

# Random Projections and Dimension Reduction

Rishi Advani<sup>1</sup>   Madison Crim<sup>2</sup>   Sean O'Hagan<sup>3</sup>

<sup>1</sup>Cornell University

<sup>2</sup>Salisbury University

<sup>3</sup>University of Connecticut

Summer@ICERM, July 2020



# Acknowledgements

Thank you to our organizers, Akil Narayan and Yanlai Chen, along with our TAs, Justin Baker and Liu Yang, for supporting us throughout this program



# Introduction

During this talk, we will focus on the use of randomness in two main areas:

- low-rank approximation
- kernel methods



# Table of Contents

## 1 Low-rank Approximation

- Johnson-Lindenstrauss Lemma
- Interpolative Decomposition
- Singular Value Decomposition
- SVD/ID Performance
- Eigenfaces

## 2 Kernel Methods

- Kernel Methods
- Kernel PCA
- Kernel SVM



# Johnson-Lindenstrauss Lemma

If we have  $n$  data points in  $\mathbb{R}^d$ , there exists a linear map into  $\mathbb{R}^k$ ,  $k < d$ , such that pairwise distances between data points can be preserved up to an  $\epsilon$  tolerance, provided  $k > C\epsilon^{-2} \log n$ , where  $C \approx 24$  [JL84]. The proof follows three steps [Mic09]:

- Define a random linear map  $f: \mathbb{R}^d \rightarrow \mathbb{R}^k$  by  $f(\mathbf{u}) = \frac{1}{\sqrt{k}} R \cdot \mathbf{u}$ , where  $R \in \mathbb{R}^{k \times d}$  is drawn elementwise from a standard normal distribution.
- If  $\mathbf{u} \in \mathbb{R}^d$ , show  $\mathbb{E}[\|f(\mathbf{u})\|_2^2] = \|\mathbf{u}\|_2^2$ .
- Show that the random variable  $\|f(\mathbf{u})\|_2^2$  concentrates around  $\|\mathbf{u}\|_2^2$ , and construct a union bound over all pairwise distances.



# Johnson-Lindenstrauss Lemma: Demonstration

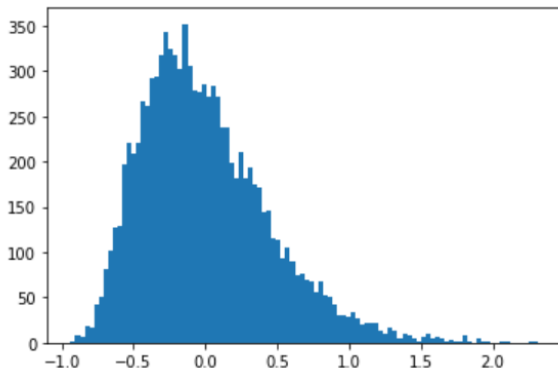


Figure: Histogram of  $\|\mathbf{u}\|_2^2 - \|f(\mathbf{u})\|_2^2$  for a fixed  $\mathbf{u} \in \mathbb{R}^{1000}$ ,  $f(\mathbf{u}) \in \mathbb{R}^{10}$



# Table of Contents

## 1 Low-rank Approximation

- Johnson-Lindenstrauss Lemma
- **Interpolative Decomposition**
- Singular Value Decomposition
- SVD/ID Performance
- Eigenfaces

## 2 Kernel Methods

- Kernel Methods
- Kernel PCA
- Kernel SVM



# Deterministic Interpolative Decomposition

Given a matrix  $A \in \mathbb{R}^{m \times n}$ , we can compute an interpolative decomposition (ID), a low-rank matrix approximation that uses  $A$ 's own columns [Yin+18]. The ID can be computed using the column-pivoted  $QR$  factorization:

$$AP = QR.$$

To obtain our low-rank approximation, we form the submatrix  $Q_k$  using the first  $k$  columns of  $Q$ . We then have the approximation

$$A \approx Q_k Q_k^* A,$$

which gives us a particular rank- $k$  projection of  $A$ .





# Randomized Interpolative Decomposition

We introduce a new method to compute randomized ID, by taking a subset  $S$  of  $p > k$  distinct, randomly-selected columns from the  $n$  columns of  $A$ . The algorithm then performs the column-pivoted  $QR$  factorization on the submatrix:

$$A_{(:,S)}P = QR$$

Accordingly we have the following rank  $k$  projection of  $A$ :

$$A \approx Q_k Q_k^* A,$$

where  $Q_k$  is the submatrix formed by the first  $k$  columns of  $Q$ .



# Table of Contents

## 1 Low-rank Approximation

- Johnson-Lindenstrauss Lemma
- Interpolative Decomposition
- **Singular Value Decomposition**
- SVD/ID Performance
- Eigenfaces

## 2 Kernel Methods

- Kernel Methods
- Kernel PCA
- Kernel SVM



# Deterministic Singular Value Decomposition

- Recall the singular value decomposition of a matrix [16],

$$A_{m \times n} = U_{m \times m} \Sigma_{m \times n} V_{n \times n}^*,$$

where  $U$  and  $V$  are orthogonal matrices, and  $\Sigma$  is a rectangular diagonal matrix with positive diagonal entries  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r$ , where  $r$  is the rank of the matrix  $A$ .

- The  $\sigma_i$ s are called the singular values of  $A$ .



# Randomized Singular Value Decomposition

Utilizing ideas from [HMT09], our algorithm executes the following steps to compute the randomized SVD:

- 1 Construct a  $n \times k$  random Gaussian matrix  $\Omega$
- 2 Form  $Y = A\Omega$
- 3 Construct a matrix  $Q$  whose columns form an orthonormal basis for the column space of  $Y$
- 4 Set  $B = Q^*A$
- 5 Compute the SVD:  $B = U'\Sigma V^*$
- 6 Construct the SVD approximation:  $A \approx QQ^*A = QB = QU'\Sigma V^*$



# Table of Contents

## 1 Low-rank Approximation

- Johnson-Lindenstrauss Lemma
- Interpolative Decomposition
- Singular Value Decomposition
- SVD/ID Performance
- Eigenfaces

## 2 Kernel Methods

- Kernel Methods
- Kernel PCA
- Kernel SVM



# Results - Testing $620 \times 187500$ Matrix

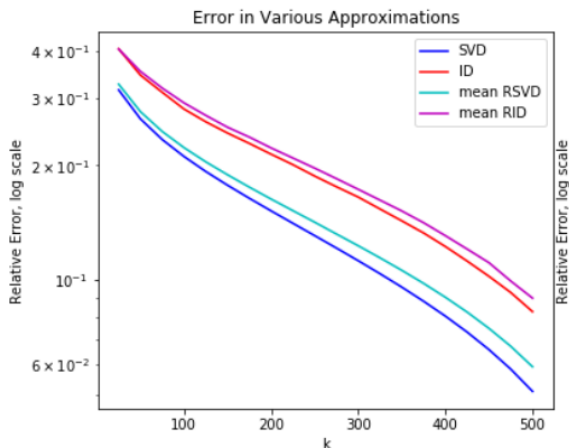


Figure: Error Relative to Original Data



# Results - Testing $620 \times 187500$ Matrix

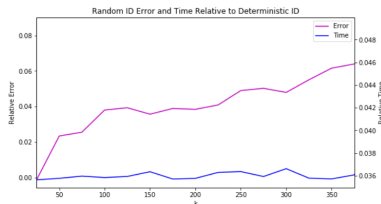


Figure: Random ID Error and Time Relative to Deterministic ID



Figure: Random SVD Error and Time Relative to Deterministic SVD



# Table of Contents

## 1 Low-rank Approximation

- Johnson-Lindenstrauss Lemma
- Interpolative Decomposition
- Singular Value Decomposition
- SVD/ID Performance
- Eigenfaces

## 2 Kernel Methods

- Kernel Methods
- Kernel PCA
- Kernel SVM





# Eigenfaces

- Using ideas from [BKP15], our eigenfaces experiment is based on the LFW dataset [Hua+07]. This dataset contains more than 13,000 RGB images of faces, where each image has dimensions  $250 \times 250$ .
- We can flatten each image to represent it as vector of length  $250 \cdot 250 \cdot 3 = 187500$ .
- In our experiment we will only use 620 images from the LFW dataset. This gives us a data matrix  $A$  of size  $187500 \times 620$ .
- We then can perform SVD on the mean-subtracted columns of  $A$ .

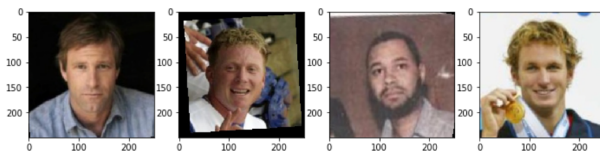


Figure: Original LFW Images



# Image Results

We obtain the following eigenfaces from the columns of the matrix  $U$ :

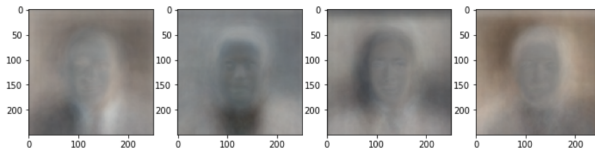


Figure: Eigenfaces Obtained using Deterministic SVD

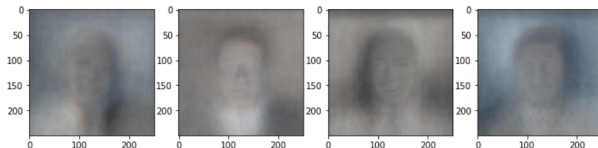


Figure: Eigenfaces Obtained using Randomized SVD



# Table of Contents

## 1 Low-rank Approximation

- Johnson-Lindenstrauss Lemma
- Interpolative Decomposition
- Singular Value Decomposition
- SVD/ID Performance
- Eigenfaces

## 2 Kernel Methods

- Kernel Methods
- Kernel PCA
- Kernel SVM



- Kernel methods work by mapping the data into a high-dimensional space to add more structure and encourage linear separability.
- Suppose we have a feature map  $\phi: \mathbb{R}^n \rightarrow \mathbb{R}^m$ ,  $m > n$ .
- The 'kernel trick' is based on the observation that we only need the inner products of vectors in the feature space, not the explicit high-dimensional mappings.

$$k(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$$

- Ex. Gaussian/RBF Kernel:  $k(\mathbf{x}, \mathbf{y}) = \exp(-\gamma \|\mathbf{x} - \mathbf{y}\|_2^2)$
- Kernel methods include kernel PCA, kernel SVM, and more.



# Randomized Fourier Features Kernel

We can sample random Fourier features to approximate a kernel [RR08]. Let  $k(\mathbf{x}, \mathbf{y})$  denote our kernel, and  $p(\mathbf{w})$  the probability distribution corresponding to the inverse Fourier transform of  $k$ .

$$\begin{aligned} k(\mathbf{x}, \mathbf{y}) &= \int_{\mathbb{R}^d} p(\mathbf{w}) e^{-j\mathbf{w}^T(\mathbf{x}-\mathbf{y})} d\mathbf{w} \\ &\approx \frac{1}{m} \sum_{i=1}^m \cos(\mathbf{w}_i^T \mathbf{x} + b_i) \cos(\mathbf{w}_i^T \mathbf{y} + b_i), \end{aligned}$$

where  $\mathbf{w}_i \sim p(\mathbf{w})$ ,  $b_i \sim \text{Uniform}(0, 2\pi)$ . For a given  $m$ , define

$$z(\mathbf{x}) = \sum_{i=1}^m \cos(\mathbf{w}_i^T \mathbf{x} + b_i)$$

to yield the approximation  $k(\mathbf{x}, \mathbf{y}) \approx \frac{1}{m} z(\mathbf{x}) z(\mathbf{y})^T$  [Lop+14].



# Table of Contents

## 1 Low-rank Approximation

- Johnson-Lindenstrauss Lemma
- Interpolative Decomposition
- Singular Value Decomposition
- SVD/ID Performance
- Eigenfaces

## 2 Kernel Methods

- Kernel Methods
- Kernel PCA
- Kernel SVM



# Data for Kernel PCA Experiments

To test kernel PCA methods, we use a dataset that is not linearly separable — a cloud of points surrounded by a circle:

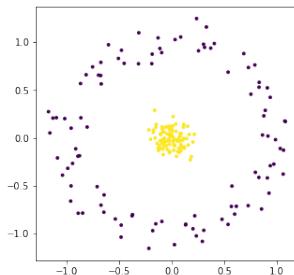


Figure: Data used to test kernel PCA methods



# Randomized Kernel PCA Results

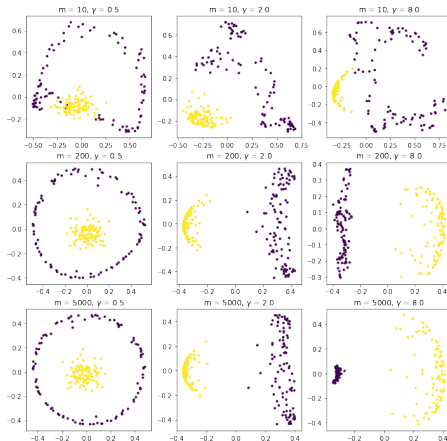


Figure: Random Fourier features KPCA results





# Table of Contents

## 1 Low-rank Approximation

- Johnson-Lindenstrauss Lemma
- Interpolative Decomposition
- Singular Value Decomposition
- SVD/ID Performance
- Eigenfaces

## 2 Kernel Methods

- Kernel Methods
- Kernel PCA
- Kernel SVM



- We may also use kernel methods for support vector machines (SVM).
- The goal of an SVM is to find the  $(d - 1)$ -hyperplane that best separates two clusters of  $d$ -dimensional data points.
- In two dimensions, this is a line separating two clusters of points in a plane.
- Using the kernel trick, we can project inseparable points into a higher dimension and run an SVM algorithm on the resulting points.



# Randomized Kernel SVM

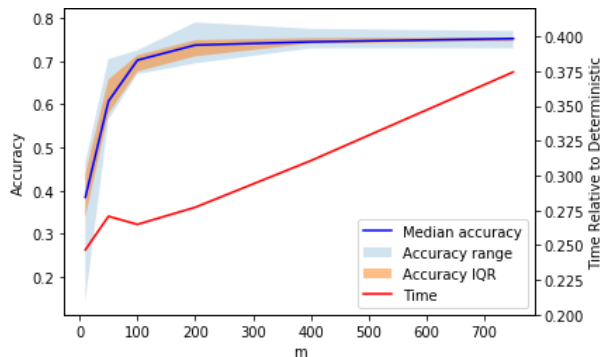


Figure: Randomized Kernel SVM Accuracy and time results as  $m$  varies



# Comparison of Deterministic and Randomized Kernel SVM

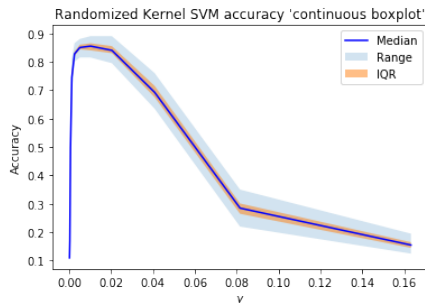
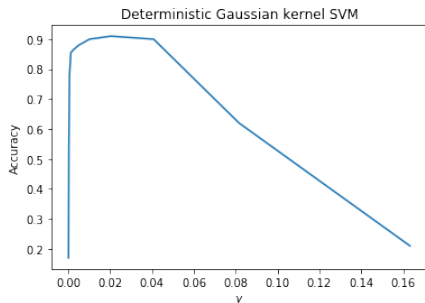
Using the MNIST dataset [LC10] we test 10000 images (784 features), for a **fixed**  $\gamma$ :

- Deterministic Kernel
  - Accuracy: 0.9195
  - Time: 37.99s
- Randomized Kernel
  - Accuracy: Mean: 0.891, St. dev. 0.0042  
Min: 0.881, Max: 0.9005
  - Mean Time: 2.14s



# Comparison of Deterministic and Randomized Kernel SVM

On 1000 MNIST images, we plot the accuracies of the deterministic and random kernel SVMs as  $\gamma$  varies:



# Application of Randomized Kernel SVM: Grid Search

Testing 100  $\gamma$  values to identify the best one:

- Deterministic Kernel, Series: 133.03s
- Randomized Kernel, Series: 78.97s
- Randomize Kernel, Parallel: 41.18s
- Best  $\gamma$  value obtained from randomized method corresponds with either best or second best deterministic  $\gamma$  (3 trials)

$$\hat{\mathbf{K}} = \frac{1}{m} \mathbf{z}(\mathbf{X}) \mathbf{z}(\mathbf{X})^T$$



# Takeaways

- When using large datasets, randomized algorithms are able to maintain most of the accuracy of their deterministic counterpart, while offering a huge reduction in computational cost
- These algorithms are useful for matrix factorization/decomposition as well as for kernel approximation



# References I



*ICERM Logo*. ICERM. URL: <https://icerm.brown.edu>.



*The Singular Value Decomposition (SVD)*. 2016. URL: [https://math.mit.edu/classes/18.095/2016IAP/lec2/SVD\\_Notes.pdf](https://math.mit.edu/classes/18.095/2016IAP/lec2/SVD_Notes.pdf).



Brunton, Kutz, and Proctor. *Eigenfaces Example*. 2015. URL: <http://faculty.washington.edu/sbrunton/me565/pdf/L29secure.pdf>.



Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*. 2009. arXiv: 0909.4061 [math.NA].





# References II



Gary B. Huang et al. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. Tech. rep. 07-49. University of Massachusetts, Amherst, Oct. 2007.



William Johnson and Joram Lindenstrauss. “Extensions of Lipschitz maps into a Hilbert space”. In: *Contemporary Mathematics* 26 (Jan. 1984), pp. 189–206. DOI: [10.1090/conm/026/737400](https://doi.org/10.1090/conm/026/737400).



Yann LeCun and Corinna Cortes. “MNIST handwritten digit database”. In: (2010). URL: <http://yann.lecun.com/exdb/mnist/>.



David Lopez-Paz et al. *Randomized Nonlinear Component Analysis*. 2014. arXiv: 1402.0119 [stat.ML].



# References III



Mahoney Michael. *The Johnson-Lindenstrauss Lemma*. Sept. 2009. URL: <https://cs.stanford.edu/people/mmahoney/cs369m/Lectures/lecture1.pdf>.



Ali Rahimi and Benjamin Recht. *Random Features for Large-Scale Kernel Machines*. Ed. by J. C. Platt et al. 2008. URL: <http://papers.nips.cc/paper/3182-random-features-for-large-scale-kernel-machines.pdf>.



Lexing Ying et al. *Interpolative Decomposition and its Applications in Quantum Chemistry*. 2018. URL: [https://www.ki-net.umd.edu/activities/presentations/9\\_871\\_cscamm.pdf](https://www.ki-net.umd.edu/activities/presentations/9_871_cscamm.pdf).



To explore more visit our website at the following link:  
<https://rishi1999.github.io/random-projections/>

