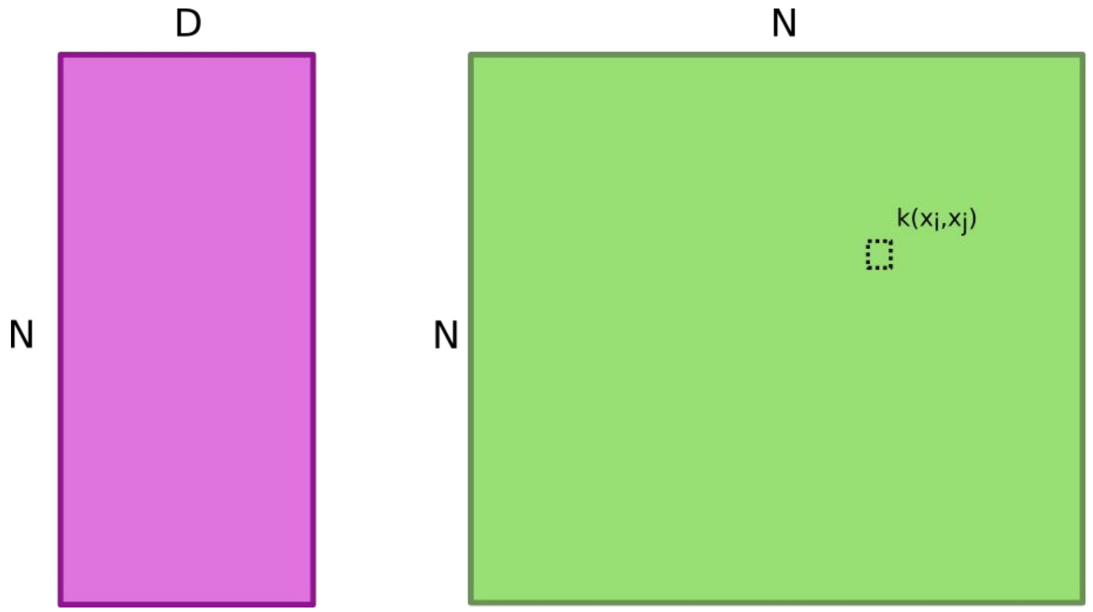
# Efficient Kernel Methods for Machine Learning

Ishika Soni · Lakshay Garg · Prawaan Singh · Shruti Agrawal · Shubham Jain

## Kernel Methods

- Allow learning complex non-linear hypotheses by mapping input to a high-dimensional space
- The high dimensional mapping is not explicitly constructed but stored in form of dot products

## Kernel Methods



Training takes time cubic in the number of training examples

Prediction requires computing kernel similarity with each point in training set

☹

## Methods Surveyed

Several alternatives to kernel methods were studied. The methods can be grouped under the following classes

- Explicit Feature Map Computation
- Kernel Approximation
- Online Learning

## Explicit Feature Maps

Every kernel induces a high (possibly infinite) dimensional mapping. The kernel matrix stores similarities between points in form of inner product between these high dim vectors

$$k(x_i, x_j) = \phi(x_i)^T \phi(x_j) \approx z(x_i)^T z(x_j)$$

Goal is to compute a mapping $z$ which can approximate the kernel similarity

## Nyström Method

- Computes low-rank approx. of kernel matrix
- Select a random subset of training data and compute kernel matrix $\tilde{K}$ for random samples
- Represents each data point by a vector based on its kernel similarity to the random samples and the sampled kernel matrix

## Random Fourier Features

Approximate positive definite, shift invariant kernel map

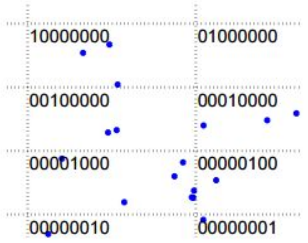
$$k(\mathbf{x}-\mathbf{y}) = \int_{\mathcal{R}^{\mathbf{d}}} \mathbf{p}(\omega)\mathbf{e}^{\mathbf{j}\omega^{\mathbf{T}}(\mathbf{x}-\mathbf{y})}\mathbf{d}\omega = \mathbb{E}_{\omega}[\mathbf{z}(\mathbf{x})\mathbf{z}(\mathbf{y})^{\mathbf{T}}]$$

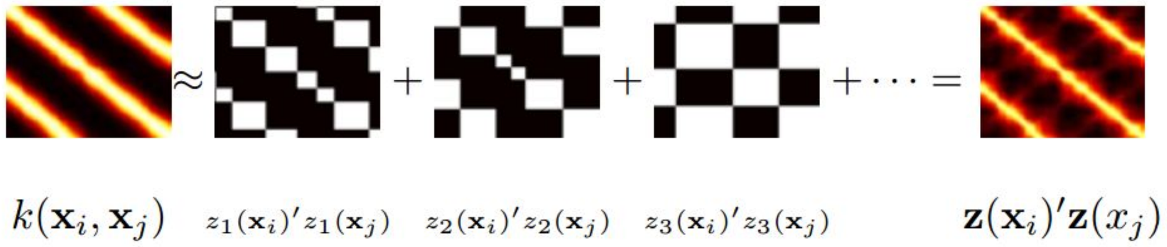
$$z(x) = \begin{bmatrix} \cos(\omega^T x) \\ \sin(\omega^T x) \end{bmatrix}$$ where ω is drawn from p(ω)

Feature is obtained by concatenating multiple z

$$z(x) = \sqrt{\frac{1}{D}}\left[\cos \omega_1^T x \quad \ldots \quad \cos \omega_D^T x \quad \sin \omega_1^T x \quad \ldots \quad \sin \omega_D^T x\right]^T$$

## Random Binning Features





$k(\mathbf{x}_i, \mathbf{x}_j)$
$\approx$
$z_1(\mathbf{x}_i)' z_1(\mathbf{x}_j)$
$+$
$z_2(\mathbf{x}_i)' z_2(\mathbf{x}_j)$
$+$
$z_3(\mathbf{x}_i)' z_3(\mathbf{x}_j)$
$+ \cdots =$
$\mathbf{z}(\mathbf{x}_i)' \mathbf{z}(x_j)$

Repeatedly partition the input space using a randomly shifted grid at a randomly chosen resolution and assign to each point x the bit string z(x) associated with the bin to which it is assigned

The grid parameters are drawn from the probability distribution obtained by computing the hat transform of the kernel function

## Random Kitchen Sinks

Approximate the kernel feature map as

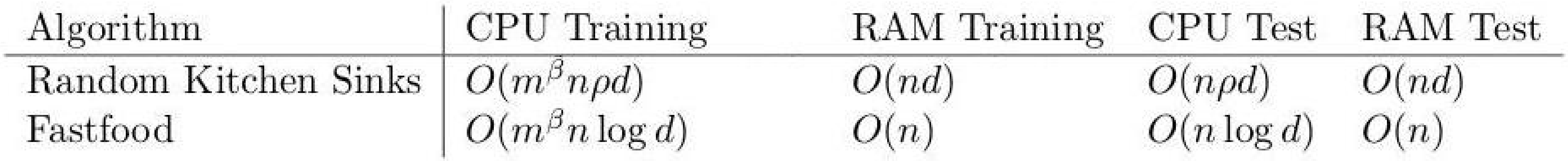
$$f(x) = \sum_{n=1}^{K} \alpha(\omega_n)\phi(x;\omega_n)$$

where Φ are the eigenfunctions of a positive semidefinite kernel parameterized by vectors ω

ω's are drawn from a probability distribution which satisfies a particular criteria and α's are then found by solving a simple convex optimization problem. This is then used to compute features.

## Fastfood

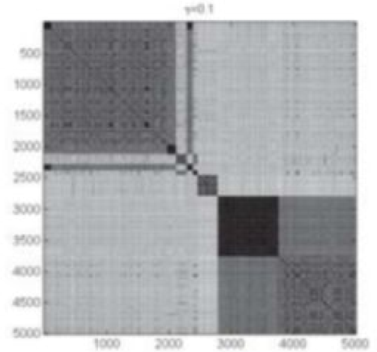
- This is an improvement on Random Kitchen Sinks.
- The random matrix used in Random Kitchen Sinks to compute the explicit mapping is expensive to evaluate and perform computations on.
- The computation is optimized based on the fact Random Gaussian matrices can be approximated by a combination of Hadamard matrix and diagonal Gaussian Matrices which admit efficient techniques for performing computations



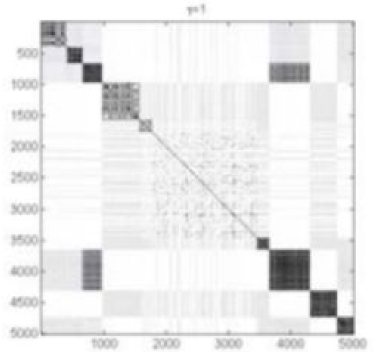
| Algorithm | CPU Training | RAM Training | CPU Test | RAM Test |
|---|---|---|---|---|
| Random Kitchen Sinks | $O(m^{\beta} n \rho d)$ | $O(nd)$ | $O(n \rho d)$ | $O(nd)$ |
| Fastfood | $O(m^{\beta} n \log d)$ | $O(n)$ | $O(n \log d)$ | $O(n)$ |

## Memory Efficient Kernel Approx

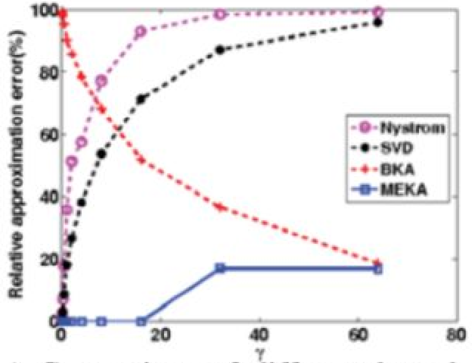
- Current state of the art methods aim to form a low rank approximation $G \approx CCT$ with $C \in R^{nxk}$ and $k<<n$
- Although it is well known that Singular Value Decomposition (SVD) can compute the best rank-k approximation, it often cannot be applied to kernel matrices as it requires the entire 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 Nystrom methods.
- For practically used shift invariant kernels It is observed that the structure of the kernel matrix changes from low-rank to block diagonal (w/o any low rank structure) as □ varies from $0 \rightarrow \infty$. For RBF kernel $G \rightarrow I$ as □$\rightarrow \infty$. Thus even the best rank *k* approximations have extremely large approximation error.
- Low Rank Approximation - Good for small □ , Block Kernel Approximation - Good for large □



(a) RBF kernel matrix with $\gamma = 0.1$



(b) RBF kernel matrix with $\gamma = 1$



(c) Comparison of different kernel approximation methods on various $\gamma$.

*Figure 1.* (a) and (b) show that the structure of the Gaussian kernel matrix $K(\boldsymbol{x}, \boldsymbol{y}) = e^{-\gamma\|\boldsymbol{x}-\boldsymbol{y}\|^2}$ for the covtype data tends to become more block diagonal as $\gamma$ increases(dark regions correspond to large values, while lighter regions correspond to smaller values). Plot (c) shows that low-rank approximations work only for small $\gamma$, and Block Kernel Approximation (BKA) works for large $\gamma$, while our proposed method MEKA works for small as well as large $\gamma$.

## Memory Efficient Kernel Approx

- MEKA exploits both the low rank structure and clustering of data
- A simple Block Kernel Approximation (BKA) proceeds as follows
  - Obtain a good partition of the data using k-means $V_1, V_2 \ldots\ldots V_c$ where each $V_s$ is a subset of {1,2...n}
  - The kernel is approximated as $G_{approx} = G^{(1,1)} \oplus G^{(2,2)} \oplus \ldots\ldots G^{(c,c)}$
  - Thus $G_{approx}^{(s,s)} = G^{(s,s)}$ and $G_{approx}^{(s,t)} = 0$ for $s \neq t$
- MEKA calculates the low rank approximation for each $G^{(s,s)}$ using Nystrom method. We assume the same rank *k* for each block. Thus $G^{(s,s)}_{approx} \approx W^{(s)} L^{(s,s)} W^{(s)T}$
- $G_{approx} \approx WLW^T$ where $W = W^{(1,1)} \oplus W^{(2,2)} \oplus \ldots\ldots W^{(c,c)}$ , L is a link matrix of $c^2$ blocks of size $k^2$ each where each $L^{(s,t)}$ captures the interaction between the sth and tth clusters thus $G^{(s,s)}_{approx} \approx W^{(s)} L^{(s,t)} W^{(t)T}$
- Only $O(k^2)$ memory is needed for each off diagonal block $\Rightarrow O(nk + (ck)^2)$ in total

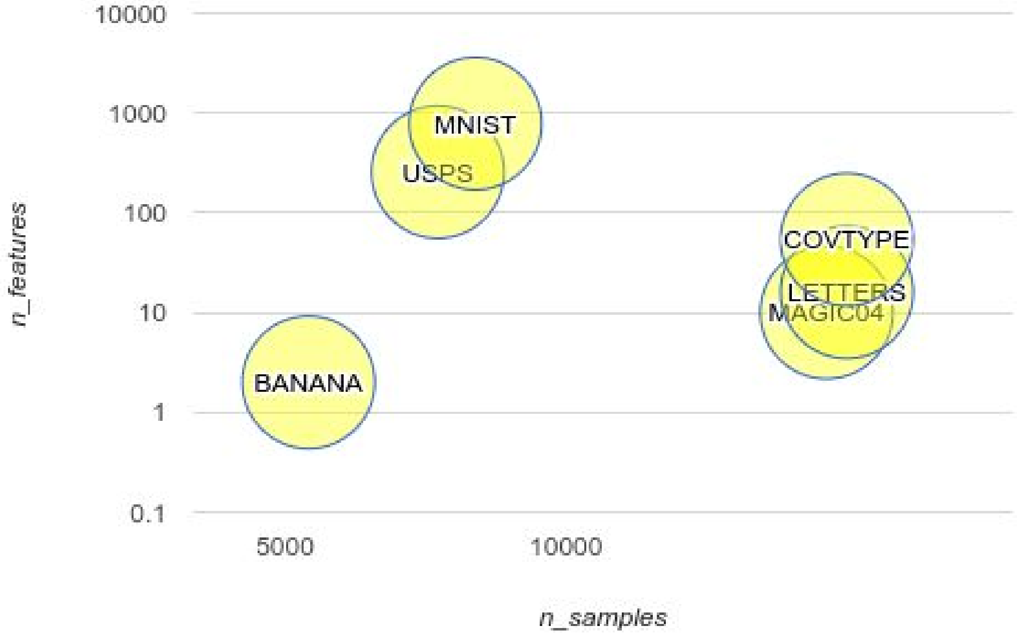
## Memory Efficient Kernel Approx

- MEKA exploits both the low rank structure and clustering of data
- A simple Block Kernel Approximation (BKA) proceeds as follows
  - Obtain a good partition of the data using k-means $V_1, V_2 \ldots\ldots V_c$ where each $V_s$ is a subset of {1,2…n}
  - The kernel is approximated as $G_{approx} = G^{(1,1)} \oplus G^{(2,2)} \oplus \ldots\ldots G^{(c,c)}$
  - Thus $G_{approx}^{(s,s)} = G^{(s,s)}$ and $G_{approx}^{(s,t)} = 0$ for $s \neq t$
- MEKA calculates the low rank approximation for each $G^{(s,s)}$ using Nystrom method. We assume the same rank *k* for each block. Thus $G^{(s,s)}_{approx} \approx W^{(s)} L^{(s,s)} W^{(s)T}$
- $G_{approx} \approx WLW^T$ where $W = W^{(1,1)} \oplus W^{(2,2)} \oplus \ldots\ldots W^{(c,c)}$ , L is a link matrix of $c^2$ blocks of size $k^2$ each where each $L^{(s,t)}$ captures the interaction between the sth and tth clusters thus $G^{(s,s)}_{approx} \approx W^{(s)} L^{(s,t)} W^{(t)T}$
- Only $O(k^2)$ memory is needed for each off diagonal block $\Rightarrow O(nk+(ck)^2)$ in total

## Fourier Online Grad. Des.

- Online method for learning a separating hyperplane
- Uses random fourier features along with a perceptron

## Experimental Results



## Experimental Results

| Dataset | Size | RBF SVM | | | Nystrom | | | Linear SVM | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Train | Test | Accuracy | Train | Test | Accuracy | Train | Test | Accuracy |
| banana | 5300 x 2 | 0.2048 | 0.0428 | 0.902 | 0.0755 | 0.0050 | 0.904 | 0.0433 | 0.0002 | 0.556 |
| magic04 | 19020 x 10 | 6.9772 | 1.4669 | 0.823 | 0.3721 | 0.0093 | 0.825 | 1.3654 | 0.0005 | 0.790 |
| letters | 20000 x 16 | 4.9477 | 3.8674 | 0.924 | 2.2720 | 0.0184 | 0.859 | 11.5490 | 0.0023 | 0.705 |
| usps | 7291 x 256 | 3.3575 | 1.4054 | 0.977 | 0.3289 | 0.0093 | 0.943 | 3.2111 | 0.0028 | 0.928 |
| covtype | 20000 x 54 | 35.5756 | 8.6684 | 0.528 | 0.5469 | 0.0049 | 0.677 | 24.2727 | 0.0015 | 0.656 |
| mnist | 8000 x 784 | 13.9363 | 5.7849 | 0.952 | 1.5239 | 0.0713 | 0.939 | 16.1708 | 0.0088 | 0.854 |

## Experimental Results

| Dataset | Size | RBF SVM | | | Random Fourier Features | | | Linear SVM | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Train | Test | Accuracy | Train | Test | Accuracy | Train | Test | Accuracy |
| banana | 5300 x 2 | 0.2048 | 0.0428 | 0.902 | 0.1163 | 0.0065 | 0.896 | 0.0433 | 0.0002 | 0.556 |
| magic04 | 19020 x 10 | 6.9772 | 1.4669 | 0.823 | 0.5448 | 0.0301 | 0.818 | 1.3654 | 0.0005 | 0.790 |
| letters | 20000 x 16 | 4.9477 | 3.8674 | 0.924 | 7.0539 | 0.1765 | 0.894 | 11.5490 | 0.0023 | 0.705 |
| usps | 7291 x 256 | 3.3575 | 1.4054 | 0.977 | 0.2156 | 0.0015 | 0.167 | 3.2111 | 0.0028 | 0.928 |
| covtype | 20000 x 54 | 35.5756 | 8.6684 | 0.528 | 1.3632 | 0.0175 | 0.676 | 24.2727 | 0.0015 | 0.656 |
| mnist | 8000 x 784 | 13.9363 | 5.7849 | 0.952 | 0.3546 | 0.0078 | 0.107 | 16.1708 | 0.0088 | 0.854 |

## Experimental Results

| Dataset | Size | RBF SVM | | | Random Kitchen Sinks | | | Linear SVM | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Train | Test | Accuracy | Train | Test | Accuracy | Train | Test | Accuracy |
| banana | 5300 x 2 | 0.2048 | 0.0428 | 0.902 | 0.0663 | 0.0053 | 0.901 | 0.0433 | 0.0002 | 0.556 |
| magic04 | 19020 x 10 | 6.9772 | 1.4669 | 0.823 | 0.4969 | 0.0288 | 0.801 | 1.3654 | 0.0005 | 0.790 |
| letters | 20000 x 16 | 4.9477 | 3.8674 | 0.924 | 8.3801 | 0.1651 | 0.691 | 11.5490 | 0.0023 | 0.705 |
| usps | 7291 x 256 | 3.3575 | 1.4054 | 0.977 | 0.1919 | 0.0023 | 0.163 | 3.2111 | 0.0028 | 0.928 |
| covtype | 20000 x 54 | 35.5756 | 8.6684 | 0.528 | 4.5047 | 0.1689 | 0.746 | 24.2727 | 0.0015 | 0.656 |
| mnist | 8000 x 784 | 13.9363 | 5.7849 | 0.952 | 0.237 | 0.0069 | 0.106 | 16.1708 | 0.0088 | 0.854 |

## Experimental Results

| Dataset | Size | RBF SVM | | | Fastfood | | | Linear SVM | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Train | Test | Accuracy | Train | Test | Accuracy | Train | Test | Accuracy |
| banana | 5300 x 2 | 0.2048 | 0.0428 | 0.902 | 0.09021 | 0.00945 | 0.89160 | 0.0433 | 0.0002 | 0.556 |
| magic04 | 19020 x 10 | 6.9772 | 1.4669 | 0.823 | 0.5287 | 0.0346 | 0.820 | 1.3654 | 0.0005 | 0.790 |
| letters | 20000 x 16 | 4.9477 | 3.8674 | 0.924 | 6.0679 | 0.1738 | 0.909 | 11.5490 | 0.0023 | 0.705 |
| usps | 7291 x 256 | 3.3575 | 1.4054 | 0.977 | 2.1740 | 0.0365 | 0.149 | 3.2111 | 0.0028 | 0.928 |
| covtype | 20000 x 54 | 35.5756 | 8.6684 | 0.528 | 4.3250 | 0.1898 | 0.741 | 24.2727 | 0.0015 | 0.656 |
| mnist | 8000 x 784 | 13.9363 | 5.7849 | 0.952 | 5.9429 | 0.1703 | 0.104 | 16.1708 | 0.0088 | 0.854 |

## Experimental Results

| Dataset | Size | RBF SVM | | | Fourier Online Gradient Descent | | | Linear SVM | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Train | Test | Accuracy | Train | Test | Accuracy | Train | Test | Accuracy |
| banana | 5300 x 2 | 0.2048 | 0.0428 | 0.902 | 0.0244 | 0.0074 | 0.8498 | 0.0433 | 0.0002 | 0.556 |
| magic04 | 19020 x 10 | 6.9772 | 1.4669 | 0.823 | 0.4097 | 0.1475 | 0.7947 | 1.3654 | 0.0005 | 0.790 |
| letters | 20000 x 16 | 4.9477 | 3.8674 | 0.924 | 0.6937 | 0.1618 | 0.8013 | 11.5490 | 0.0023 | 0.705 |
| usps | 7291 x 256 | 3.3575 | 1.4054 | 0.977 | 0.0189 | 0.0042 | 0.1099 | 3.2111 | 0.0028 | 0.928 |
| covtype | 20000 x 54 | 35.5756 | 8.6684 | 0.528 | 3.6581 | 0.7955 | 0.6824 | 24.2727 | 0.0015 | 0.656 |
| mnist | 8000 x 784 | 13.9363 | 5.7849 | 0.952 | 0.0311 | 0.0038 | 0.100 | 16.1708 | 0.0088 | 0.854 |

## References

[1] Cristianini, N. (n.d.). Kernel Methods for General Pattern Analysis. Reading. Retrieved from http://www.kernel-methods.net/tutorials/KMtalk.pdf

[2] A. Rahimi and B. Recht, “Random Features for Large-Scale Kernel Machines,” in Neural Information Processing Systems, 2007, pp. 1177–1184.

[3] S. Si, C. J. Hsieh, and I. S. Dhillon, “Memory efficient kernel approximation,” in 31st International Conference on Machine Learning, ICML 2014, vol. 2, International Machine Learning Society (IMLS), 2014, pp. 1057–1069.

[4] A. Rahimi and B. Recht, “Weighted Sums of Random Kitchen Sinks: Replacing minimization with randomization in learning.”

[5] J. Lu, S. C. H. Hoi, J. Wang, P. Zhao, and Z.-Y. Liu, “Large Scale Online Kernel Learning,” J. Mach. Learn. Res., vol. 17, pp. 1–43, 2016.

[6] Q. V. Le, T. Sarlos, and A. J. Smola, “Fastfood: Approximate Kernel Expansions in Loglinear Time,” 2014.

[7] T. Yang, Y.-F. Li, M. Mahdavi, R. Jin, and Z.-H. Zhou, “Nyström Method vs Random Fourier Features: A Theoretical and Empirical Comparison.”

# Thanks!

## Any *questions* ?