Budget formulation

Many learning algorithms are based on the kernel method formulation shown in chapter 2, notably the support vector machines (SVM). However, that formulation corresponds to a quadratic optimization problem, because evaluating $pred(x)$ has linear time complexity regarding n, therefore just the evaluation over all n samples in the dataset has $\Theta(n^2)$ computational time complexity. In particular, given $\alpha \in \mathbb{R}^n$ evaluating the prediction function for the whole dataset corresponds to calculate $f(\alpha) = k(X, X)\alpha + b^{(n)}$ where $b^{(n)} \in \mathbb{R}^n$ is a vector consisting of n times the value b and $k(X, X)$ is the kernel matrix, in this case $f(\alpha)$ is a vector of size n, such that $f(\alpha)_i$ is the predicted value for the *i-th* sample. This formulation is not scalable for big datasets, in order to solve this we add the budget restriction, which approximates w to be a combination of a subset of $B \subseteq X$ instead of a combination of the whole dataset, i.e. Let $B = \{x_{i_1}, x_{i_2}, ..., x_{i_c}\}$, then $\hat{w} = \sum_{k=1}^{c} \hat{\alpha}_k x_{i_k}$, and we would have that $\hat{f}(\hat{\alpha}) = k(X, B)\hat{\alpha} + b^{(n)}$, in this case $\hat{\alpha} \in \mathbb{R}^c$ and $k(X, B)$ is the kernel matrix between elements of X and elements of B, therefore the computational complexity is now $O(nc)$ which is much better than the original formulation when $c \ll n$. However this may be a constraint where the methods are formulated with the full kernel matrix, such as the ridge regression, which just by definition in the dual uses always the full kernel matrix.

When using the budget method, the quality of the learned patterns depends on the budget selection, in particular, for methods like SVM, the optimal budget corresponds to the set of support vectors obtained in the optimal solution in the SVM. However, this would imply calculating the optimization problem with the original quadratic formulation, which is infeasible for large datasets and thus we need to avoid it. Therefore, it is necessary to consider more efficient strategies to select the budget, but then our solution could be sub-optimal in comparison to the solution with the exact formulation. Consequently, we have here a trade-off between efficiency and accuracy. Considering this, we want to focus only on strategies to select the budget with computational complexity close or better than $O(nc)$, which is what we would need in any case to calculate the kernel matrix between the budget and the full dataset.

3.1.1. Randomized numerical linear algebra

There are various approaches to select the budget, however considering the limitations we have to select this subset of the dataset in order to be efficient, i.e. avoiding to calculate the full kernel matrix, meaning that these strategies should be heuristic at some point. In this regard, a common approach to be considered is randomization, probabilistic algorithms have held a central role in scientific computing for many decades now, which have been useful to give competitive suboptimal solutions to different problems in computer science that treated otherwise could have been infeasible to solve. Particularly, there has been some recent work in the area of randomized numerical lineal algebra (RNLA) which consists in the development of randomized algorithms to solve traditional numerical linear algebra problem such as: singular value decomposition, calculation of eigenvalues and eigenvectors, orthogonalization, least squares, solution of sparse and dense linear systems and others [Martinsson and Tropp, 2021].

RNLA has gained relevance in machine learning applications as the algorithms and strategies from this field have shown outstanding results in applications like matrix factorization [Witten and Candes, 2015], embeddings [Mineiro and Karampatziakis, 2015] and matrix approximation [Cohen et al., 2015] and other algorithms used to solve very important learning related issues. In this regard, RNLA encapsulates a set of methods with the same idea of solving problems in a randomized manner, with some probability distribution, and obtaining from it different theoretical error bounds and mathematical guarantees about the performance of the algorithms. In the case of the budget selection, the problem of finding a good subset of the data to be used in the approximation kernel method falls within the category of numerical linear algebra algorithms we need to solve in an efficient manner. Moreover in the best of our knowledge, this has not been explored yet from a theoretical point of view using other RNLA approaches.

3.2. Random budget

The first approach can be considered is a naive approach consisting in selecting the budget as a random subset of the elements in the dataset using a uniform distribution. In this section we will show that selecting the budget in this random manner gives in average already acceptable results when used in learning algorithms based on kernel methods. Also, we show how the budget size impacts the performance of the approximation from a theoretical point of view. Finally, some experimental results applied to a support vector machines algorithm are presented.

3.2.1. Theoretical results

We will show that, when selecting the values of B randomly, in average the values of the approximated prediction function $\hat{f}(\hat{\alpha})$ are not far from the values in the original prediction function $f(\alpha)$, consequently obtaining a theoretically good approximation. To show this we need the following lemmas from Randomized Numerical Linear Algebra theory [Drineas and Mahoney, 2017]:

Lemma 3.2.1. *Let $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times q}$ and a probability distribution $\{p_i\}_{i=1}^n$. Choose $c > 0$ elements from $\{1, ..., n\}$ randomly from the distribution, let $S \in \mathbb{R}^{p\times c}$ be a matrix with $S_{i,t} = \frac{1}{\sqrt{cp_{i_t}}}$ if i_t is chosen in the t-th trial, and 0 otherwise, then:*

- $E[(ASS^TB)_{i,j}] = (AB)_{i,j}$
- $Var[(ASS^TB)_{i,j}] = \sum_{k=1}^n \frac{A_{i,k}^2 B_{k,j}^2}{cp_k} - \frac{(AB)_{i,j}^2}{c}$

Proof. Fix i and j and consider $X_t = \left(\frac{A_{*,i_t}B_{i_t,*}}{cp_{i_t}}\right)_{i,j} = \frac{A_{i,i_t}B_{i_t,j}}{cp_{i_t}}$ for $t = 1, ..., c$.

Now, $E[X_t] = \sum_{k=1}^n p_k X_k = \sum_{k=1}^n p_k \frac{A_{i,k}B_{k,j}}{cp_k} = \frac{1}{c}(AB)_{i,j}$.

Note that $(ASS^TB)_{i,j} = \sum_{t=1}^c X_t$. Thus $E[(ASS^TB)_{i,j}] = E[\sum_{t=1}^c X_t] = \sum_{t=1}^c E[X_t] = (AB)_{i,j}$.

Additionally, $E[X_t^2] = \sum_{k=1}^n \frac{A_{i,k}^2 B_{k,j}^2}{c^2 p_k}$, then $Var[X_t] = E[X_t^2] - E[X_t]^2 = \sum_{k=1}^n \frac{A_{i,k}^2 B_{k,j}^2}{c^2 p_k} - \frac{1}{c^2}(AB)_{i,j}^2$.

Finally $Var[(ASS^TB)_{i,j}] = Var[\sum_{t=1}^c X_t]$, as the samples are independent, then $Var[\sum_{t=1}^c X_t] = \sum_{t=1}^c Var[X_t] = \sum_{t=1}^c \left(\sum_{k=1}^n \frac{A_{i,k}^2 B_{k,j}^2}{c^2 p_k} - \frac{1}{c^2}(AB)_{i,j}^2\right) = \sum_{t=1}^c \frac{1}{c^2}\left(\sum_{k=1}^n \frac{A_{i,k}^2 B_{k,j}^2}{p_k} - (AB)_{i,j}^2\right) = \sum_{k=1}^n \frac{A_{i,k}^2 B_{k,j}^2}{cp_k} - \frac{(AB)_{i,j}^2}{c}$ □

An important remark about the previous lemma is that the result holds for any probability distribution. Moreover, for a uniform distribution, we have the following corollary:

Corollary 3.2.1.1. *Let $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times q}$. Choose $c > 0$ elements from $\{1, ..., n\}$ randomly from an uniform distribution, let $D \in \mathbb{R}^{n\times c}$ be a matrix with $D_{i,j} = 1$ if i_j is chosen in the j-th trial, and 0 otherwise, then:*

- $E[(ADD^TB)_{i,j}] = \frac{c}{n}(AB)_{i,j}$ *and* $E[\frac{n}{c}(ADD^TB)_{i,j}] = (AB)_{i,j}$

- $Var[(ADD^TB)_{i,j}] = \frac{c}{n}\left(\sum_{k=1}^{n}(A_{i,k}B_{k,j})^2 - \frac{(AB)_{i,j}^2}{n}\right)$ *and*
  $Var[\frac{n}{c}(ADD^TB)_{i,j}] = \frac{n}{c}\left(\sum_{k=1}^{n}(A_{i,k}B_{k,j})^2 - \frac{(AB)_{i,j}^2}{n}\right)$

Proof. Using a uniform probability distribution on the lemma we get that $S = \frac{1}{\sqrt{c\frac{1}{n}}}D$, then $D = \sqrt{\frac{c}{n}}S$ and therefore $ADD^TB = \frac{c}{n}ASS^TB$. Applying that $E[aX] = aE[X]$ and $Var[aX] = a^2Var[X]$ we have the result. □

Example 3.2.1. *To ilustrate the previous corollary lets consider matrices* $A = \begin{bmatrix}3 & 8 & 5\\5 & 1 & 8\\2 & 6 & 7\end{bmatrix}$ *and* $B = \begin{bmatrix}2 & 5 & 1\\4 & 8 & 9\\8 & 9 & 3\end{bmatrix}$, *with multiplication* $AB = \begin{bmatrix}78 & 124 & 90\\78 & 105 & 38\\84 & 121 & 77\end{bmatrix}$.
let's consider the case when we get just 1 sample $(c = 1)$, *then we have 3 possible matrices* D *with* $\frac{1}{3}$ *probability each, that is* $D = D_1 = \begin{bmatrix}1\\0\\0\end{bmatrix}$, $D = D_2 = \begin{bmatrix}0\\1\\0\end{bmatrix}$ *or* $D = D_3$ $\begin{bmatrix}0\\0\\1\end{bmatrix}$, *and we have the following possibilities:*

- $\frac{3}{1}AD_1D_1^TB = 3\begin{bmatrix}3 & 8 & 5\\5 & 1 & 8\\2 & 6 & 7\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}\begin{bmatrix}1 & 0 & 0\end{bmatrix}\begin{bmatrix}2 & 5 & 1\\4 & 8 & 9\\8 & 9 & 3\end{bmatrix} = 3\begin{bmatrix}6 & 15 & 3\\10 & 25 & 5\\4 & 10 & 2\end{bmatrix} = \begin{bmatrix}18 & 45 & 9\\30 & 75 & 15\\12 & 30 & 6\end{bmatrix}$

- $\frac{3}{1}AD_2D_2^TB = 3\begin{bmatrix}3 & 8 & 5\\5 & 1 & 8\\2 & 6 & 7\end{bmatrix}\begin{bmatrix}0\\1\\0\end{bmatrix}\begin{bmatrix}0 & 1 & 0\end{bmatrix}\begin{bmatrix}2 & 5 & 1\\4 & 8 & 9\\8 & 9 & 3\end{bmatrix} = 3\begin{bmatrix}32 & 64 & 72\\4 & 8 & 9\\24 & 48 & 54\end{bmatrix} = \begin{bmatrix}96 & 192 & 216\\12 & 24 & 27\\72 & 144 & 162\end{bmatrix}$

- $\frac{3}{1}AD_3D_3^TB = 3\begin{bmatrix}3 & 8 & 5\\5 & 1 & 8\\2 & 6 & 7\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix}\begin{bmatrix}0 & 0 & 1\end{bmatrix}\begin{bmatrix}2 & 5 & 1\\4 & 8 & 9\\8 & 9 & 3\end{bmatrix} = 3\begin{bmatrix}40 & 45 & 15\\64 & 72 & 24\\56 & 63 & 21\end{bmatrix} = \begin{bmatrix}120 & 135 & 45\\196 & 216 & 72\\168 & 189 & 63\end{bmatrix}$

therefore

$$\begin{aligned}E[\frac{n}{c}ADD^TB] &= \frac{1}{3}\begin{bmatrix}18 & 45 & 9\\30 & 75 & 15\\12 & 30 & 6\end{bmatrix} + \frac{1}{3}\begin{bmatrix}96 & 192 & 216\\12 & 24 & 27\\72 & 144 & 162\end{bmatrix} + \frac{1}{3}\begin{bmatrix}120 & 135 & 45\\196 & 216 & 72\\168 & 189 & 63\end{bmatrix}\\ &= \begin{bmatrix}78 & 124 & 90\\78 & 105 & 38\\84 & 121 & 77\end{bmatrix}\\ &= AB\end{aligned}$$

.

Let's suppose now that we have $c = n = 3$ samples, in that case we could have matrices like $D = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ where $\frac{n}{c} ADD^T B = AB$, but as we are allowing repetition we could have also matrices like $D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$ where not necessarily $\frac{n}{c} ADD^T B = AB$, therefore we would get that variance may be greater than 0. However, if we add the restriction of sampling without repetition, we can check that variance would be 0, as in this particular case the matrix D would be an orthogonal matrix and then $ADD^T B = AIB = AB$.

Now, when selecting c elements $\{i_1, i_2, ..., i_c\}$ from $\{1, ..., n\}$ randomly from an uniform distribution we construct the budget B as $B = [x_{i_1}, x_{i_2}, ..., x_{i_c}]$. Let $D \in \mathbb{R}^{n \times c}$ be a matrix with $D_{i,j} = 1$ if i_j is chosen in the *j-th* trial and 0 otherwise, then note that $k(X, B) = k(X, X)D$. Finally, if we fix $\alpha \in \mathbb{R}^n$ we can calculate $\hat{f}(D^T \alpha) = k(X, B)D^T \alpha + b^{(n)}$, and using the previous observation, we have that $\hat{f}(D^T \alpha) = k(X, X)DD^T \alpha + b^{(n)}$, which corresponds to the approximation described in the previous corollary, meaning that the budgeted prediction function $\hat{f}(\hat{\alpha})$ gives us a randomized numerical linear approximation of the original prediction function $f(\alpha)$, multiplied by a constant, with the corresponding properties. In particular, the expected value of the approximated function is the same as the original function multiplying by a constant. For classification this means that the expected predicted class using the budgeted function is equal to the predicted class with the original function. However, note that the variance depends on the ratio between the original size and the budget size, so that when the budget is extremely small the result could be very far from the expectation.

3.2.2. Experimentation

In this section we compare experimentally the budget method with a random budget to the non-approximated version and the Nyström approximation for standard binary SVM classification applied to two datasets. We evaluate the performance of the algorithms calculating the classification accuracy and training time for all datasets, in tables **3-1**, **3-2**, **3-3** and **3-4** we used kernels linear, polynomial, RBF and Chi-square respectively for two classes of the MNIST [LeCun and Cortes, 2010], while in tables **3-5**, **3-6**, **3-7** and **3-8** we used same kernels applied to two classes of the Fashion-MNIST dataset [Xiao et al., 2017]. For all kernels we have a regularization parameter c and for RBF and Chi-square we have a parameter γ, these parameters were fine-tuned for all algorithms. As the approximations are random the presented results correspond to the mean and standard deviation of 10 runs for each configuration.

Size	Method	Accuracy	Time
16	Nyström	74,535 ± 3,112	42.958
	Budget	77,822 ± 1,597	30.205
64	Nyström	80,214 ± 1,889	87.304
	Budget	83,522 ± 1,038	62.107
256	Nyström	86,55 ± 0,479	190.894
	Budget	88,171 ± 0,515	126.457
1024	Nyström	89,417 ± 0,339	368.620
	Budget	90,802 ± 0,714	242.626
4096	Nyström	92,390 ± 0,246	887.915
	Budget	92,301 ± 0,433	648.331
full dataset		95,824	5782.805

Table 3-1: SVM Approximations using linear kernel for MNIST

Size	Method	Accuracy	Time
16	Nyström	71,835 ± 4,112	46.900
	Budget	74,257 ± 2,732	31.205
64	Nyström	76,563 ± 2,374	90.621
	Budget	79,544 ± 1,857	64.009
256	Nyström	79,653 ± 0,754	200.489
	Budget	81,574 ± 0,945	130.788
1024	Nyström	83,683 ± 0,584	412.285
	Budget	85,288 ± 1,013	298.833
4096	Nyström	88,457 ± 0,463	928.729
	Budget	88,564 ± 0,735	687.915
full dataset		92,484	6182.735

Table 3-2: SVM Approximations using polynomial kernel for MNIST

Size	Method	Accuracy	Time
16	Nyström	78,882 ± 2,527	56.993
	Budget	80,927 ± 1,263	40.002
64	Nyström	83,583 ± 1,634	96.162
	Budget	84,184 ± 1,185	81.904
256	Nyström	86,286 ± 0,463	218.142
	Budget	87,376 ± 0,542	187.391
1024	Nyström	90,285 ± 0,285	462.282
	Budget	90,142 ± 0,653	328.208
4096	Nyström	94,274 ± 0,372	968.331
	Budget	93,194 ± 0,475	717.522
full dataset		96,733	6821.752

Table 3-3: SVM Approximations using RBF kernel for MNIST

Size	Method	Accuracy	Time
16	Nyström	73,185 ± 2,464	59.205
	Budget	76,046 ± 1,383	42.900
64	Nyström	79,173 ± 1,935	98.009
	Budget	80,683 ± 1,258	83.384
256	Nyström	83,573 ± 0,583	220.489
	Budget	84,568 ± 0,654	189.475
1024	Nyström	87,342 ± 0,472	468.117
	Budget	87,566 ± 0,735	332.520
4096	Nyström	91,734 ± 0,582	973.211
	Budget	90,246 ± 0,834	720.911
full dataset		94,282	6973.543

Table 3-4: SVM Approximations using Chi-square kernel for MNIST

Size	Method	Accuracy	Time
16	Nyström	72,535 ± 3,573	42.958
	Budget	73,822 ± 1,735	30.205
64	Nyström	74,214 ± 1,835	87.304
	Budget	76,522 ± 1,584	62.107
256	Nyström	78,55 ± 0,856	190.894
	Budget	79,171 ± 0,573	126.457
1024	Nyström	81,417 ± 0,586	368.620
	Budget	81,802 ± 0,634	242.626
4096	Nyström	83,390 ± 0,573	887.915
	Budget	83,301 ± 0,434	648.331
full dataset		85,372	5782.805

Table 3-5: SVM Approximations using linear kernel for Fashion

Size	Method	Accuracy	Time
16	Nyström	72,645 ± 4,457	46.900
	Budget	73,554 ± 2,865	31.205
64	Nyström	77,272 ± 2,784	90.621
	Budget	79,352 ± 1,567	64.009
256	Nyström	80,255 ± 1,346	200.489
	Budget	81,245 ± 1,235	130.788
1024	Nyström	83,457 ± 0,357	412.285
	Budget	83,856 ± 1,568	298.833
4096	Nyström	86,354 ± 0,632	928.729
	Budget	86,356 ± 0,234	687.915
full dataset		87,331	6182.735

Table 3-6: SVM Approximations using polynomial kernel for Fashion

Size	Method	Accuracy	Time
16	Nyström	68,5355 ± 2,433	56.993
	Budget	70,822 ± 1,835	40.002
64	Nyström	73,214 ± 1,860	96.162
	Budget	74,5225 ± 1,434	81.904
256	Nyström	76,55 ± 1,481	218.142
	Budget	77,171 ± 1,234	187.391
1024	Nyström	80,417 ± 1,185	462.282
	Budget	80,8025 ± 1,246	328.208
4096	Nyström	84,3905 ± 0,635	968.331
	Budget	83,301 ± 0,865	717.522
full dataset		86,925	6821.752

Table 3-7: SVM Approximations using RBF kernel for Fashion

Size	Method	Accuracy	Time
16	Nyström	65,657 ± 2,573	59.205
	Budget	69,683 ± 1,352	42.900
64	Nyström	67,547 ± 1,572	98.009
	Budget	70,546 ± 1,786	83.384
256	Nyström	71,146 ± 1,325	220.489
	Budget	72,171 ± 0,482	189.475
1024	Nyström	73,774 ± 0,754	468.117
	Budget	73,8025 ± 0,564	332.520
4096	Nyström	85,346 ± 0,693	973.211
	Budget	85,301 ± 0,583	720.911
full dataset		87,236	6973.543

Table 3-8: SVM Approximations using Chi-square kernel for Fashion

3.3. Budged based on landmarks

We showed that a random budget is good enough in average comparing it to the original kernel formulation, and it actually gets very close to the original function when the budget gets larger in size. However, for big datasets, and in order to have scalability, it is important to have a budget selection method for small budgets, which could improve the results compared to the random method, but also considering that this selection method should avoid the calculation of the full kernel matrix.

3.3.1. Column selection

In RNLA theory the problem described before is the column subset selection problem (CSSP) [Wang et al., 2016] which, for a matrix $A \in \mathbb{R}^{m \times n}$, consists in finding a matrix $X \in \mathbb{R}^{m \times c}$ containing c columns of A for which $A - XX^{+}A$ is small.

For kernel matrix methods, we aim to approximate K with a low dimensional approximation, exactly the role that X does in the CSSP. However, the original CSSP algorithm relies on having the original approximate matrix, therefore if applied directly to kernel matrix K, we would need to calculate that matrix beforehand, which could be too expensive. A strategy to solve this is to make a random presampling on the columns, approximating that matrix instead of the full matrix K [Boutsidis et al., 2009].

Note that $XX^{+}A$ is the best possible reconstruction of A by projection into the space spanned by the columns of X [Paul et al., 2015], so with this approach we would be able to get good columns for which the span of X may contain similar information as the original matrix A. For our learning algorithms, we would use the subset obtained by the aproximated CSSP as the budget.

3.3.2. Locality sensitive hashing

Another approach to select representative points which maximizes the information about a dataset i.e. the budget, is to obtain different points distributed evenly along the whole space, this could be done by clustering and selecting a relevant point by each calculated cluster. However, we need to consider that the budget selection process should be computationally efficient, in other words, this process will not be using traditional clustering methods like kernel K-means which would need to calculate the whole kernel matrix.

In this context we can use Locality-Sensitive Hashing (LSH) which is a fast technique to find landmarks in a space through the use of fast hash functions, these hash functions are locality-sensitive in the sense that two similar objects in the space have high probability of collision, while two very different objects have low probability of collision [Dasgupta et al., 2011].

LSH, for instance, has been used as a tool to find nearest neighbors in an efficient way [Slaney and Casey, 2008] through random projections using dot products, in this RNLA approach the main idea is to select randomly x such that when calculating $h(v) = \lfloor \frac{v \cdot x + b}{w} \rfloor$, the hash function, such that for any two points p, q close to each other, there is a high probability the fall into the same bucket, this is $P_H[h(p) = h(q)] \geq P_1$ for $\|p - q\| \leq R_1$ and when p and q are very far from each other the probability of falling into the same bucket is low, this is $P_H[h(p) = h(q)] \leq P_2$ for $\|p - q\| \geq cR_1 = R_2$, where $R_2 > R_1$ and $\|\|$ is the norm associated to the dot product. Also, as the dot product is a linear operator, the difference between the two image points $\|h(p) - h(q)\|$ is proportional to $\|p - q\|$ then $P_1 > P_2$.

Within this technique, one of the most common approaches is the Distance Based Hashing (DBH) which is a type of LSH for metric spaces, where the main idea is to use exclusively a distance function d between the data in order to create the buckets from the hash function, but with the same approach from all LSH methods of having two closer points with higher probability of being in the same bucket and 2 far points with lower probability of being in distinct buckets [Wang et al., 2014]. This strategy has been used before in order to index image collections using the hashing as a tool for feature combination [Hassan et al., 2012]. This has also been one approach to do fast similarity search [Zhang et al., 2010]. The way DBH strategy solves these problems is by using projections of the form $f(x, a_1, a_2) = \frac{d^2(x,a_1) + d^2(a_1,a_2) - d^2(x,a_2)}{2d(a_1,a_2)}$, which basically are projecting x to the line formed by points a_1 and a_2, in this projection closer points are expected to have similar values, while distant points are expected to have different values. Finally, in order to make a hash function out the projections DBH uses randomization selecting different pairs of points and from the projections use binarization in order to separate the points and create the buckets, i.e. for each pair of points projecting all samples $x \in X$ and selecting a threshold t_1, t_2 such that $h(x, a_1, a_2) = 1$ if $f(x, a_1, a_2) \in [t_1, t_2]$ and $h(x, a_1, a_2) = 0$ otherwise.

Mercer's theorem states that that for any valid kernel function there is a Hilbert space $\mathcal{H}$ associated to it, where that function corresponds to the dot product in $\mathcal{H}$ [Dostanic, 1993]. Therefore, we can use directly the DBH method with a kernel function, using the distance generated by the kernel function, i.e. let $x_1, x_2 \in X$, then $d(x_1, x_2) = \sqrt{\langle \phi(x_1) - \phi(x_2), \phi(x_1) - \phi(x_2) \rangle} = \sqrt{\langle \phi(x_1), \phi(x_1) \rangle - 2\langle \phi(x_1), \phi(x_2) \rangle + \langle \phi(x_2), \phi(x_2) \rangle} = \sqrt{k(x_1, x_1) - 2k(x_1, x_2) + k(x_2, x_2)}$. Note that here we are using the distance in the feature space, which allows us to get the complex non-linear relationships of the data when selecting these landmarks. This approach to select the budget is shown in algorithm 1

Algorithm 1: Kernelized Distance Based Budget

```
   Input    : Training set $X$ and size of budget $b$
   Output   : Budget $B$
1  Select $m = log_2(b)$ pairs of elements from $X$, $(x_{1,1}, x_{1,2}), (x_{2,1}, x_{2,2}), ..., (x_{m,1}, x_{m,2})$
2  $hashcodes[x] = 0$ for all $x \in X$
3  for $i = 1$ to $m$ do
4      $proj = \{\}$
5      for $x$ in $X$ do
6          $proj[x] = k(x, x_{i,1}) - k(x, x_{i,2})$
7      end for
8      Choose the median from $proj$ and split the data in halves
9      Append 0 to $hashcodes$ of the first half and 1 to $hashcodes$ of the second half
10 end for
11 $Budget = \{\}$
12 for $hash$ in $hashcodes$ do
13     Let $X_{hash} \subset X$ the subset of elements with $hashcode[x] = hash$
14     Let $\hat{X}_{hash}$ be $w$ random samples from $X_{hash}$
15     Calculate the medoid $x_{hash}$ of $\hat{X}_{hash}$
16     $Budget = Budget \cup x_{hash}$
17 end for
18 return $Budget$
```

4 Budget on machine learning methods

This chapter presents a budget approximation strategy for kernel methods. The strategy is evaluated on different learning algorithms. In particular, in this chapter we present the formulation of this approximation for the support vector machines and least squares support vector machines, the semi-supervised online kernel matrix factorization and an unsupervised kernel k-medoids.

4.1. Supervised methods

In this section we present the budget formulation for supervised kernel methods: support vector machines (SVM) [Hearst et al., 1998] and least squares support vector machines (LSSVM) [Suykens and Vandewalle, 1999], both are kernel methods with a similar training approach for classification and regression problems, but with differences in their loss functions, therefore in their formulation and optimization derivation.

4.1.1. Support vector machines

Loss formulation

Let $X \in \mathbb{R}^{d\times n}$ be a dataset of n samples $x_1, x_2, ..., x_n$ of dimension d with labels $y_1, y_2, ..., y_n$ respectively, consider the primal formulation for classification in the SVM:

$$\min_w ||w||^2 + C\sum_{i=1}^{n}\xi_i \quad \text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \tag{4-1}$$

We can reformulate (4-1) to:

$$\min_w L(w) \quad \text{where} \quad L(w) = ||w||^2 + C\sum_{i=1}^{n}\max\left(0, 1 - y_i(w^T x_i + b)\right) \tag{4-2}$$

Then, we add a restriction over w, forming it as a linear combination of elements in X, i.e. $w = \sum_{x_j \in X}\alpha_j x_j$, considering this over (4-2) we get;

$$L(w) = L'(\alpha) = ||\sum_{x_j\in X}\alpha_j x_j||^2 + C\sum_{i=1}^{n}\max\left(0, 1 - y_i(\sum_{x_j\in X}\alpha_j x_j^T x_i + b)\right)$$

Now, consider a mapping $\phi : \mathbb{R}^d \to \mathbb{R}^D$ if we map the data through a function $\phi(x)$ we have:

$$L'(\alpha) = ||\sum_{x_j \in X} \alpha_j \phi(x_j)||^2 + C\sum_{i=1}^{n} \text{máx}\,(0, 1 - y_i(\sum_{x_j \in X} \alpha_j \phi(x_j)^T \phi(x_i) + b)) \tag{4-3}$$

As $\mathbb{R}^D$ is a Hilbert space we can use the dot product in (4-3) in the following form:

$$\begin{aligned} L'(\alpha) &= < \sum_{x_j \in X} \alpha_j \phi(x_j), \sum_{x_j \in X} \alpha_j \phi(x_j) > + C\sum_{i=1}^{n} \text{máx}\,(0, 1 - y_i(\sum_{x_j \in X} \alpha_j < \phi(x_j), \phi(x_i) > + b)) \\ &= \sum_{x_i, x_j \in X} \alpha_i \alpha_j < \phi(x_i), \phi(x_j) > + C\sum_{i=1}^{n} \text{máx}\,(0, 1 - y_i(\sum_{x_j \in X} \alpha_j < \phi(x_j), \phi(x_i) > + b)) \end{aligned}$$

Finally, using the kernel trick:

$$\begin{aligned} L'(\alpha) &= \sum_{x_i, x_j \in X} \alpha_i \alpha_j k(x_i, x_j) + C\sum_{i=1}^{n} \text{máx}\,(0, 1 - y_i(\sum_{x_j \in X} \alpha_j k(x_j, x_i) + b)) \\ &= \alpha^T K(X, X)\alpha + C\sum_{i=1}^{n} \text{máx}\,(0, 1 - y_i(\alpha^T K(X, x_i) + b)) \end{aligned} \tag{4-4}$$

Now, let's consider the budget B, which is a subset of samples from X, i.e. $B = [x_{a_1}, x_{a_2}, ..., x_{a_b}]$ for $a_1, a_2, ..., a_b \in \{1, 2, ..., n\}$. For a budgeted version of SVM we make our restriction over w to be a linear combination of B instead of a linear combination of X, i.e. $w = \sum_{x_j \in B} \alpha_j x_j$, with this restriction we get an approximation $\tilde{L'}(\alpha)$ of the loss function $L'(\alpha)$ for the SVM. With the same procedure we used to get to equation (4-4) we obtain the following equation:

$$\tilde{L'}(\hat{\alpha}) = \hat{\alpha}^T K(B, B)\hat{\alpha} + C\sum_{i=1}^{n} \text{máx}\,(0, 1 - y_i(\hat{\alpha}^T K(B, x_i) + b)) \tag{4-5}$$

We have to consider also the prediction function, let $x \in \mathbb{R}^d$, in order to classify x in a class using binary original SVM given w we calculate $f(w, x) = w^T x$, and $pred(w, x) = \begin{cases} \text{-1}, & \text{if } f(w,x) < 0 \\ 1, & \text{otherwise.} \end{cases}$. With the budget restrictions we have then that $w = \sum_{x_j \in B} \alpha_j x_j$, therefore $f(w, x) = \hat{f}(\alpha, x) = \sum_{x_j \in B} \alpha_j x_j^T x = \sum_{x_j \in B} \alpha_j k(x_j, x)$, and the prediction function would remain the same using $\hat{f}$ instead of f. Note that $\hat{f}$ is a randomized numerical linear algebra approximation of f as showed in section 3.

Optimization

Equation (4-5) is the loss function budget formulation for SVM, this is the function we should try to minimize. However, only calculating this function would imply to calculate the prediction function for all points in the dataset, doing this is computationally very expensive considering that it would be done for each epoch in the training phase. Our approach to avoid this is to formulate the loss with regard to a single data point, if this function is differentiable respecting α and b, then we can apply a stochastic gradient descent algorithm in order to do the optimization procedure without the extensive cost of having to evaluate the complete loss function, i.e. avoiding to predict all points in every epoch of the algorithm.

Lets consider $\tilde{L}'(\alpha)$ as stated in equation (4-5), and a given training sample x_i. Considering just that sample from $\tilde{L}'(\alpha)$, we get $\tilde{L}_i(\alpha) = \frac{1}{n}\alpha^T K(B,B)\alpha + C\ \text{máx}\,(0, 1 - y_i(\alpha^T K(B, x_i) + b))$, in particular note that $\tilde{L}'(\alpha) = \sum_{i=1}^{n} \tilde{L}_i(\alpha)$. Observe that $\tilde{L}_i$ is differentiable everywhere except when the second summand of the equation is exactly 0, however this unlikely case can be solved by using a subgradient, obtaining the following:

$$d\tilde{L}_i/d\alpha = \frac{2}{n}\alpha^T K(B,B) - Cy_i K(B,x_i)\delta(x_i) \tag{4-6}$$

In equation (4-6), $\delta(x_i)$ is a piecewise function, which is equal to 0 when $1 - y_i(\alpha^T K(B, x_i) \leq -b$ and 1 otherwise. We also have to consider the intercept b, which we find doing the same optimization procedure, i.e. we calculate the subgradient and apply stochastic gradient descent at the same time that α, with this we obtain the following:

$$d\tilde{L}_i/d\overline{\overline{=}} - Cy_i\delta(x_i) \tag{4-7}$$

With equations (4-6) and (4-7) we can apply directly a stochastic gradient descent algorithm to minimize (4-5) and therefore solving the classification problem with the SVM formulation. Note that from the problem formulation to the optimization process we never needed to calculate the kernel of all points against all points, but only the kernel of all points against the budget, depending on the size of the budget we can do this calculation once and store it in matrices where the calculations described above can be done efficiently using GPU.

Additionally, we can consider calculating the gradient for a small batch of points in the same epoch, summing (4-6) and (4-7) calculated for those points, this is known as mini-batch stochastic gradient descent and it has been shown to speed up optimization processes [Khirirat et al., 2017]. The full training procedure is shown in algorithm 2.

Algorithm 2: Budgeted SVM with mini-batch gradient descent

```
Input  : Training set $X$ with labels $Y$, budget $B$, step training size $\eta$, batch
         size $m$
Output : $\alpha$ vector and $b$ intercept for which the loss function is expected to be
         minimum
1  Let $n \leftarrow |X|$
2  Randomize initizalize values for $\alpha$ and $b$
3  Compute matrix $K(B, X)$
4  while an approximate minimum is not reached do
5      /* We could define the number of epochs apriori                    */
6      Select a random batch $\hat{X}$ from $X$ of size $m$
7      for $x_i$ in $\hat{X}$ do
8          Let $y_i$ be the label of $x_i$
9          $\delta(x_i) \leftarrow \operatorname{sign}(\text{máx}\,(0, 1 - y_i(\alpha^T K(B, x_i) + b)))$
10     end for
11     $\alpha \leftarrow \alpha - \eta\left(\frac{2m}{n}\alpha^T K(B, B) - C\sum_{x_i \in \hat{X}} y_i K(B, x_i)\delta(x_i)\right)$
12     $b \leftarrow b + \eta C \sum_{x_i \in \hat{X}} y_i \delta(x_i)$
13     Decrease the step size $\eta$ for next batch
14 end while
15 return $\alpha$ and $b$
```

4.1.2. Least squares support vector machines

Loss formulation

Let $X \in \mathbb{R}^{d\times n}$ be a dataset of n samples $x_1, x_2, ..., x_n$ of dimension d with labels $y_1, y_2, ..., y_n$ respectively, consider the primal formulation for classification in the LSSVM:

$$\min_{w} \frac{1}{2}(||w||^2 + C\sum_{i=1}^{n} \xi_i^2) \quad \text{subject to} \quad y_i(w^T x_i + b) = 1 - \xi_i \tag{4-8}$$

Once the Lagrangian is defined subject to Kuhn-Tucker conditions, the dual problem arises as a system of equations

$$\begin{bmatrix} K + I_n/\gamma & 1_n \\ 1_n^T & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ b \end{bmatrix} = \begin{bmatrix} y \\ 0 \end{bmatrix} \tag{4-9}$$

where $K_{ij} = k\left(x_i, x_j\right) = \langle \phi(x_i), \phi(x_j)\rangle$ is the kernel matrix, $1_n \in \mathbb{R}^{1\times n}$ is a matrix containing only ones, $\alpha = [\alpha_1, \ldots, \alpha_n]$ is the vector of Lagrange multipliers, $y = [y_1, \ldots, y_n]$, and I_n is the $n \times n$ identity matrix. Once the system is solved for α and b, the model is given by:

$$y\left(x\right) = w^T \phi\left(x\right) + b, \tag{4-10}$$

where $w = \sum_{i=1}^{n} \alpha_i y_i \phi(x_i)$.

For the dual version, we take the Lagrangian of the original LS-SVM problem 4-8

$$\mathcal{L}\left(w, b, e, \alpha\right) = J\left(w, b, e\right) - \sum_{k=1}^{n} \alpha_k \left(y_k \left[w^T \varphi\left(x_k\right) + b\right] - 1 + e_k\right) \tag{4-11}$$

subject to $w = \sum_{k=1}^{n} \alpha_k y_k \varphi\left(x_i\right)$, $\sum_{k=1}^{n} \alpha_k y_k = 0$, and $\alpha_k = \gamma e_k$, $y_k \left[w^T \varphi\left(x_k\right) + b\right] - 1 + e_k = 0$ for $k = 1, \ldots n$. Plugging this into equation 4-11, we get the dual problem

$$\mathcal{L}\left(w, b, e, \alpha\right) = -\frac{1}{2}\left(\alpha y\right)^T k\left(X, X\right)\left(\alpha y\right) + \sum_{k=1}^{n} \alpha_k - \frac{C}{2}\sum_{k=1}^{n}\left(1 - y_k\left[\left(\alpha y\right)^T k\left(X, x_k\right) + b\right]\right)^2 \tag{4-12}$$

where (αy) represents a pairwise product of α and y, and must be maximized for α_k, $k = 1, \ldots, n$, and b.

Budget strategy can be implemented in LS-SVM as follows: instead of computing the entire kernel matrix, a selection of $\beta \ll n$ instances will be made, selecting a sub-matrix B from the input data matrix X to train the machine. From 4-12 we get that the loss function will be

$$\min_{\alpha,b} \mathcal{L}' = \frac{1}{2} (\alpha y)^T k(B,B)(\alpha y) - \sum_{k=1}^{\beta} \alpha_k + \frac{C}{2} \sum_{k=1}^{n} \Big(1 - y_k \Big[(\alpha y)^T k(B, x_k) + b\Big]\Big)^2 \tag{4-13}$$

Optimization

SGD permits an online implementation as it updates the solution using a single training sample at a time, which alleviates even more the memory requirements. Following this, given the derivatives

$$\frac{\partial \mathcal{L}'}{\partial \alpha_i} = \sum_{k=1}^{\beta} \alpha_k y_k y_i k(x_i, x_k) - 1 - C \sum_{k=1}^{n} \Big(1 - y_k \Big[(\alpha y)^T k(B, x_k) + b\Big]\Big) y_k y_i k(x_i, x_k) \tag{4-14}$$

the update rule is given by

$$\begin{aligned} \alpha_m = \alpha_m &- \eta y_m (\alpha y)^T k(B, x_m) + \eta \\ &+ \eta C n \left(1 - y_j \left[(\alpha y)^T k(B, x_j) + b\right]\right) y_j y_m k(x_j, x_m) \end{aligned} \tag{4-15}$$

where (x_j, y_j) is a randomly chosen instance of X. The entire procedure of the Online Budgeted LS-SVM is described in algorithm 3.

4.2. Semi-supervised methods

In this section we present the budget formulation for the SSOKMF [Vanegas et al., 2018], which is a method to perform semantic embedding trained in a semi-supervised fashion, therefore it does not need to have labels for all inputs but only for some in order to perform a classification task.

4.2.1. SSOKMF

Loss formulation

For a dataset $X \in \mathbb{R}^{n\times m}$ we can find a low-level semantic representation by factorization, i.e. finding matrices $F \in \mathbb{R}^{n\times r}$ and $H \in \mathbb{R}^{r\times m}$ with $r \ll n$ such that $X \approx FH$. In order to find non-linear relationships let $\phi : \mathbb{R}^n \to F$ a high dimensional mapping with inner product $\langle , \rangle_F$, moreover with kernel k, now we would have $\phi(X) \approx F_\phi H$, which may be unfortunately infeasible to calculate due to the high dimensionality of ϕ. Therefore, instead of calculating F_ϕ, let's add the restriction that F_ϕ is composed by linear combination of the points of X in the feature space, i.e. $F_\phi = \phi(X) W_x$, then $\phi(X) \approx \phi(X) W_x H$ [Vanegas et al., 2018].

Algorithm 3: Online budgeted LS-SVM

```
Input  : Training set $X$ with labels $Y$, budget $B$, step training size $\eta$, batch
         size $m$
Output : $\alpha$ vector and $b$ intercept for which the loss function is expected to be
         minimum
1  Let $n \leftarrow |X|$
2  Randomize initizalize values for $\alpha$ and $b$
3  Compute matrix $K(B, X)$
4  while an approximate minimum is not reached do
5      /* We could define the number of epochs apriori                    */
6      Randomly shuffle $X$
7      for $j$ in $\{1, ..., n\}$ do
8          for $m = 1$ to $\beta$ do
9              $\alpha_m = \alpha_m - \eta y_m (\alpha y)^T k(B, x_m) + \eta + \eta C n \left(1 - y_j \left[(\alpha y)^T k(B, x_j) + b\right]\right) y_j y_m k(x_j, x_m),$
10         end for
11         $b = b - \eta C n \left(1 - y_j \left[(\alpha y)^T k(B, x_j) + b\right] (-y_j)\right)$
12     end for
13     Decrease the step size $\eta$ for next batch
14 end while
15 return $\alpha$ and $b$
```

In this setup we can use representative points from the basis matrix, i.e. a budget $B \in \mathbb{R}^{b\times m}$ instead of the full matrix. Therefore, the formulation of the budgeted matrix factorization algorithm is to find matrices W_x and H such that $\phi(X) \approx \phi(B)W_xH$.

Additionally, we have only t annotated instances, let $Y \in \mathbb{R}^{c\times t}$ be the one hot encoding of the t annotated examples into the c corresponding classes, this leads us to a second factorization problem, finding $W_y \in \mathbb{R}^{c\times r}$ and $H_t \in \mathbb{R}^{r\times t}$ such that $Y \approx W_yH_t$, where H_t is the subset of H composed from the annotated instances of the dataset. Considering all previous conditions, the general problem is to minimize the following loss function:

$$J(W_x, W_y, H) = \frac{\alpha}{2}\|\phi(X) - \phi(B)W_xH\|^2 + \frac{\beta}{2}\|Y - W_yH_t\|^2 + \frac{\lambda_1}{2}\|W_x\|^2 + \frac{\lambda_2}{2}\|W_y\|^2 + \frac{\lambda_3}{2}\|H\|^2 \tag{4-16}$$

Optimization

In order to calculate the optimal values in an efficient way for large datasets, an online SGD can be formulated from equation 4-16 taking only single points as follows:

$$J(W_x, W_y, h_i) = \frac{\alpha}{2}\|\phi(x_i) - \phi(B)W_xh_i\|^2 + \frac{\beta}{2}\|Y - W_yh_i\|^2 + \frac{\lambda_1}{2}\|W_x\|^2 + \frac{\lambda_2}{2}\|W_y\|^2 + \frac{\lambda_3}{2}\|h_i\|^2 \tag{4-17}$$

and the corresponding gradients:

$$\frac{\delta J_i(W_x, W_y, h_i)}{\delta W_x} = \alpha\phi(B)^T\phi(x_i)h_i^T + \alpha\phi(B)^T\phi(B)W_xh_ih_i^T + \lambda_1 W_x \tag{4-18}$$

$$\frac{\delta J_i(W_x, W_y, h_i)}{\delta W_y} = \beta y_ih_i^T - \beta W_yh_ih_i^T + \lambda_2 W_y \tag{4-19}$$

Although evaluating directly 4-18 may be infeasible, because ϕ may be a very high or even infinite dimensional mapping. However, using kernel trick we have that:

$$\frac{\delta J_i(W_x, W_y, h_i)}{\delta W_x} = \alpha k(B, x_i)h_i^T + \alpha k(B, B)W_xh_ih_i^T + \lambda_1 W_x \tag{4-20}$$

4.3. Unsupervised methods

In this section we present the budget formulation of a kernel K-Medoids algorithm for clustering, this is an adaptation of classic K-Means algorithm, which defines clusters depending on the distance of each data sample to a representative point for each cluster, which in this case is a medoid. For this algorithm we used the notion of distance inherited by the kernel function and therefore by the high dimensional feature space, data does not need to be labeled for this, thus it is an unsupervised method.

4.3.1. Kernel k-medoids

Formulation

Let X be a dataset, and k a kernel function for X. Due to Mercer's theorem we know that there is a reproducing kernel hilbert space H and a map ϕ such that for each $x_1, x_2 \in X$, $k(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle_H$, thanks to this theorem we get a corresponding distance d in H as the following:

$$\begin{aligned} d(x_1, x_2) &= \sqrt{\langle \phi(x_1) - \phi(x_2), \phi(x_1) - \phi(x_2) \rangle_H} \\ &= \sqrt{\langle \phi(x_1), \phi(x_1) \rangle_H - 2\langle \phi(x_1), \phi(x_2) \rangle_H + \langle \phi(x_2), \phi(x_2) \rangle_H} \\ &= \sqrt{k(x_1, x_1) - 2k(x_1, x_2) + k(x_2, x_2)} \end{aligned} \tag{4-21}$$

Just with the distance defined in 4-21 it is possible to formulate a clustering based on a subset of points $\tilde{X}$. Formally let $\tilde{X} = \{\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_k\} \subset X$ be k different samples from the dataset we can define the i-th cluster as the following:

$$C_i = \{x \in X \mid i = \text{argmin}_{j=1,\ldots,k} d(x, \tilde{x}_j)\} \tag{4-22}$$

In this case or clustering method aims to find the best subset of points $\tilde{X}$ such that the clusters in 4-21 define some relationship on the real data, this has been shown to be a NP-Hard problem, therefore our approach is to use the Lloyds algorithm heuristic, which consists in selecting a subset of points randomly and by iterations changing the points calculating the clusters and the medoid $\tilde{x}_i$ of each cluster.

$$\tilde{x}_i = \text{argmin}_{x \in X} \sum_{y \in C_i} d(x, y) \tag{4-23}$$

Note that for this exact approach we would need the kernel matrix K as we would have to calculate the distance between each pair of points in X. Which means the above algorithm is $\Omega(n^2)$, which is infeasible. Our budgeted approach for this case consists in restricting the calculation of the medoid not taking the sum of all distances to the points in the cluster but also the points in the budget B as shown in 4-24.

$$\tilde{x}_i = \text{argmin}_{x \in X} \sum_{y \in C_i, y \in B} d(x, y) \tag{4-24}$$

Using the budget formulation here we not only reduce drastically the computational complexity, but even with random budgets we are able to get medoids very close to the medoid we would get with the exact version [Wang, 2006], therefore obtaining a good approximation. The entire prodceure of the Budgeted kernel K-medoids is described in algorithm 4.

Algorithm 4: Kernel budgeted k-Medoids

```
   Input   : Training set $X$, budget $B$ and number of clusters $k$
   Output  : Clustered dataset
1  Let $n \leftarrow |X|$
2  Randomly select $k$ elements from $X$ as the medoids $\hat{x}_1, ..., \hat{x}_k$
3  Compute matrix $K(B, X)$ and the diagonal of $K(X, X)$
4  while new medoids are calculated do
5      /* We could define the number of epochs apriori */
6      for $x_i$ in $B$ do
7          Set $cluster_i = argmin_j\{k(x_i, x_i) - 2k(x_i, \hat{x}_j) + k(\hat{x}_j, \hat{x}_j) \mid j \in \{1, 2, ..., k\}\}$
8      end for
9      for $j = 1$ to $k$ do
10         Set $\hat{x}_j = argmin_{x \in X}\{\sum_{x_i \in B,\ cluster_i = j} k(x, x) - 2k(x, x_i) + k(k_i, k_i)\}$
11     end for
12 end while
13 $Y = \{y_i \mid y_i = argmin_j\{k(x_i, x_i) - 2k(x_i, \hat{x}_j) + k(\hat{x}_j, \hat{x}_j) \mid j \in \{1, 2, ..., k\}\}$ for $x_i \in X\}$
   return $Y$
```

5 Experimental evaluation

This chapter presents the experimental evaluation of the algorithms using the learning-on-a-budget strategy, including the setup of the experiments and some discussion about the results towards at end of the chapter.

5.1. Experimental setup

For the experimental setup we trained four different algorithms: support vector machines (SVM), least squares support vector machines (LSSVM), semi-supervised online kernel matrix factorization (SSOKMF) and budgeted kernelized k-medoids. All these algorithms were first implemented in Python 3 and then were trained 20 times in each of the given configurations using a processor AMD Ryzen 5 2500U with Radeon Vega Mobile Gfx 2.00 GHz with 4 gigabytes of RAM. For each of the models we evaluate the performance using budgets corresponding approximately to the 0,015 %, 0,03 %, 0,06 %, 0,12 %, 0,25 %, 0,5 %, 1 % and 2 % of the datasets. Larger approximations were not considered because of the current restrictions of the kernel method for large datasets which would have limited the quantity of trained models due to time constrains. Also, it is more relevant for this work to evaluate the performance when reducing significantly the size of the budget instead of using something proportionally close to the actual size of the dataset. Finally, we also considered different kernel functions in each of the configurations: linear, polynomial using a second grade polynomial, radial basis function (RBF) and chi^2 kernels.

5.1.1. Datasets

In the experimental evaluation we considered two datasets: MNIST [LeCun and Cortes, 2010] and Fashion MNIST [Xiao et al., 2017], which are datasets with 60000 training samples each and 10000 testing samples. MNIST dataset consists of handwritten digits normalized to 28×28 grayscale pixels, meaning for this dataset we have matrices of size 28×28 with values between 0 and 255, for the training setup as none of our models use convolutional filters we trained over arrays of size 784 instead of matrices and we normalized the values of each pixel to be between 0 and 1, all data is labeled with one of 10 clases representing the handwritten digit. Fashion MNIST dataset consists of images of pieces of clothes converted to grayscale images of 28×28 pixels, as their characteristics are similar as the ones from

MNIST we made the same normalization process, Fashion MNIST data is also labeled in 10 different classes.

5.1.2. Hyperparameter tuning

To train all different models various hyperparameters were tuned in order to improve those corresponding models, for SVM and LSSVM algorithms a hyperparameter C to handle regularization was defined as well as parameters γ for RBF and chi^2 kernels, in order to choose the values for those parameters we used a cross validation approach with one instance of training, for C the values explored were: 0.001, 0.01, 0.1, 1.0, 10.0 and 100.0, additionally for the instances with RBF and chi^2 the *gamma* values explored were: 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2 and 0.5. For SSKOMF we have 3 regularization hyperparameters λ_1, λ_2 and λ_3, and also parameters α and β for trade-off of the supervised and unsupervised parts of this specific model, in this case we made the cross validation exploring all combinations of parameters considering the values: 0.1, 1.0 and 10.0. Additionally, for RBF and chi^2 kernel instances the *gamma* was tuned in a similar manner to the supervised methods. Our approach of kernel k-medoids didn't have any hyperparameters to be tuned. Finally, in all the models we tuned the η parameter regarding the training step impact, which was cross validated with the values 0.0001, 0.001, 0.1 and 1.0.

5.1.3. Performance metrics

In the case of the supervised and semi-supervised methods (SVM, LSSVM and SSOKMF) we trained a multi-class model for a classification task and evaluated the accuracy when calculating to which class each sample in the test set belongs, having results varying from 0 % to 100 % of accuracy. Additionally, as all of the methods presented here are randomized approximations, in order to have strong results statistically, models were trained 20 times and the mean and standard deviation of the accuracy is presented for each model.

In the case of the unsupervised method (kernel k-medoids), different metrics were evaluated for the clustering results, in particular we used: Silhouette score, which attempts to describe the similarity of points with other elements from the same cluster relative to samples not in that cluster. This metric can range from -1 to 1. Fowlkes score, which uses the geometric mean between precision and recall using the original labels. This metric is bounded between 0 and 1. Adjusted Rand index, which try to express what proportion of the cluster assignments are right using the original labels. This metric is bounded between -1 and 2. Calinski Harabaz index, which is the radio of the variance of a datapoint compared to points in other clusters. This metric is not bounded. All these metrics were evaluated using from the scikit-learn library [Pedregosa et al., 2011].

Kernel	Method	10	20	40	80	160	320	640	1280
Linear	Random	47.033 ± 8.44059	69.751 ± 8.77415	80.885 ± 1.96237	85.039 ± 0.67971	86.631 ± 0.86826	86.988 ± 1.42925	87.614 ± 0.73597	86.725 ± 2.03073
	DBH	53.289 ± 6.69605	76.078 ± 5.43888	83.358 ± 0.92505	85.646 ± 0.90593	86.691 ± 0.58418	86.919 ± 1.26139	86.946 ± 1.92266	86.589 ± 1.77896
Poly	Random	46.032 ± 4.55359	57.663 ± 3.04513	67.066 ± 1.93720	73.256 ± 1.29606	76.788 ± 0.63313	78.370 ± 0.53820	79.277 ± 0.35560	79.520 ± 0.29833
	DBH	50.910 ± 3.02336	61.059 ± 2.11795	68.757 ± 1.69689	73.818 ± 1.00143	76.391 ± 0.94982	77.793 ± 0.79505	78.262 ± 0.59503	78.811 ± 0.52820
RBF	Random	58.006 ± 5.66718	70.170 ± 3.37027	77.169 ± 1.28402	81.728 ± 0.53167	84.321 ± 0.35351	86.006 ± 0.22201	87.151 ± 0.18986	87.801 ± 0.15553
	DBH	62.042 ± 2.77467	73.558 ± 1.40058	78.920 ± 0.78610	82.193 ± 0.48172	84.466 ± 0.33751	85.914 ± 0.30893	86.959 ± 0.26793	87.463 ± 0.25890
Chi^2	Random	62.536 ± 4.11211	67.214 ± 1.88904	71.550 ± 0.47929	73.417 ± 0.33988	75.391 ± 0.24631	77.369 ± 0.20460	79.014 ± 0.15859	80.304 ± 0.17376
	DBH	67.822 ± 1.59739	70.523 ± 1.03870	72.171 ± 0.51518	73.803 ± 0.71423	75.301 ± 0.43378	77.236 ± 0.56235	78.936 ± 0.27150	80.320 ± 0.26102

Table 5-1: Budgeted SVM Algorithm for MNIST

5.2. Results

This section presents the obtained results of the trained models evaluated with the test datasets. The first tables **5-1** and **5-2** and corresponding figures **5-1** and **5-2** show the mean and standard deviation of accuracy obtained by the SVM for all different kernels and budget sizes for the MNIST and Fashion datasets respectively. The next tables **5-3** and **5-4** and corresponding figures **5-4** and **5-5** show the results of LSSVM for MNIST and Fashion datasets respectively. As it is a supervised method like SVM, the results are presented in a similar manner. The following are tables **5-5** and **5-6** and corresponding figures **5-7** and **5-8**, which show the results from SSOKMF method for MNIST and Fashion respectively. Even though this is a semi-supervised method as we used the model for a classification task, the results are presented in the same way as SVM and LSSVM. Finally, table **5-7** shows the results obtained by the unsupervised clustering task performed by kernel k-medoids in both datasets. For this dataset we only considered the linear and polynomial kernels.

Additionally, training times are also reported for all methods: figure **5-3** shows training times for SVM, figure **5-6** shows training times for LSSVM, figure **5-9** shows times for SSKOMF and figure **5-10** shows training times for the kernel k-medoids. As both trained datasets have the exact same data characteristics, we only report the average training time for the MNIST dataset using linear and polynomial kernel functions.

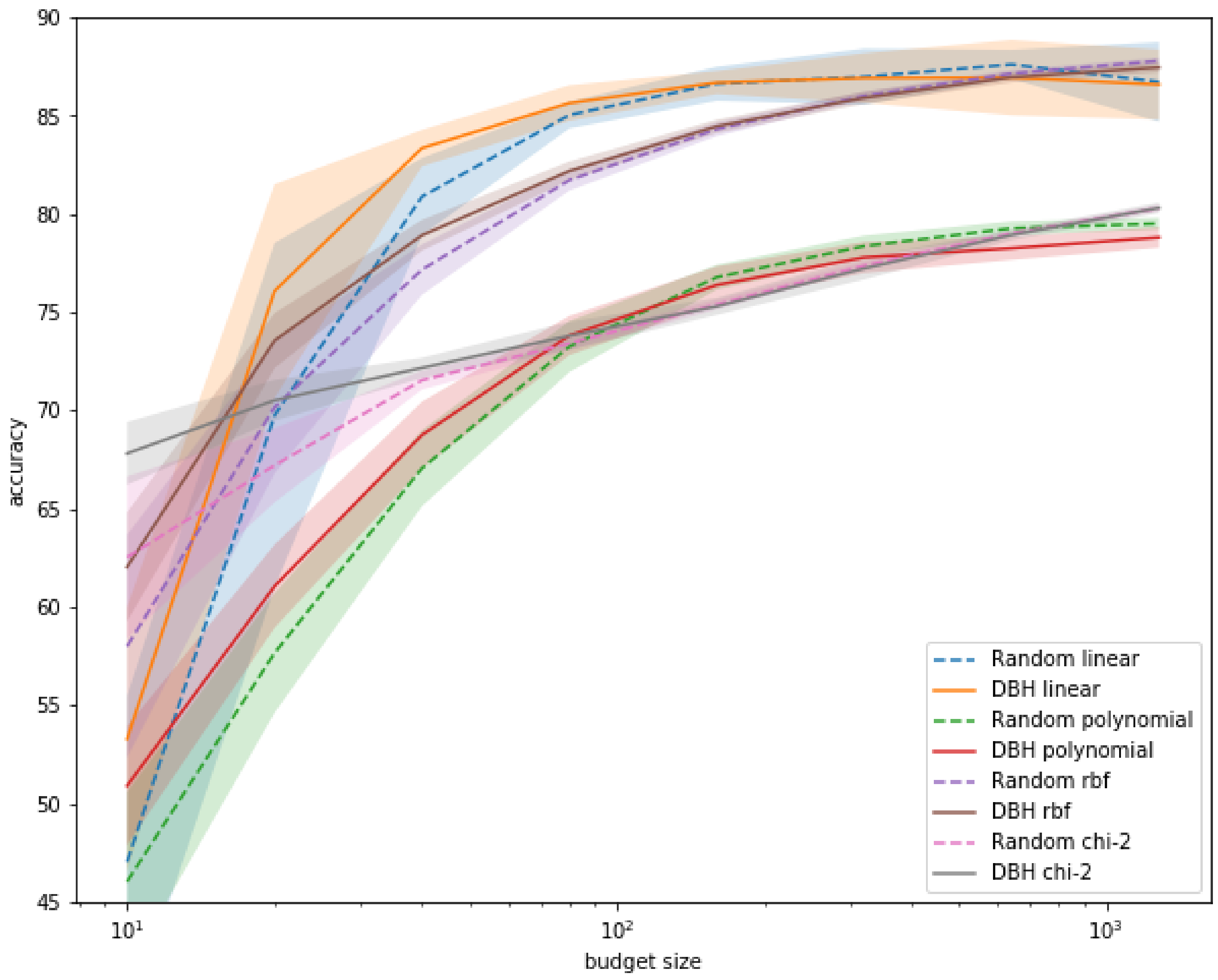

Figure 5-1: Budgeted SVM Algorithm for MNIST

Kernel	Method	10	20	40	80	160	320	640	1280
Linear	Random	57.577 ± 8.66467	65.577 ± 5.98915	71.506 ± 4.12655	71.568 ± 4.45038	72.535 ± 2.60871	70.316 ± 5.39309	72.410 ± 5.29541	72.399 ± 4.08543
	DBH	64.454 ± 5.42968	70.542 ± 3.61247	73.300 ± 2.54681	74.053 ± 2.92980	74.782 ± 2.47078	74.958 ± 2.85269	74.493 ± 2.50493	73.172 ± 2.75969
Poly	Random	50.356 ± 2.12367	55.935 ± 2.64582	61.783 ± 1.76181	66.358 ± 1.31680	69.583 ± 0.55404	70.333 ± 0.36570	70.474 ± 0.60754	70.063 ± 0.79049
	DBH	53.056 ± 2.05468	58.368 ± 1.8819	62.302 ± 1.95277	65.904 ± 1.75847	68.182 ± 0.80219	69.160 ± 0.77060	69.214 ± 0.67978	69.100 ± 0.66704
RBF	Random	57.523 ± 3.91910	65.836 ± 1.35847	69.832 ± 0.66214	71.789 ± 0.30759	73.386 ± 0.33878	75.203 ± 0.27511	77.066 ± 0.19332	78.361 ± 0.46449
	DBH	63.187 ± 4.80881	68.275 ± 0.85820	69.947 ± 0.74203	71.579 ± 0.46720	73.041 ± 0.46958	75.094 ± 0.49570	76.931 ± 0.42412	78.246 ± 0.54901
Chi^2	Random	59.511 ± 2.90397	67.214 ± 1.91779	71.047 ± 1.01897	73.434 ± 0.34260	75.205 ± 0.34248	77.413 ± 0.22866	79.056 ± 0.21167	80.342 ± 0.13485
	DBH	62.338 ± 1.59739	70.756 ± 1.03870	72.493 ± 0.51518	73.617 ± 0.71423	75.475 ± 0.43378	77.442 ± 0.56235	79.012 ± 0.27150	80.142 ± 0.26102

Table 5-2: Budgeted SVM Algorithm for Fashion

Kernel	Method	10	20	40	80	160	320	640	1280
Linear	Random	57.980 ± 6.61510	74.123 ± 4.69095	79.826 ± 2.81704	81.578 ± 1.60975	82.461 ± 1.48808	82.708 ± 1.44367	82.740 ± 1.37434	82.795 ± 1.28959
	DBH	60.986 ± 4.84497	77.715 ± 3.86827	79.835 ± 2.76646	81.338 ± 1.42225	81.868 ± 1.30632	82.192 ± 0.92663	82.455 ± 0.86978	82.638 ± 0.84875
Poly	Random	44.739 ± 5.38853	50.936 ± 4.74669	55.822 ± 4.40795	59.399 ± 3.34163	66.341 ± 3.17897	71.150 ± 2.72277	75.004 ± 2.33068	77.884 ± 2.00109
	DBH	48.681 ± 3.06171	54.278 ± 2.096230	60.271 ± 2.733723	65.065 ± 3.168820	70.884 ± 3.095702	74.778 ± 2.68382	77.477 ± 1.96580	79.738 ± 1.87232
RBF	Random	48.774 ± 5.80612	52.111 ± 3.61013	55.080 ± 2.97762	59.005 ± 2.06814	61.581 ± 1.46006	63.881 ± 1.03247	67.492 ± 0.74602	71.533 ± 0.52878
	DBH	50.578 ± 3.20372	54.643 ± 1.96905	57.002 ± 1.89325	60.628 ± 1.73879	62.805 ± 0.989987	64.066 ± 0.985902	67.100 ± 1.19687	71.368 ± 1.36538
Chi^2	Random	59.648 ± 3.93757	63.182 ± 2.52579	66.643 ± 1.51689	67.717 ± 1.70601	69.240 ± 0.90276	70.422 ± 0.58853	72.141 ± 0.48955	76.646 ± 0.336787
	DBH	62.631 ± 2.23176	65.949 ± 1.70108	67.280 ± 1.43601	68.259 ± 1.45519	70.859 ± 1.94908	71.955 ± 1.24660	73.742 ± 1.22127	77.106 ± 1.20294

Table 5-3: Online Budgeted LSSVM Algorithm for MNIST

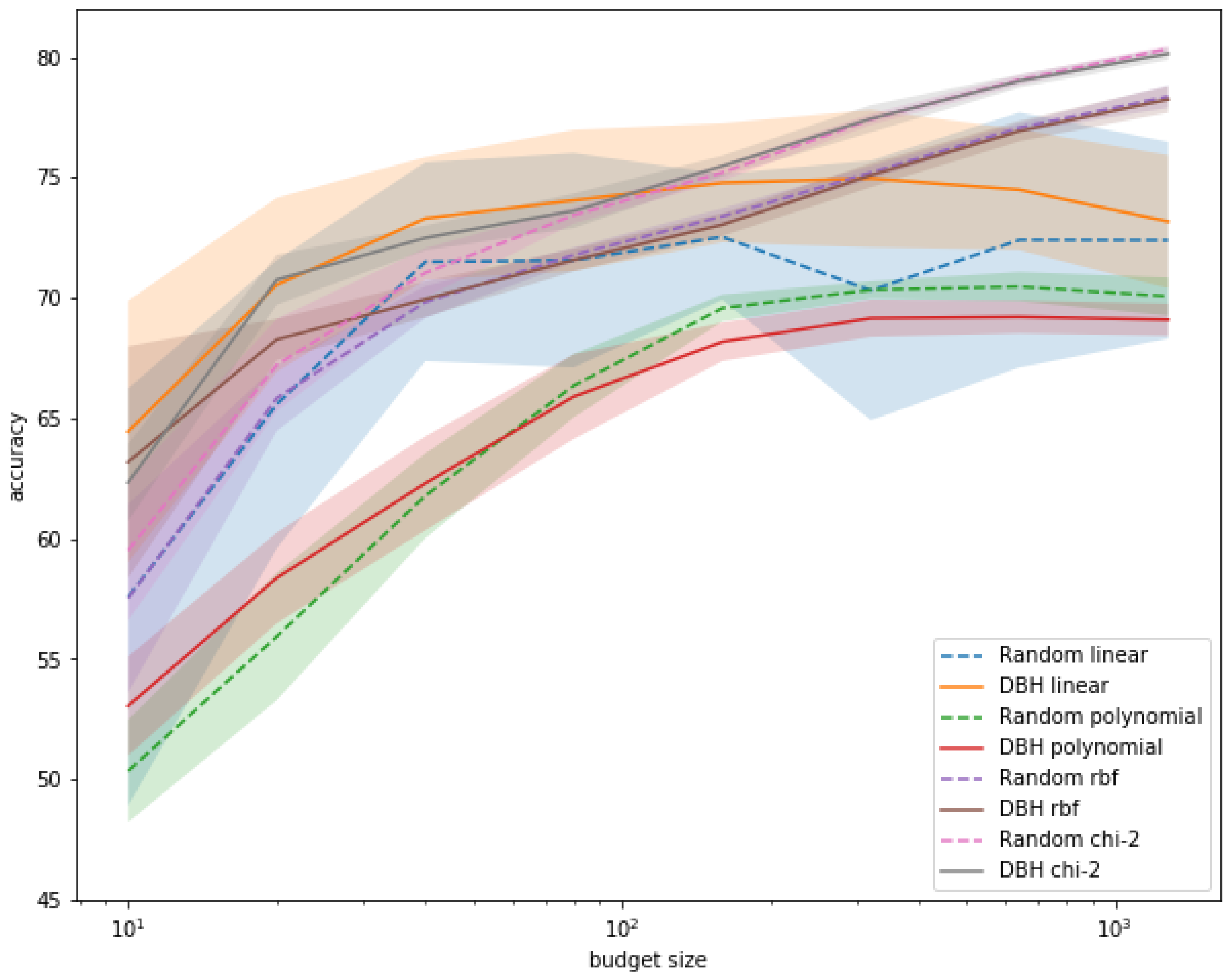

Figure 5-2: Budgeted SVM Algorithm for Fashion

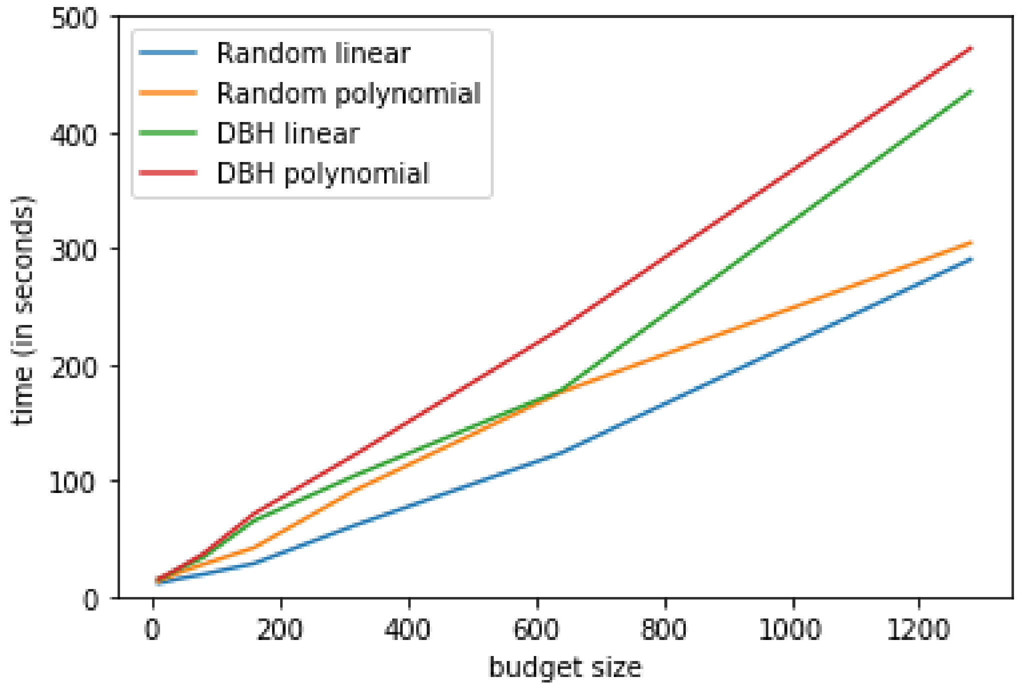

Figure 5-3: Training times SVM algorithm

Kernel	Method	10	20	40	80	160	320	640	1280
Linear	Random	57.195 ± 2.97334	60.624 ± 1.24949	62.327 ± 1.132345	63.126 ± 0.70914	63.617 ± 0.42146	63.649 ± 0.33304	63.618 ± 0.29372	63.753 ± 0.15454
	DBH	59.933 ± 1.37877	61.582 ± 1.05770	62.623 ± 1.26895	62.985 ± 0.70494	63.124 ± 0.47813	63.506 ± 0.45067	63.772 ± 0.41474	63.690 ± 0.29510
Poly	Random	46.656 ± 9.99795	55.806 ± 2.79922	57.947 ± 1.68461	59.227 ± 1.32042	60.571 ± 0.83247	61.499 ± 0.59852	61.862 ± 0.48245	62.706 ± 0.37504
	DBH	54.020 ± 2.30761	57.011 ± 2.49404	58.130 ± 1.85519	59.802 ± 1.56119	60.619 ± 1.49700	61.378 ± 1.44657	62.399 ± 1.65191	64.145 ± 1.49632
RBF	Random	56.172 ± 3.29632	61.123 ± 2.80570	62.176 ± 2.45608	63.663 ± 1.33115	64.457 ± 1.47146	64.730 ± 0.69578	65.517 ± 0.63754	66.291 ± 0.50241
	DBH	60.647 ± 2.98784	62.143 ± 2.40106	63.939 ± 1.93008	64.534 ± 1.24422	65.295 ± 1.31678	66.123 ± 0.95803	67.559 ± 0.89460	68.586 ± 0.47043
Chi^2	Random	59.343 ± 4.72012	62.683 ± 3.00656	64.009 ± 1.51823	68.145 ± 1.20718	70.547 ± 1.15149	71.396 ± 0.69208	72.204 ± 0.44652	74.753 ± 0.31088
	DBH	62.239 ± 3.12781	65.441 ± 2.79143	67.929 ± 2.17742	69.695 ± 1.85360	70.608 ± 1.30654	71.347 ± 1.05140	72.769 ± 0.99364	74.264 ± 0.68086

Table 5-4: Online Budgeted LSSVM Algorithm for Fashion

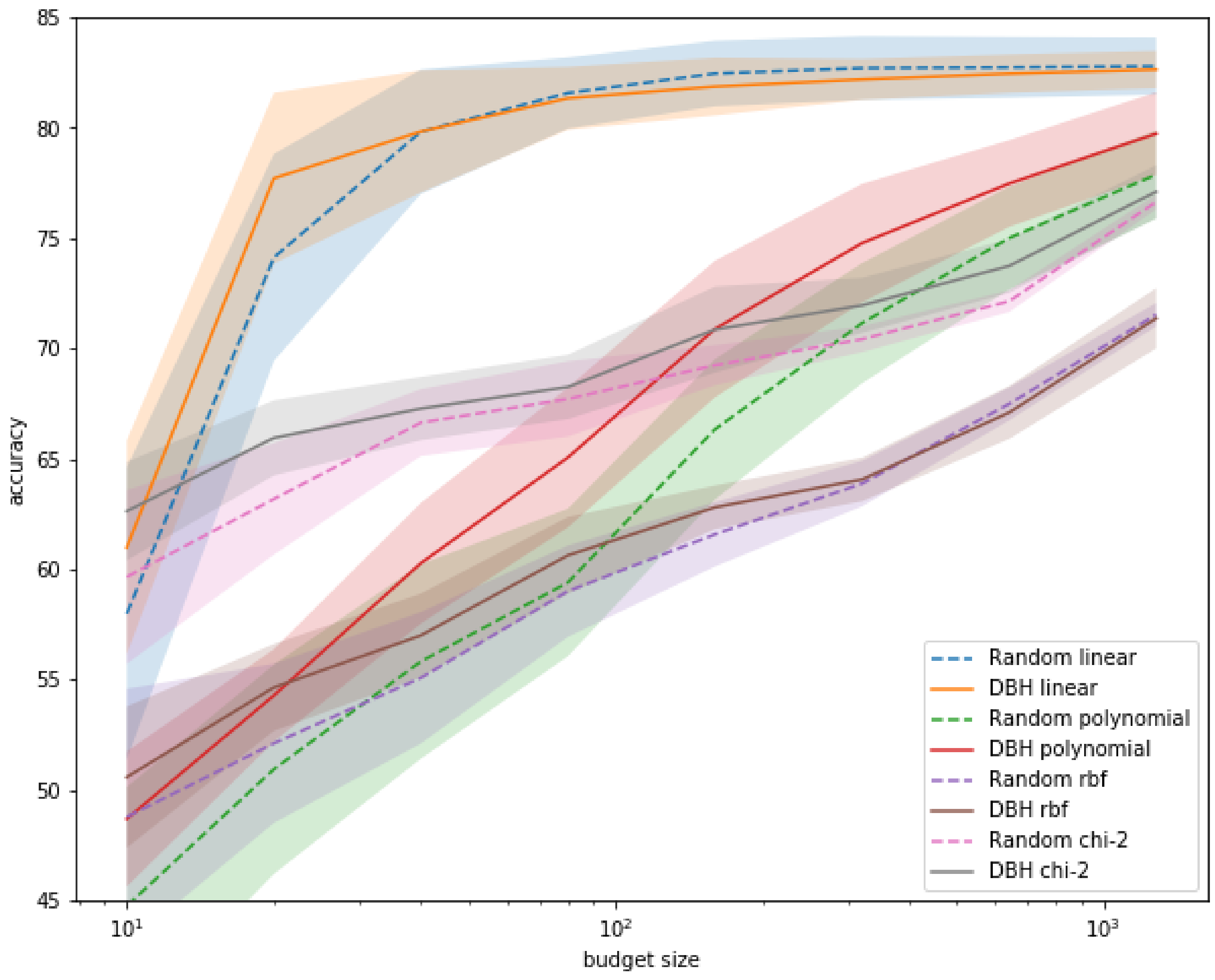

Figure 5-4: Budgeted LSSVM Algorithm for MNIST

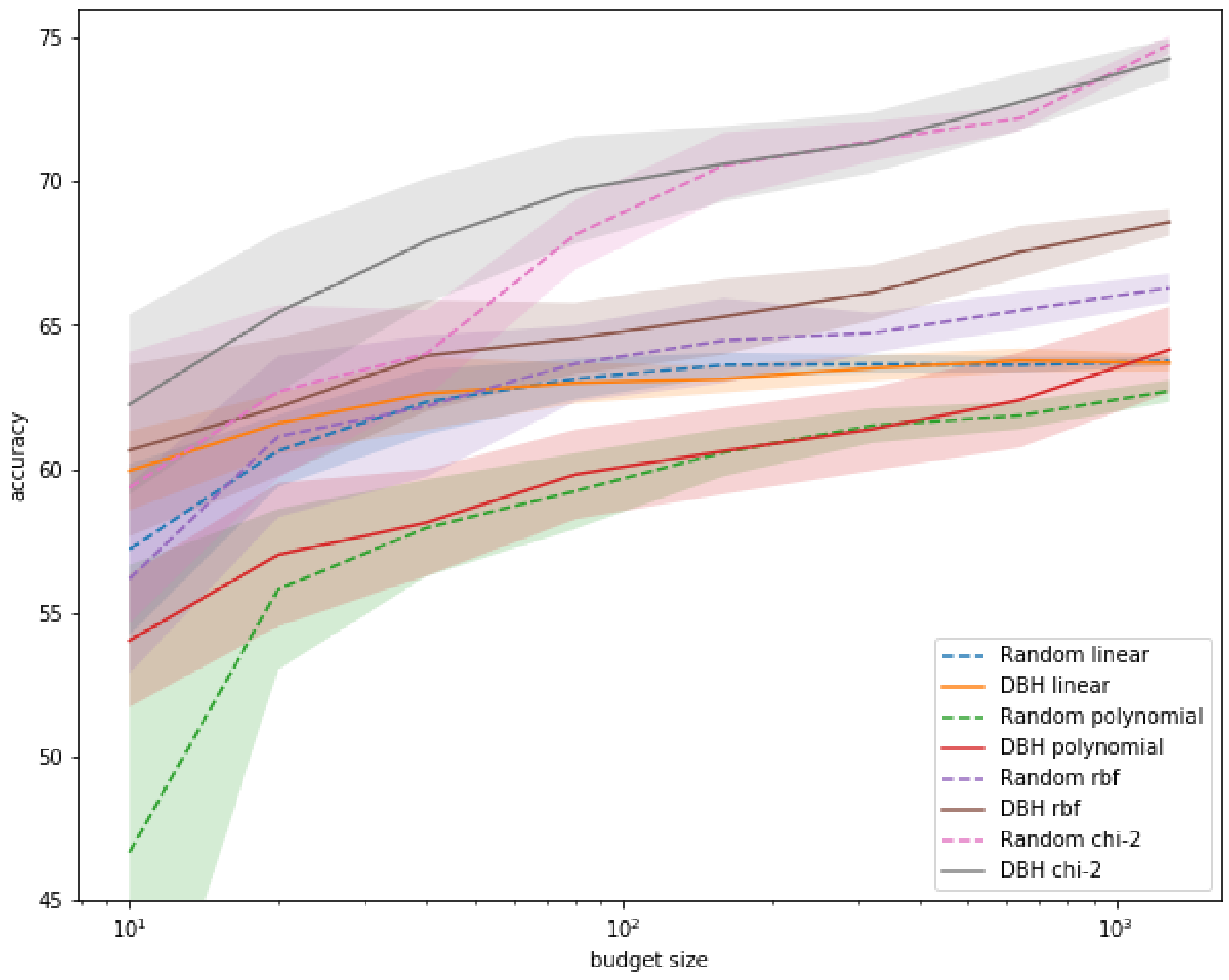

Figure 5-5: Budgeted LSSVM Algorithm for Fashion

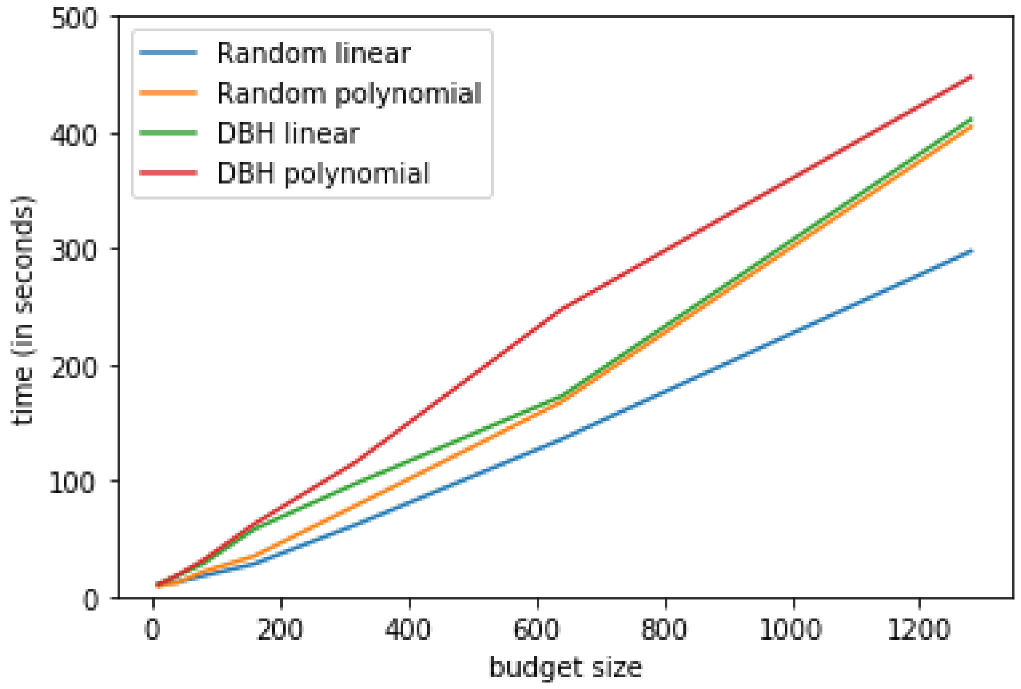

Figure 5-6: Training times LSSVM algorithm

Kernel	Method	10	20	40	80	160	320	640	1280
Linear	Random	58.625 ± 4.16351	73.844 ± 1.88951	77.362 ± 0.91664	79.285 ± 0.61925	79.991 ± 0.15936	79.943 ± 0.11017	79.776 ± 0.13735	79.012 ± 0.15897
	DBH	62.995 ± 2.15931	74.662 ± 1.34927	77.054 ± 0.62542	77.869 ± 0.49104	77.938 ± 0.19461	77.920 ± 0.15293	77.981 ± 0.10017	78.072 ± 0.09318
Poly	Random	56.261 ± 3.94501	70.544 ± 1.93993	76.930 ± 0.89693	77.413 ± 0.85116	77.908 ± 0.69003	77.691 ± 0.70221	77.083 ± 0.37333	77.805 ± 0.18704
	DBH	60.961 ± 2.34052	75.866 ± 0.68589	77.809 ± 0.44547	77.897 ± 0.31112	78.282 ± 0.12728	78.594 ± 0.33941	78.626 ± 0.32527	78.589 ± 0.70004
RBF	Random	55.807 ± 5.85485	68.502 ± 1.86517	76.483 ± 1.03019	80.047 ± 0.74205	80.937 ± 0.56564	81.105 ± 0.75660	81.413 ± 0.30559	81.523 ± 0.26592
	DBH	63.913 ± 3.92520	73.521 ± 1.08187	78.649 ± 0.67882	80.228 ± 0.31113	80.989 ± 0.09192	81.177 ± 0.46891	81.337 ± 0.29698	81.583 ± 0.00707
Chi^2	Random	32.418 ± 3.88452	41.083 ± 4.38922	53.390 ± 2.88561	62.884 ± 2.81430	70.653 ± 2.58641	77.796 ± 1.29752	83.548 ± 0.61410	87.109 ± 0.55168
	DBH	46.752 ± 2.68701	54.532 ± 1.49422	60.051 ± 0.96689	70.756 ± 1.02642	76.314 ± 0.57796	80.066 ± 1.22738	85.397 ± 1.03844	88.749 ± 0.26870

Table 5-5: SSOKMF Algorithm for MNIST

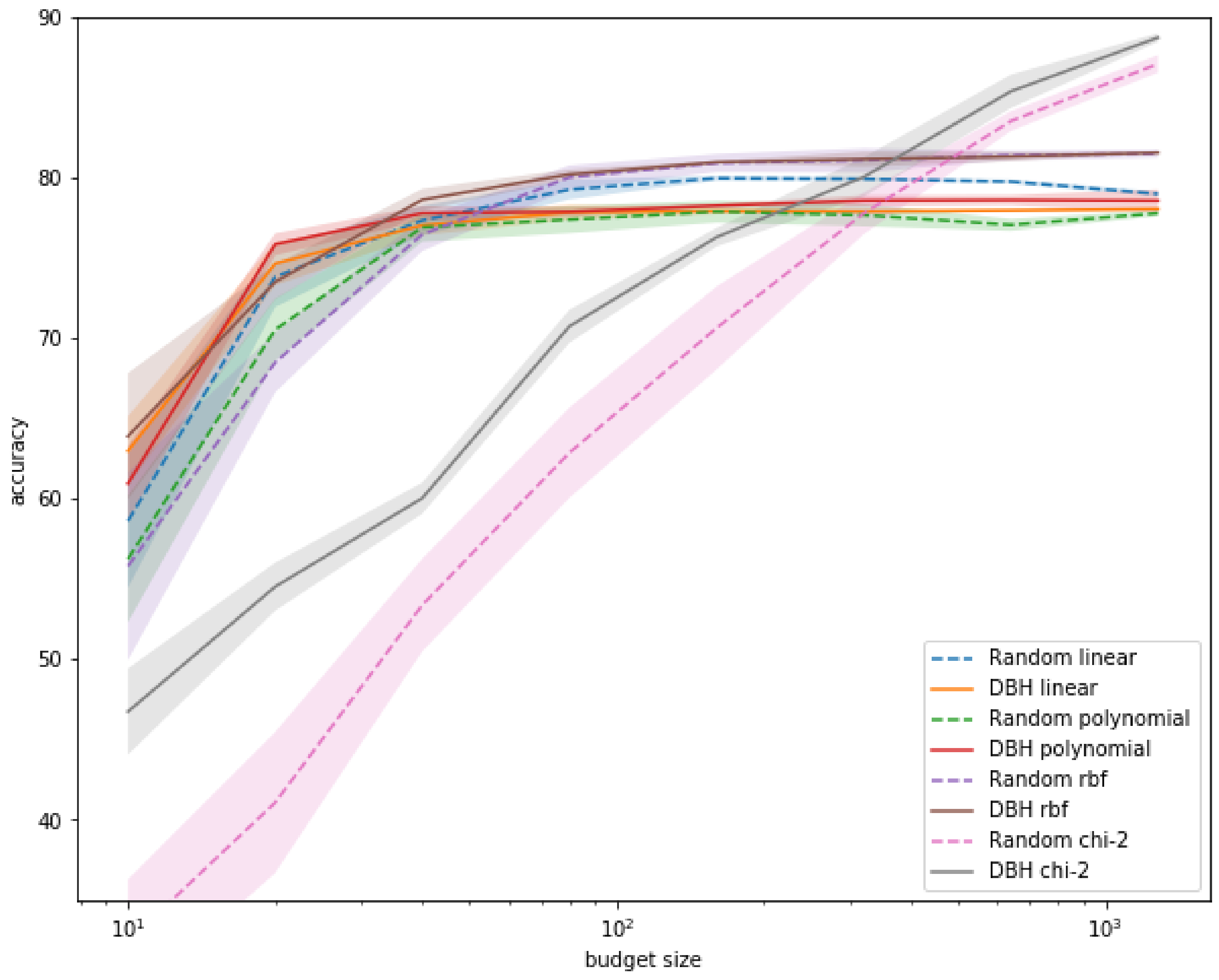

Figure 5-7: Budgeted SSOKMF Algorithm for MNIST

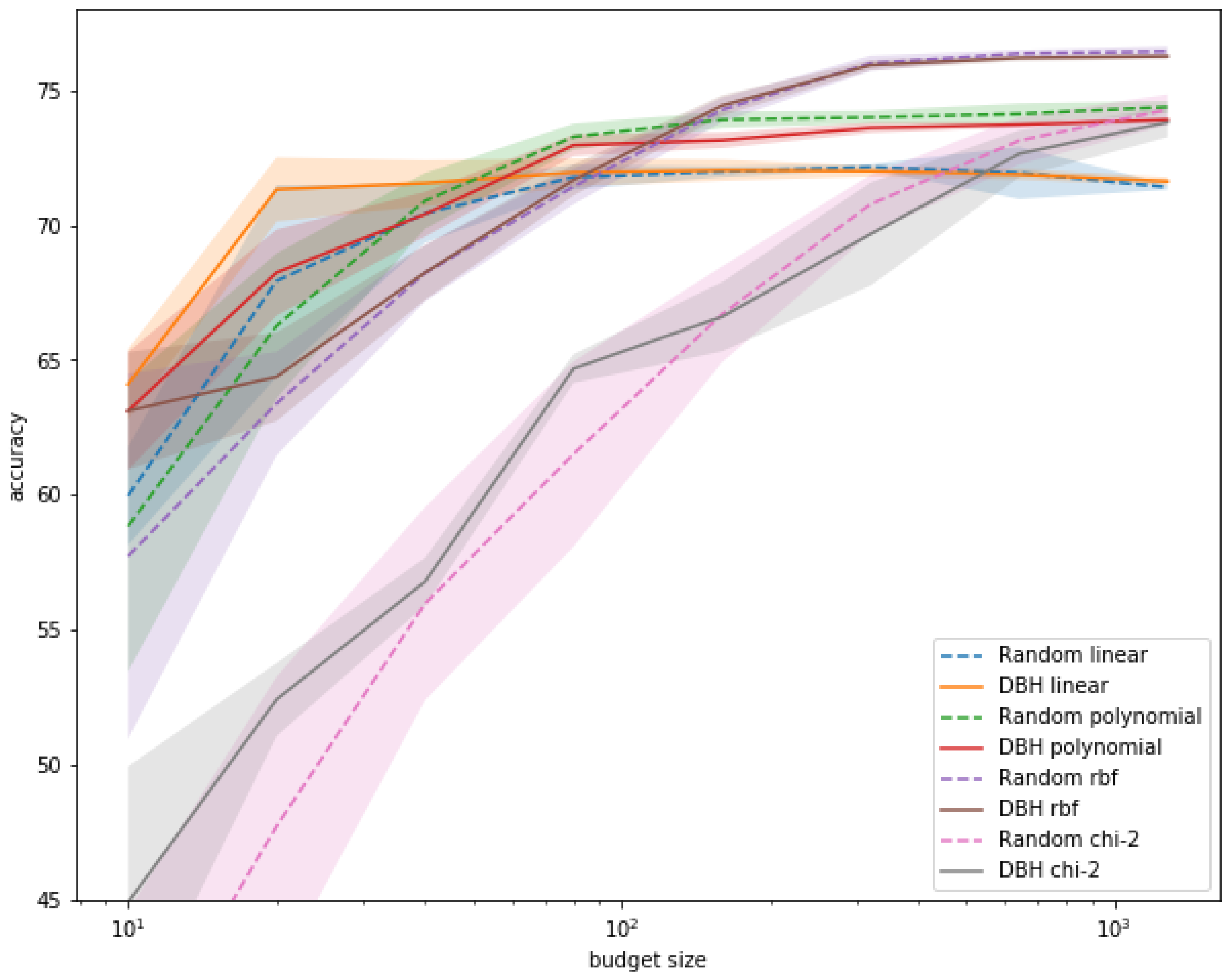

Figure 5-8: Budgeted SSOKMF Algorithm for Fashion

Kernel	Method	10	20	40	80	160	320	640	1280
Linear	Random	59.965 ± 1.81978	67.931 ± 3.54892	70.424 ± 1.08112	71.773 ± 0.45491	71.975 ± 0.20169	72.145 ± 0.10526	71.931 ± 0.96600	71.402 ± 0.09670
	DBH	64.101 ± 1.29974	71.318 ± 1.18743	71.549 ± 0.85434	71.946 ± 0.51249	72.028 ± 0.39628	71.996 ± 0.20277	71.871 ± 0.19634	71.616 ± 0.15732
Poly	Random	58.838 ± 5.38262	66.266 ± 2.68429	70.894 ± 1.02983	73.266 ± 0.48489	73.898 ± 0.30329	73.990 ± 0.24151	74.113 ± 0.38797	74.369 ± 0.22152
	DBH	63.114 ± 2.18562	68.240 ± 1.59272	70.399 ± 0.82658	72.953 ± 0.39561	73.138 ± 0.28562	73.598 ± 0.19635	73.717 ± 0.10363	73.895 ± 0.09462
RBF	Random	57.733 ± 6.76862	63.385 ± 1.91839	68.224 ± 1.04595	71.421 ± 0.65270	74.263 ± 0.44933	75.987 ± 0.28588	76.356 ± 0.12529	76.432 ± 0.18241
	DBH	63.120 ± 2.17562	64.378 ± 1.63930	68.239 ± 1.01756	71.664 ± 0.57261	74.423 ± 0.37562	75.921 ± 0.18564	76.191 ± 0.11937	76.253 ± 0.11299
Chi^2	Random	38.389 ± 6.20782	47.731 ± 5.53423	55.972 ± 3.57820	61.516 ± 3.43047	66.694 ± 1.77674	70.768 ± 1.08816	73.124 ± 0.87797	74.271 ± 0.56690
	DBH	44.911 ± 5.03664	52.422 ± 1.35057	56.789 ± 0.88388	64.680 ± 0.53033	66.603 ± 1.27799	69.662 ± 1.90918	72.638 ± 0.86787	73.802 ± 0.54856

Table 5-6: SSOKMF Algorithm for Fashion

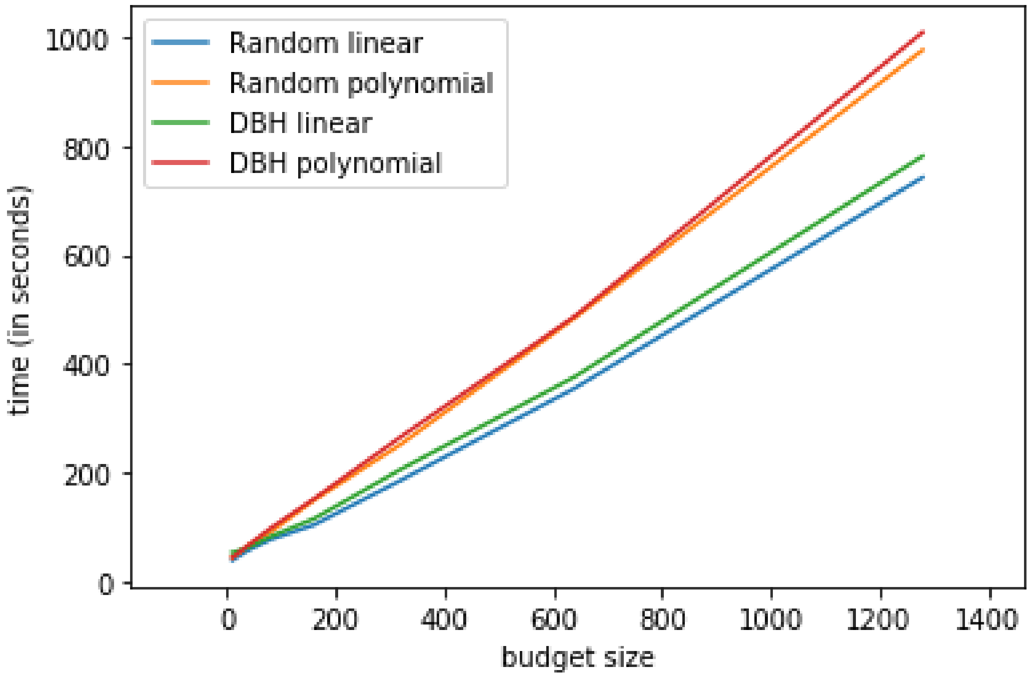

Figure 5-9: Training times SSOKMF algorithm

Dataset	Kernel	Method	Metric	100	200	400	800	1600
MNIST	Linear	Random	Silhouette	0.054282	0.062761	0.059066	0.055952	0.057852
			Fowlkes	0.335933	0.365679	0.320509	0.340059	0.352507
			Adjusted	0.265922	0.280658	0.172619	0.256655	0.287336
			Calinski	1102.175	1852.812	1709.791	1828.319	1294.055
		DBH	Silhouette	0.059749	0.057661	0.061438	0.063496	0.063475
			Fowlkes	0.358528	0.381417	0.322452	0.353997	0.356066
			Adjusted	0.250118	0.300170	0.227752	0.270548	0.290987
			Calinski	1687.638	1833.704	2629.871	1974.527	1324.711
	Poly	Random	Silhouette	0.052245	0.058438	0.056290	0.057788	0.059002
			Fowlkes	0.301699	0.351399	0.353073	0.339468	0.350791
			Adjusted	0.185328	0.262706	0.211961	0.252455	0.253440
			Calinski	1574.958	1743.086	2623.169	1792.237	1789.763
		DBH	Silhouette	0.054533	0.052296	0.058122	0.053423	0.056630
			Fowlkes	0.308069	0.315941	0.335699	0.325883	0.308840
			Adjusted	0.214494	0.244728	0.245969	0.186976	0.115783
			Calinski	1703.782	1903.076	1799.907	2087.573	2097.498
Fashion	Linear	Random	Silhouette	0.110166	0.133932	0.125616	0.138084	0.137671
			Fowlkes	0.377694	0.403035	0.434970	0.425285	0.440913
			Adjusted	0.270652	0.330311	0.365857	0.356770	0.372084
			Calinski	5445.031	7304.384	6818.497	7353.225	7169.399
		DBH	Silhouette	0.130016	0.153168	0.121204	0.132703	0.162670
			Fowlkes	0.405900	0.360234	0.436370	0.445159	0.441093
			Adjusted	0.228924	0.217454	0.377957	0.353672	0.328619
			Calinski	7083.605	8847.857	9458.409	5172.134	5736.156
	Poly	Random	Silhouette	0.153450	0.151241	0.159755	0.162571	0.171283
			Fowlkes	0.392344	0.359553	0.341326	0.361216	0.403808
			Adjusted	0.162985	0.195807	0.196765	0.214465	0.256543
			Calinski	5332.86	9252.595	9656.249	9942.985	10001.17
		DBH	Silhouette	0.163980	0.165223	0.177271	0.182273	0.193934
			Fowlkes	0.339741	0.398389	0.386507	0.305613	0.296363
			Adjusted	0.153882	0.246730	0.128922	0.073094	0.051982
			Calinski	7462.935	8508.496	12462.34	14847.62	14448.04

Table 5-7: Budgeted Kernel K-Medoids

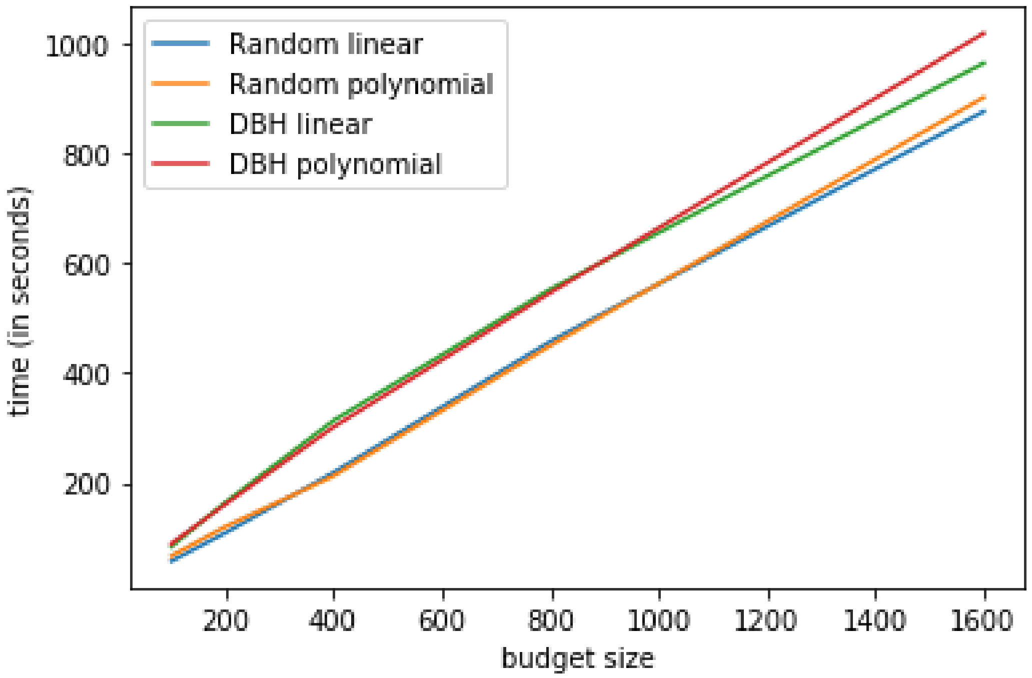

Figure 5-10: Training times kernel k-medoids algorithm

5.3. Discussion

The experimentation shows that the learning-on-a-budget method can significantly improve training times and memory usage in comparison to exact kernel methods where even for fairly large datasets with thousands of samples calculating a full kernel matrix is infeasible and very costly. Although this improvement in efficiency had a cost in performance as expected, we can see that for most of our classification models the accuracy using budgets of around 2 % of the data is still greater than 70 % in a multiclass setting, which may be good enough for some applications where memory and time are more important than an almost perfect accuracy. In addition to this, we also can see that when training a model with different budget sizes, after the budget starts to increase, in most cases we get to a stable point where increasing the budget even more does not really increase accuracy. This signifies that we could use the faster approximation without a loss in the accuracy performance. However, we see that this stabilization point is not the same in all configurations; it is still an open question how to find that point before training the models.

Regarding the DBH method to select the budget, we can see that it is indeed a better way to select a budget for very small approximations, without a significant additional cost in time, meaning we could use this strategy when memory and time are extremely limited and get decent accuracy even with such restrictions. Nevertheless, we see that for bigger approximations the difference in performance between DBH and selecting a random budget fades away. This is likely due to the randomness in both methods. As when selecting a bigger budget in a purely random way, there is more chance to get points evenly distributed through the feature space and get as relevant samples as if we were using any other method.

6 Conclusions and future work

This thesis presents different scalable versions of kernel methods based on the idea of using a budget. The main contribution of this work is the analysis of a simple approximation approach using the concept of budget from a theoretical and experimental perspective as an alternative to use different kernelized algorithms with lower computational complexity. Other relevant contributions of this work are the following:

- A technical comparison of different approximated kernel methods
- A deep exploration of the budget approach considering different alternatives to select the budget
- The formulation and implementation of several learning algorithms (supervised, semi-supervised and unsupervised) using the budget approach

It was shown that experimentally selecting a good budget increases the accuracy in a statistically significant manner in comparison to a simple random budget and that it is a very competitive scalable method for using kernel methods even compared to other traditional approximated approaches like Nyström.

Our approach to choose a relevant budget shows better improvements when the budget is actually smaller, which means this is a good approach for large datasets where the approximation needs to be very small in order to be computationally feasible. Moreover, the results also show that at some point increasing the budget does not necessarily translate into an increasing of accuracy.

An additional advantage shown with the budget approach is that it can be formulated for different learning algorithms despite of them being supervised, unsupervised or semi-supervised with good results in different metrics. Also, unlike other approximation methods, it does not rely on properties of the specific kernel function as long as it is used just with a valid kernel function.

Currently, data is growing in an exponential manner and therefore it is important to build not only robust, but also scalable learning methods. We think that the approach presented in this work has the potential to make kernel methods usable in large scale settings, particularly combined with neural network methods in hybrid formulations. In which cases, having a good budget could have huge impacts on the performance of the learning model. In that regard, we believe there are still some challenges to be considered for the future:

- Can we measure the performance of a budget before training in order to decide whether it is a good budget or not?
- Is it possible to formulate hybrid learning methods combining budgeted kernels with deep neural network approaches?
- Does the kernel function influence how representative a budget is in a dataset or is the budget relevance the same regardless of the kernel function?
- Is distance-based hashing suitable to optimize the performance of other distance-based learning methods such as neural networks?

References

[Ahuja and Angra, 2017] Ahuja, S. and Angra, S. (2017). Machine learning and its applications: A review.

[Baveye et al., 2015] Baveye, Y., Dellandréa, E., Chamaret, C., and Chen, L. (2015). Deep learning vs. kernel methods: Performance for emotion prediction in videos. In *2015 International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 77–83.

[Belkin et al., 2018] Belkin, M., Ma, S., and Mandal, S. (2018). To understand deep learning we need to understand kernel learning.

[Bengio et al., 2005] Bengio, Y., Delalleau, O., and Le Roux, N. (2005). The curse of dimensionality for local kernel machines. *Techn. Rep*, 1258:12.

[Borgwardt, 2011] Borgwardt, K. M. (2011). *Kernel Methods in Bioinformatics*, pages 317–334. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Bousquet and Herrmann, 2003] Bousquet, O. and Herrmann, D. J. (2003). On the complexity of learning the kernel matrix. *Advances in neural information processing systems*, pages 415–422.

[Boutsidis et al., 2009] Boutsidis, C., Mahoney, M. W., and Drineas, P. (2009). *An Improved Approximation Algorithm for the Column Subset Selection Problem*, pages 968–977.

[Chen et al., 2020] Chen, D., Jacob, L., and Mairal, J. (2020). Convolutional kernel networks for graph-structured data. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1576–1586. PMLR.

[Chitta et al., 2014] Chitta, R., Jin, R., Havens, T. C., and Jain, A. K. (2014). Scalable kernel clustering: Approximate kernel k-means. *CoRR*, abs/1402.3849.

[Chitta et al., 2012] Chitta, R., Jin, R., and Jain, A. K. (2012). Efficient kernel clustering using random fourier features. In *2012 IEEE 12th International Conference on Data Mining*, pages 161–170. IEEE.

[Choromanska et al., 2013] Choromanska, A., Jebara, T., Kim, H., Mohan, M., and Monteleoni, C. (2013). Fast spectral clustering via the nyström method. In *International Conference on Algorithmic Learning Theory*, pages 367–381. Springer.

[Cohen et al., 2015] Cohen, M. B., Lee, Y. T., Musco, C., Musco, C., Peng, R., and Sidford, A. (2015). Uniform sampling for matrix approximation. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 181–190.

[Courrieu, 2008] Courrieu, P. (2008). Fast computation of moore-penrose inverse matrices.

[Dasgupta et al., 2011] Dasgupta, A., Kumar, R., and Sarlos, T. (2011). Fast locality-sensitive hashing. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, page 1073–1081, New York, NY, USA. Association for Computing Machinery.

[Domingos, 2020] Domingos, P. (2020). Every model learned by gradient descent is approximately a kernel machine.

[Dostanic, 1993] Dostanic, M. (1993). Generalization of the mercer theorem. *Publ. Inst. Math.(Beograd)(NS)*, 54(68):63–70.

[Drineas and Mahoney, 2005] Drineas, P. and Mahoney, M. W. (2005). On the nyström method for approximating a gram matrix for improved kernel-based learning. *J. Mach. Learn. Res.*, 6:2153–2175.

[Drineas and Mahoney, 2016] Drineas, P. and Mahoney, M. W. (2016). Randnla: Randomized numerical linear algebra. 59(6).

[Drineas and Mahoney, 2017] Drineas, P. and Mahoney, M. W. (2017). Lectures on randomized numerical linear algebra. *CoRR*, abs/1712.08880.

[Ganapathiraju et al., 2004] Ganapathiraju, A., Hamaker, J. E., and Picone, J. (2004). Applications of support vector machines to speech recognition. *IEEE Transactions on Signal Processing*, 52(8):2348–2355.

[Garriga-Alonso et al., 2018] Garriga-Alonso, A., Rasmussen, C. E., and Aitchison, L. (2018). Deep convolutional networks as shallow gaussian processes. *arXiv preprint arXiv:1808.05587*.

[Gittens and Mahoney, 2013] Gittens, A. and Mahoney, M. W. (2013). Revisiting the nystrom method for improved large-scale machine learning. *CoRR*, abs/1303.1849.

[Glasmachers and Qaadan, 2018] Glasmachers, T. and Qaadan, S. (2018). Speeding up budgeted stochastic gradient descent SVM training with precomputed golden section search. *CoRR*, abs/1806.10180.

[Hassan et al., 2012] Hassan, E., Chaudhury, S., and Gopal, M. (2012). Feature combination in kernel space for distance based image hashing. *IEEE Transactions on Multimedia*, 14(4):1179–1195.

[Hearst et al., 1998] Hearst, M. A., Dumais, S. T., Osuna, E., Platt, J., and Scholkopf, B. (1998). Support vector machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28.

[Hoi et al., 2013] Hoi, S. C., Wang, J., Zhao, P., Zhuang, J., and Liu, Z.-Y. (2013). Large scale online kernel classification. In *Twenty-Third International Joint Conference on Artificial Intelligence*. Citeseer.

[Huang et al., 2013] Huang, P.-S., Deng, L., Hasegawa-Johnson, M., and He, X. (2013). Random features for kernel deep convex network. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3143–3147. IEEE.

[Jian et al., 2017] Jian, L., Shen, S., Li, J., Liang, X., and Li, L. (2017). Budget online learning algorithm for least squares svm. *IEEE Transactions on Neural Networks and Learning Systems*, 28(9):2076–2087.

[Kamilaris and Prenafeta-Boldú, 2018] Kamilaris, A. and Prenafeta-Boldú, F. X. (2018). Deep learning in agriculture: A survey. *Computers and electronics in agriculture*, 147:70–90.

[Khirirat et al., 2017] Khirirat, S., Feyzmahdavian, H. R., and Johansson, M. (2017). Mini-batch gradient descent: Faster convergence under data sparsity. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 2880–2887.

[Kudo and Matsumoto, 2003] Kudo, T. and Matsumoto, Y. (2003). Fast methods for kernel-based text analysis. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 24–31.

[Kumar et al., 2017] Kumar, A., Goyal, S., and Varma, M. (2017). Resource-efficient machine learning in 2 KB RAM for the internet of things. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1935–1944, International Convention Centre, Sydney, Australia. PMLR.

[Kumar et al., 2012] Kumar, S., Mohri, M., and Talwalkar, A. (2012). Sampling methods for the nyström method. *The Journal of Machine Learning Research*, 13(1):981–1006.

[LeCun and Cortes, 2010] LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

[Li et al., 2009] Li, H., Liang, Y., and Xu, Q. (2009). Support vector machines and its applications in chemistry. *Chemometrics and Intelligent Laboratory Systems*, 95(2):188–198.

[Li and Marlin, 2015] Li, S. C.-X. and Marlin, B. M. (2015). Classification of sparse and irregularly sampled time series with mixtures of expected gaussian kernels and random features. In *UAI*, pages 484–493.

[Litjens et al., 2017] Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., Van Der Laak, J. A., Van Ginneken, B., and Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88.

[Martinsson and Tropp, 2021] Martinsson, P.-G. and Tropp, J. (2021). Randomized numerical linear algebra: Foundations & algorithms.

[Mehrkanoon and Suykens, 2018] Mehrkanoon, S. and Suykens, J. A. (2018). Deep hybrid neural-kernel networks using random fourier features. *Neurocomputing*, 298:46–54.

[Mehrkanoon et al., 2017] Mehrkanoon, S., Zell, A., and Suykens, J. A. (2017). Scalable hybrid deep neural kernel networks. In *ESANN*.

[Min et al., 2017] Min, S., Lee, B., and Yoon, S. (2017). Deep learning in bioinformatics. *Briefings in bioinformatics*, 18(5):851–869.

[Mineiro and Karampatziakis, 2015] Mineiro, P. and Karampatziakis, N. (2015). Fast label embeddings via randomized linear algebra. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 37–51. Springer.

[Paul et al., 2015] Paul, S., Magdon-Ismail, M., and Drineas, P. (2015). Column selection via adaptive sampling. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 406–414. Curran Associates, Inc.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Pilario et al., 2020] Pilario, K. E., Shafiee, M., Cao, Y., Lao, L., and Yang, S.-H. (2020). A review of kernel methods for feature extraction in nonlinear process monitoring. *Processes*, 8(1):24.

[Rahimi and Recht, 2007] Rahimi, A. and Recht, B. (2007). Random features for large-scale kernel machines. In *Proceedings of the 20th International Conference on Neural*

Information Processing Systems, NIPS'07, page 1177–1184, Red Hook, NY, USA. Curran Associates Inc.

[Rojo-Álvarez et al., 2018] Rojo-Álvarez, J. L., Martínez-Ramón, M., Muñoz-Mari;, J., and Camps-Valls, G. (2018). *Reproducing Kernel Hilbert Space Models for Signal Processing*, pages 241–279.

[Rudi and Rosasco, 2017] Rudi, A. and Rosasco, L. (2017). Generalization properties of learning with random features. In *NIPS*, pages 3215–3225.

[Rudin, 1990] Rudin, W. (1990). *Fourier Analysis on Groups*. Wiley Classics Library. Wiley.

[Schölkopf et al., 2007] Schölkopf, B., Platt, J., and Hofmann, T. (2007). *Support Vector Machines on a Budget*, pages 345–352.

[Scholkopf, 2001] Scholkopf, B. (2001). The kernel trick for distances. *Advances in neural information processing systems*, pages 301–307.

[Schuld, 2021] Schuld, M. (2021). Quantum machine learning models are kernel methods.

[Shawe-Taylor and Cristianini, 2004] Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press, USA.

[Sheikholeslami and Giannakis, 2017] Sheikholeslami, F. and Giannakis, G. B. (2017). Scalable kernel-based learning via low-rank approximation of lifted data. In *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 596–603.

[Sinha and Duchi, 2016] Sinha, A. and Duchi, J. C. (2016). Learning kernels with random features. In *NIPS*, pages 1298–1306.

[Slaney and Casey, 2008] Slaney, M. and Casey, M. (2008). Locality-sensitive hashing for finding nearest neighbors [lecture notes]. *Signal Processing Magazine, IEEE*, 25:128 – 131.

[Sriperumbudur and Szabo, 2015] Sriperumbudur, B. K. and Szabo, Z. (2015). Optimal rates for random fourier features.

[Sun, 2005] Sun, H. (2005). Mercer theorem for rkhs on noncompact sets. *Journal of Complexity*, 21(3):337–349.

[Suykens and Vandewalle, 1999] Suykens, J. and Vandewalle, J. (1999). Least squares support vector machine classifiers. *Neural Processing Letters*, 9:293–300.

[Tian et al., 2019] Tian, Y., Zhao, L., Peng, X., and Metaxas, D. N. (2019). Rethinking kernel methods for node representation learning on graphs.

[Trokicic and Todorovic, 2020] Trokicic, A. and Todorovic, B. (2020). On expected error of randomized nystrom kernel regression. *Filomat*, 34(11):3871–3884.

[Vanegas et al., 2018] Vanegas, J. A., Escalante, H. J., and González, F. A. (2018). Semi-supervised online kernel semantic embedding for multi-label annotation. In Mendoza, M. and Velastín, S., editors, *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 693–701, Cham. Springer International Publishing.

[Wang, 2006] Wang, D. E. J. (2006). Fast approximation of centrality. *Graph algorithms and applications*, 5(5):39.

[Wang et al., 2018] Wang, J., Cao, B., Yu, P., Sun, L., Bao, W., and Zhu, X. (2018). Deep learning towards mobile applications. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1385–1393.

[Wang et al., 2014] Wang, J., Shen, H. T., Song, J., and Ji, J. (2014). Hashing for similarity search: A survey. *CoRR*, abs/1408.2927.

[Wang et al., 2019] Wang, S., Gittens, A., and Mahoney, M. W. (2019). Scalable kernel k-means clustering with nyström approximation: relative-error bounds. *The Journal of Machine Learning Research*, 20(1):431–479.

[Wang et al., 2016] Wang, S., Luo, L., and Zhang, Z. (2016). Spsd matrix approximation vis column selection: Theories, algorithms, and extensions.

[Wang and Zhang, 2013] Wang, S. and Zhang, Z. (2013). Improving cur matrix decomposition and the nyström approximation via adaptive sampling. *The Journal of Machine Learning Research*, 14(1):2729–2769.

[Wang et al., 2017] Wang, Y., Liu, X., Dou, Y., Lv, Q., and Lu, Y. (2017). Multiple kernel learning with hybrid kernel alignment maximization. *Pattern Recognition*, 70:104–111.

[Wang et al., 2012] Wang, Z., Crammer, K., and Vucetic, S. (2012). Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training. *Journal of Machine Learning Research*, 13(100):3103–3131.

[Williams and Seeger, 2001] Williams, C. K. I. and Seeger, M. (2001). Using the nyström method to speed up kernel machines. In Leen, T. K., Dietterich, T. G., and Tresp, V., editors, *Advances in Neural Information Processing Systems 13*, pages 682–688. MIT Press.

[Witten and Candes, 2015] Witten, R. and Candes, E. (2015). Randomized algorithms for low-rank matrix factorizations: sharp performance bounds. *Algorithmica*, 72(1):264–281.

[Wu et al., 2018] Wu, L., Chen, P.-Y., Yen, I. E.-H., Xu, F., Xia, Y., and Aggarwal, C. (2018). Scalable spectral clustering using random binning features. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2506–2515.

[Xiao et al., 2017] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[Yang, 2002] Yang, M.-H. (2002). Kernel eigenfaces vs. kernel fisherfaces: Face recognition using kernel methods. In *Fgr*, volume 2, page 215.

[Yu et al., 2016] Yu, F. X., Suresh, A. T., Choromanski, K., Holtmann-Rice, D. N., and Kumar, S. (2016). Orthogonal random features. *CoRR*, abs/1610.09072.

[Zhang et al., 2010] Zhang, D., Wang, J., Cai, D., and Lu, J. (2010). Self-taught hashing for fast similarity search. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 18–25.

[Zhang et al., 2017] Zhang, Q., Filippi, S., Gretton, A., and Sejdinovic, D. (2017). Large-scale kernel methods for independence testing. *Statistics and Computing*, 28(1):113–130.