A Survey of Efficient Kernel Methods

Ishika Soni (14275)

Lakshay Garg (13373)

Prawaan Singh (14495)

Shruti Agrawal (14668)

Shubham Jain (13689)

1 Introduction

Kernel methods, first introduced in 1964 by Aizerman et al. [1] are techniques which enable simple linear models to learn complex non-linear hypotheses. These methods essentially map the input space into a high dimensional *feature space* where the data becomes easily separable. Kernel functions allow these methods to compute pairwise similarity between data points and avoid computing the feature space. Kernel methods have demonstrated their efficacy in the past in several pattern recognition challenges but these methods tend to become unfeasible as the data grows. Computing pairwise similarity between points required $\mathcal{O}(N^2)$ space and time. Training algorithms require inverting the kernel matrix which requires $\mathcal{O}(N^3)$ time. In this project, we survey and experiment with some efficient alternatives to kernel methods.

The methods that we survey belong to three broad categories of methods listed below

1. Explicit feature map computation
2. Memory efficient kernel approximations
3. Online learning methods

Following three sections describe the various methods that we surveyed. Section 5 onward we provide the implementation details, experimental results and present discussion on the experimental results.

2 Approximate Feature Maps

This class of methods work by constructing an approximation of the feature space representation of the data. They learn a map $z : \mathbb{R}^d \to \mathbb{R}^D$ such that $k(x, y) \approx z(x)^T z(y)$.

2.1 Nyström Method

Nyström Method was proposed by Christopher et al.[2] and the current modified algorithm was provided by Petros et al. [13] This method gives easily-interpretable low-rank approximation to an $n \times n$ Gram matrix G such that computations of interest may be performed more rapidly. The approximation is of the form $\tilde{G}_k = CW_K^+C^T$ where C is a matrix consisting of a small number c of columns of G and W_k is the best rank-k approximation to W, the matrix formed by the intersection between those c columns of G and the corresponding c rows of G. An important aspect of the algorithm is the probability distribution used to randomly sample the columns; we will use uniform distribution in our code but a judiciously-chosen and data-dependent nonuniform probability distribution is more preferable as chosen by Petros et al. in [2].

Algorithm 1 Nyström Method

Require: $n \times n$ Gram matrix G, $\{p_i\}_{n=1}^{n}$ such that $\sum_{i=1}^{n} p_i = 1$, $c \leq n$, and $k \leq c$.
Ensure: $n \times n$ Gram matrix $\tilde{G}$
1: Define the $(n \times c)$ matrix $S = 0_{n \times c}$;
2: Define the $(c \times c)$ matrix $D = 0_{c \times c}$;
3: **for** $t = 1, ..., c$ **do**
4: Pick $i_t \in [n]$, where $Pr(i_t = i) = p_i$;
5: $D_{tt} = (cp_{i_t})^{-\frac{1}{2}}$;
6: $S_{i_t t} = 1$;
7: **end for**
8: Let $C = GSD$ and $W = DS^T GSD$
9: Compute W_k, the best rank-k approximation to W.
10: Return $\tilde{G}_k = CW_K^+ C^T$.

2.2 Random Fourier Features

Random Fourier Features were proposed in 2007 by Rahimi et al. [9]. This method generates a feature space representation for data by exploiting the Bochner's theorem which states that the Fourier transform for a positive semi-definite kernel function is a probability distribution if properly scaled. This result allows us to interpret the Fourier transform integral of the kernel as an expectation.

$$k(x-y) = \int_{\mathcal{R}^d} p(\omega) e^{j\omega'(x-y)} d\omega = \mathbb{E}_\omega[e^{j\omega' x} e^{-j\omega' y}]$$

Defining $\zeta_\omega(x) = e^{j\omega' x}$, we observe that $\zeta_\omega(x)\zeta_\omega(y)^*$ is an unbiased estimate of $k(x, y)$ when ω is drawn from p. To obtain a real-valued random feature for k, note that the probability distribution $p(\omega)$ and the kernel $k(\Delta)$ are real, so the integrand may be replaced by $\cos(\omega'(x-y))$. Defining $z_\omega(x) = \begin{bmatrix}\cos(\omega' x) & \sin(\omega' x)\end{bmatrix}'$ gives a real valued mapping that satisfies the the condition $\mathbb{E}[z_\omega(x)' z_\omega(y)] = k(x, y)$, since $z_\omega(x)' z_\omega(y) = \cos \omega'(x-y) = \mathrm{Re}\{\zeta_\omega(x)\zeta_\omega(y)^*\}$. The variance of $z_\omega(x)' z_\omega(y)$ can be lowered by concatenating D randomly chosen z_ω into a column vector z and normalizing each component by $\sqrt{D}$. The inner product of points represented by this 2D-dimensional random feature $\mathbf{z}$ is a sample average of $z_{\omega_j}(x)' z_{\omega_j}(y)$ and therefore a lower variance approximation to the expectation.

Algorithm 2 Random Fourier Features

Require: A positive semi-definite shift-invariant kernel $k(x, y) = k(x - y)$
Ensure: A randomized feature map $z(x) : \mathcal{R}^d \to \mathcal{R}^{2D}$ such that $z(x)^T z(x) \approx k(x - y)$
1: Compute the Fourier transform p of kernel $k : p(\omega) = \frac{1}{2\pi} \int e^{-j\omega^T \delta} k(\delta) d\delta$
2: Draw D i.i.d. samples $\omega_1, \omega_2, \ldots, \omega_D \in \mathcal{R}^d$ from p
3: Let $\Omega = [\omega_1, \omega_2, \ldots, \omega_D]$ be a $d \times D$ matrix
4: Let $z(x) = \frac{1}{\sqrt{D}}[\cos(\Omega x) \ \sin(\Omega x)]$

2.3 Random Binning Features

Random Binning Features were proposed in 2007 by Rahimi et al. [9]. This algorithm seeks to partition the input space into randomly shifted grids at randomly chosen resolutions. The grids are constructed such that the probability that two points x and y are assigned to the same bin is the same as $k(x, y)$. Hence, we have an unbiased estimate of $k(x, y)$ as the inner product between a pair of transformed points is proportional to the number of times they are binned together.

A grid is defined by its pitch δ and a shift u which partitions the real number line into intervals $[u + n\delta, u + (n + 1)\delta]$ for all integers n. The pitch is sampled from a probability distribution p and the shift is sampled uniformly from $[0, \delta]$. This ensures that the probability that x and y have been

binned together is $k(x, y)$. For example, for a Laplacian Kernel, $k_{Laplacian}(x, y) = \exp(-|x - y|)$, $p(\delta) = \delta \exp(-\delta)$. For a Gaussian Kernel, is not positive everywhere, so it does not yield a random map. For a multidimensional setting, this same binning procedure is applied independently over all the dimensions. Since the binning is done independently, the probability that two points x and y are binned together in every dimension is $k(x - y)$.

Algorithm 3 Random Binning Features

Require: A point $x \in \mathcal{R}^d$. A kernel function $k(x, y) = \prod_{m=1}^{d} k_m(|x^m - y^m|)$, so that $p_m(\Delta) = \Delta\ddot{k}_m(\Delta)$ is a probability distribution on $\Delta \geq 0$.

Ensure: A randomized feature map $z(x)$ so that $z(x)'z(y) \approx k(x - y)$

1: **for** $p = 1...P$ **do**
2: Draw grid parameters $\delta, u \in \mathcal{R}^d$ with the pitch $\delta^m \sim p_m$ and shift u^m from the uniform distribution on $[0, \delta^m]$
3: Let z return the coordinate of the bin containing x as a binary indicator vector $z_p(x) = hash(\lceil \frac{x^1 - u^1}{\delta^1} \rceil, ..., \lceil \frac{x^1 - u^1}{\delta^1} \rceil)$
4: **end for**
5: $z(x) = \sqrt{\frac{1}{P}}[z_1(x)...z_p(x)]'$.

2.4 Random Kitchen Sinks

This method [8] uses the fact that randomization is cheaper than optimization. Given a training set we wish to fit a function to minimize the empirical loss and show its convergence to the actual loss function.

The algorithm randomly picks K-parameter vectors w_1,w_2,...,w_K, which are iid's, from the $p(w)$ distribution function defined on the parameters of ϕ, the given bounded feature function (i.e, $|\phi(x; w)| \leq 1$). All input points x_i's are mapped into the space defined by the feature function and called z_i's. We now define the risk minimization on these mapped points and accordingly calculate and minimize the empirical loss function over α, the weights given to different feature functions. Now, the weights assigned to individual features functions $\phi(x; w_i)$, α_i's are estimated by minimizing the obtained cost function.

Algorithm 4 Fitting procedure for Random Kitchen Sinks

Require: A dataset $\{x_i, y_i\}_{i=1:m}$ of m points, a bounded feature function $|\phi(x; w)| \leq 1$, an integer K, a scalar C, and a probability distribution $p(w)$ on parameters of ϕ.

Ensure: A function $\hat{f}(x) = \sum_{k=1}^{K} \phi(x; w_k)\alpha_k$.

1: Draw K iid samples $w_1, w_2, \ldots, w_K$ from p
2: Featurize the input: $z_i \leftarrow [\phi(x_i; w_1), \ldots, \phi(x_i; w_K)]^T$
3: With w fixed, solve the empirical risk minimization problem

$$\min_{\alpha \in \mathbb{R}^K} \frac{1}{m} \sum_{i=1}^{m} c(\alpha^T z_i, y_i)$$

$$\text{s.t.} \|\alpha\|_\infty \leq C/K$$

The fitting function $\hat{f}(x)$ is defined as the weighted sum of all the feature functions determined by different parameter vectors, $\hat{f}(x) = \sum_{k=1}^{K} \alpha_k \phi(x; w_k)$.

This algorithm unlike in the optimal algorithm replaces the minimization step over parameter vector by randomly choosing it from the probability distribution and than minimizing over the weight vector. Our choice of parameter vectors is thus not the optimal choice but a randomized one.

It requires $\mathcal{O}(nd)$ computation and storage.

This algorithm, with a significantly higher probability, returns the fitting function with error bounds close to the ones obtained by optimal function.

2.5 Fastfood

This paper [3] is mainly on a storage and computation optimization on Kitchen-Sinks. Fastfood obtains similar accuracy as that of full kernel expansions and Random Kitchen Sinks, with significant decrease in storage and computation.

The main idea behind this optimization lies in the fact that Random Gaussian Matrices, used in Kitchen Sinks, can be approximated as the combination of Hadamard matrix and diagonal Gaussian matrices. This approximation matrices easy to multiply and store. This approximation of the matrix significantly improves the computation and storage and brings down to $\mathcal{O}(n \log(d))$ computation time and $\mathcal{O}(n)$ storage.

3 Memory Efficient Kernel Methods

3.1 Memory Efficient Kernel Approximations

Current state of the art methods aim to form a low rank approximation $G = CC^T$ with $C \in R^{n \times k}$ and $k << n$. Although it is well known that Singular Value Decomposition (SVD) can compute the best rank k approximation it often cannot be applied to kernel matrices as it requires the entire kernel matrix to be computed and stored. To overcome this issue, many methods have been proposed to approximate the best rank k approximation of kernel matrix, including Greedy basis selection techniques (Smola & Scholkopf, 2000), incomplete Cholesky decomposition and Nyström methods.

For practically used shift invariant kernels it is observed the structure of the kernel matrix changes from low-rank to block diagonal (without any low rank structure) as γ varies from $0 \Rightarrow \inf$. For RBF kernel $G \Rightarrow I$ as $\gamma \Rightarrow \inf$. Thus even the best rank k approximation have extremely large approximation error for large γ. Thus Low Rank Approximation work well for small γ and Block Kernel Approximation works well for large γ

A simple Block Kernel Approximation (BKA) proceeds as follows:

- Obtain a good partition of the data using k-means $V_1, V_2, \ldots, V_c$ where each V_s is a subset of $1, 2...n$
- The kernel is approximated as $\tilde{G} = G^{(1,1)} \oplus G^{(2,2)} \ldots \oplus G^{(c,c)}$
- Thus $\tilde{G}^{(s,s)} = G^{(s,s)}$ and $\tilde{G}^{(s,t)} = 0$ for $s \neq t$

Algorithm 5 Memory Efficient Kernel Approximation (MEKA)

Require: Data Points $\{(x_i)\}_{i=1}^n$, scaling parameter γ , rank k and number of clusters c
Ensure: A rank ck approximation $\tilde{G} = WLW^T$ using $\mathcal{O}(nk + (ck)^2)$ space
1: Generate the partition$\mathcal{V}_1, \ldots, \mathcal{V}_c$ by k means
2: For each $s = 1, \ldots, c$ perform the rank k approximation $G^{(s,s)} \approx W^{(s)}(W^{(s)})^T$ by standard Nyström
3: **for** each $(s,t)(s \neq t)$ **do**
4: Sample a submatrix $\tilde{G}^{(s,t)}$ from $G^{(s,t)}$ with row index set v_s and column index set v_t;
5: Form $W_{v_s}^{(s)}$ by selecting the rows in $W^{(s)}$ according to index set v_s
6: Form $W_{v_t}^{(t)}$ by selecting the rows $W^{(t)}$ in according to index set v_t
7: Solve the least squares problem $\tilde{G}^{(s,t)} \approx W_{v_s}^{(s)} L^{(s,t)} (W_{v_t}^{(t)})^T$to obtain $L^{(s,t)}$;
8: **end for**

MEKA[11] exploits both the low rank structure and clustering of data to learn a Memory Efficient Kernel Approximation. The data is clustered using k-means and a low rank approximation for each $G^{(s,s)}$ is calculated using Nyström method. We assume the same rank k for each block. Thus $\tilde{G}^{(s,s)} \approx W^{(s)} L^{(s,s)} (W^{(s)})^T$.

Thus the kernel matrix is approximated as $\tilde{G} \approx WLW^T$ where $W = W^{(1,1)} \oplus W^{(2,2)} \oplus \ldots \oplus W^{(c,c)}$. L is a link matrix of c^2 blocks of size k^2 each where each $L^{(s,t)}$ captures the interaction between the $s^t h$ and $t^t h$ clusters giving $\tilde{G}^{(s,t)} \approx W^{(s)} L^{(s,s)} (W^{(s)})^T$.

We use $\mathcal{O}(nk)$ space for the c diagonal blocks whereas only $\mathcal{O}(k^2)$ space is needed for each off-diagonal block. Thus $\mathcal{O}(nk + (ck)^2)$ space is used for the complete approximate kernel matrix.

4 Online Kernel Methods

Online methods are fundamentally different from batch learning methods as they do not have a separate training and testing phase, but since perceptron like classifiers once trained in an online setting can be used on test data, we explore the possibility of using them as efficient methods of learning classifiers by presenting data to the classifier in an online manner.

For our experiments, we implemented the methods proposed by [4] which are essentially perceptrons trained on Fourier and Nyström features. These methods are called the Fourier online gradient descent (FOGD) and Nyström online gradient descent (NOGD) respectively.

We however did not explore an important class of online SVMs which work by maintaining a budget of support vectors which are updated as more data is provided to the SVM.

5 Implementation Details

A number of methods were surveyed during the project. Implementations of some of these methods were available and used unmodified [6, 10], others had to be modified to suit our needs [12] and some of them were written from scratch. We used python [7] for our code due to the availability of numerical [5] and machine learning [6] libraries. Our code is available as a python package on GitHub (`github.com/lakshayg/yellow/`). The datasets we used are available at `github.com/lakshayg/pydata`.

6 Experiments

6.1 Experimental Methodology

We implemented several of the methods that we study to compare them against kernel methods. We compared the methods by taking their test time, training time and classification performance. In case of feature map approximation methods, we computed the features and trained a linear SVM on extracted features. The results are reported in tables 2, 3 and 4. The details of the datasets we used in our experiments are given in table 1.

Dataset	Description
BANANA	A very noisy artificially created dataset using a mixture of overlapping Gaussians that look like bananas. It contains 5300 classes with 3 attributes
USPS	One of the standard datasets for handwritten digit recognition. It consists of 7291 training images and 2007 test images
MNIST	Database for handwritten digit classification. Has 60000 training and 10000 test images with 784 dimensions
Forest Cover	Predicting forest cover type from cartographic variables only. Contains 581012 instances with 54 dimensions
Letter Recogn.	Consists of black-and-white rectangular pixel displays which are to be classified as one of 26 capital letters in the English alphabet. The character images are based on 20 different fonts with each letter randomly distorted to to produce 20000 unique stimuli
MAGIC04	MAGIC gamma telescope data. 19020 instances with 10 features in 2 classes

Table 1: Datasets used in for conducting experiments

6.2 Results

We performed comparison of the methods based on their training, testing times and classification accuracy. The results demonstrate that the approximate methods are much faster than the regular kernel methods while being competitive in terms of performance. The time taken is generally within two times of the linear SVM but with much better accuracy.

Among the feature approximation methods, there does not seem to be a clear *winner* and the performances are highly variable across datasets but there are a few observations worth mentioning

1. Fastfood generally has the longest training time which we suspect is due to the training time of SVM being longer for those features as the feature extraction process is almost 10 times faster for fastfood than other methods.
2. Nyström features tend to outperform Fourier features in terms of training time and test accuracy which corroborates the findings of [14].
3. Feature approximation methods outperform linear SVM in terms of training time in several cases which is because the SVM takes longer to converge in the linear case.
4. Contrary to other methods, Nyström features do a much better job on the MNIST and USPS datasets.
5. Fourier online gradient descent outperforms Nyström online gradient descent in training time and accuracy.
6. We also conducted some experiments on the CIFAR10 dataset and none of the feature approximation methods, linear SVM or kernel SVM were able to complete training in a reasonable time but online methods were able to do so and with a reasonable performance, this indicates that online methods offer substantial performance benefits over other methods.

Each of the feature approximation methods can give variable number of features as output dimension, we have presented those results here which seem to best represent them along with other methods due to the limited space, however the complete results are available at `tinyurl.com/zypb5du`.

Dataset	Linear SVM	Nyström	RFF	RKS	Fastfood	FOGD	NOGD	Kernel SVM
BANANA	0.0433	0.0755	0.1163	**0.0663**	0.09021	**0.0244**	0.0428	0.2048
MAGIC04	1.3654	**0.3721**	0.5488	0.4969	0.5287	**0.4097**	0.4814	6.9772
LETTERS	11.5490	**2.2720**	7.0539	8.3801	6.0679	**0.6937**	0.9839	4.9477
USPS	3.2111	0.3289	0.2156	**0.1919**	2.1740	**0.0189**	0.4860	3.3575
COVTYPE	24.2727	**0.5469**	1.3632	4.5047	4.3250	3.6581	**0.7044**	35.5756
MNIST	16.1708	1.5239	0.3546	**0.237**	5.9429	**0.0311**	0.6495	13.9363

Table 2: Training time for different classifiers

Dataset	Linear SVM	Nyström	RFF	RKS	Fastfood	FOGD	NOGD	Kernel SVM
BANANA	0.0002	**0.0050**	0.0065	0.0053	0.00945	0.0074	**0.0139**	0.0428
MAGIC04	0.0005	**0.0093**	0.0301	0.0288	0.0346	0.1475	**0.1241**	1.4669
LETTERS	0.0023	**0.0184**	0.1765	0.1651	0.1738	0.1618	**0.1556**	3.8674
USPS	0.0028	0.0093	**0.0015**	0.0023	0.0365	**0.0042**	0.0686	1.4054
COVTYPE	0.0015	0.0049	0.0175	0.1689	0.1898	0.7955	**0.1580**	8.6684
MNIST	0.0088	0.0713	0.0078	0.0069	0.1703	**0.0038**	0.1044	5.7849

Table 3: Testing Time for different methods

Dataset	Linear SVM	Nyström	RFF	RKS	Fastfood	FOGD	NOGD	Kernel SVM
BANANA	0.556	**0.904**	0.896	0.901	0.892	0.8498	**0.8651**	0.902
MAGIC04	0.790	**0.825**	0.818	0.801	0.820	0.7947	**0.7986**	0.902
LETTERS	0.705	0.859	0.894	0.691	**0.909**	**0.8013**	0.7630	0.823
USPS	0.928	**0.943**	0.167	0.163	0.149	0.1099	**0.9261**	0.924
COVTYPE	0.656	0.677	0.676	**0.746**	0.741	**0.6824**	0.6014	0.977
MNIST	0.854	**0.939**	0.107	0.106	0.104	0.100	**0.872**	0.528

Table 4: Testing accuracy for different methods

Dataset	no. of clusters-c	cluster rank-k	γ	Approximation Error
IJCNN	10	1280	1.0000	0.003319
PENDIGITS	5	640	2.0000	0.072549

Table 5: Experimental results for memory efficient kernel approximation

7 Project Takeaways

We surveyed a number of techniques for making kernel methods efficient. Each of the methods we studied approach the problem differently thus the project gave us an opportunity to familiarize ourselves with a breadth of topics. This being an active field of research for long time, allowed us to explore some classical and some more recent works. We also looked at the optimization aspect of machine learning which is essential if the methods are to be used on real world data. The project involved both extensive literature review and implementation. Not only our findings support work by other authors, we also performed experiments which had not been done previously.

Other than the academic side, this project required us to work in a much larger team as compared to the regular team size in other courses. Having a larger team was both helpful and challenging at times but finally allowed us to do extensive literature survey which would not have been possible in a smaller team and would not have done justice to the project.

The project was a great learning experience and allowed us to go beyond the course and explore an area of active research.

References

[1] M A Aizerman, E M Braverman, and L I Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25(6):917–936, 1964.

[2] Petros Drineas and Michael W Mahoney. On the NyströM Method for Approximating a Gram Matrix for Improved Kernel-Based Learning. *J. Mach. Learn. Res.*, 6:2153–2175, 12 2005.

[3] Quoc Viet Le, Tamas Sarlos, and Alexander J Smola. Fastfood: Approximate Kernel Expansions in Loglinear Time. 2014.

[4] Jing Lu, Steven C H Hoi, Jialei Wang, Peilin Zhao, and Zhi-Yong Liu. Large Scale Online Kernel Learning. *Journal of Machine Learning Research*, 17:1–43, 2016.

[5] Travis E Oliphant. SciPy: Open source scientific tools for Python. *Computing in Science and Engineering*, 9:10–20, 2007.

[6] Fabian Pedregosa, Gal Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and douard

Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2012.

[7] Python Software Foundation. Python Language Reference, version 2.7, 2013.

[8] Ali Rahimi and Benjamin Recht. Weighted Sums of Random Kitchen Sinks: Replacing minimization with randomization in learning.

[9] Ali Rahimi and Benjamin Recht. Random Features for Large-Scale Kernel Machines. In *Neural Information Processing Systems*, pages 1177–1184, 2007.

[10] Si Si, Cho-Jui Hsieh, and Inderjit S Dhillon. Memory Efficient Kernel Approximation.

[11] Si Si, Cho Jui Hsieh, and Inderjit S Dhillon. Memory efficient kernel approximation. In *31st International Conference on Machine Learning, ICML 2014*, volume 2, pages 1057–1069. International Machine Learning Society (IMLS), 2014.

[12] Dougal J. Sutherland. Fastfood.py, 2014.

[13] Christopher K I Williams and Matthias Seeger. Using the Nyström Method to Speed Up Kernel Machines. In T K Leen, T G Dietterich, and V Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 682–688. MIT Press, 2001.

[14] Tianbao Yang, Yu-Feng Li, Mehrdad Mahdavi, Rong Jin, and Zhi-Hua Zhou. Nyström Method vs Random Fourier Features: A Theoretical and Empirical Comparison.