10-605/10-805: Machine Learning with Large Datasets

FALL 2022

Kernel Approximation

Outline

1. Background: Kernel Methods
2. Kernel Ridge Regression
3. SVD
4. Large-Scale Kernel Methods (via Approximate SVD)

RECALL

Empirical risk minimization

$$\hat{f}_n = \arg\min_{f \in F} \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i)$$

- A popular approach in supervised ML
- Given a loss ℓ and data (x_1, y_1), … (x_n, y_n), we estimate a predictor *f* by minimizing the *empirical risk*
- We typically restrict this predictor to lie in some class, *F*
  - Could reflect our prior knowledge about the task
  - Or may be for computational convenience

Question: how should we select our function class, *F*, and our loss function, ℓ ??

LINEAR REGRESSION VIA

Empirical risk minimization

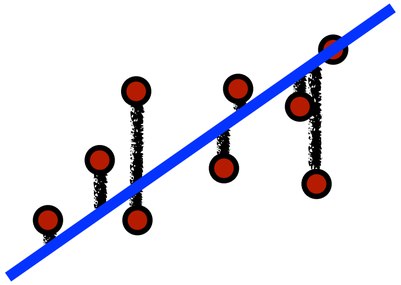
$$\hat{f}_n = \arg\min_{f \in F} \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i)$$

linear (actually, affine) functions

square loss

- A popular approach in supervised ML
- Given a loss ℓ and data (x_1, y_1), … (x_n, y_n), we estimate a predictor *f* by minimizing the *empirical risk*
- We typically restrict this predictor to lie in some class, *F*
  - Could reflect our prior knowledge about the task
  - Or may be for computational convenience

Question: how should we select our function class, *F*, and our loss function, ℓ ??



Given *n* training points with *k* features, we define:

- $\mathbf{X} \in \mathbb{R}^{n \times k}$: matrix storing points
- $\mathbf{y} \in \mathbb{R}^{n}$: real-valued labels
- $\hat{\mathbf{y}} \in \mathbb{R}^{n}$: predicted labels, where $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$
- $\mathbf{w} \in \mathbb{R}^{k}$: regression parameters / model to learn

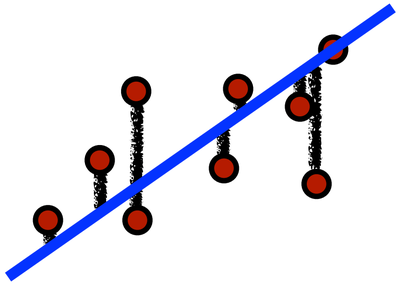
Least Squares Regression: Learn mapping ($\mathbf{w}$) from features to labels that minimizes residual sum of squares:

$$\min_{\mathbf{w}} ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2$$

Equivalent $\min_{\mathbf{w}} \sum_{i=1}^{n} (\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})^2$ by definition of Euclidean norm

Given *n* training points with *k* features, we define:

- $\mathbf{X} \in \mathbb{R}^{n \times k}$: matrix storing points
- $\mathbf{y} \in \mathbb{R}^{n}$: real-valued labels
- $\hat{\mathbf{y}} \in \mathbb{R}^{n}$: predicted labels, where $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$
- $\mathbf{w} \in \mathbb{R}^{k}$: regression parameters / model to learn



Ridge Regression: Learn mapping ($\mathbf{w}$) that minimizes residual sum of squares along with a regularization term:

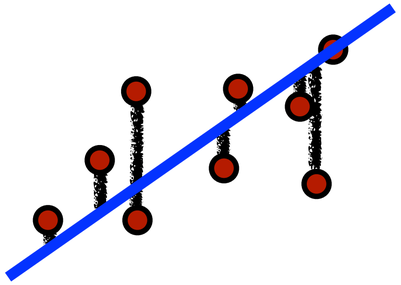
Training Error | Model Complexity

$$\min_{\mathbf{w}} ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 + \lambda||\mathbf{w}||_2^2$$

free parameter trades off between training error and model complexity

Closed-form solution: $\mathbf{w} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_k)^{-1}\mathbf{X}^\top\mathbf{y}$

Why a linear mapping?



Simple

Often works well in practice

Can introduce complexity via feature extraction

Motivation

How to choose nonlinear basis function for regression?

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$$

- $\boldsymbol{\phi}(\cdot)$ maps the original feature vector $\boldsymbol{x}$ to a *new* M-dimensional feature vector

Motivation

How to choose nonlinear basis function for regression?

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$$

- $\boldsymbol{\phi}(\cdot)$ maps the original feature vector $\boldsymbol{x}$ to a *new* M-dimensional feature vector
- We can sidestep the issue of choosing which $\boldsymbol{\phi}(\cdot)$ to use by *equivalently* choosing a *kernel function*

  ➡ kernel function operates on inner products of the observations/data points

Common kernel functions

Polynomial kernel function with degree of d

$$k(\boldsymbol{x}_m, \boldsymbol{x}_n) = (\boldsymbol{x}_m^{\mathrm{T}} \boldsymbol{x}_n + c)^d$$

for $c \geq 0$ and d is a positive integer.

Common kernel functions

Polynomial kernel function with degree of d

$$k(\boldsymbol{x}_m, \boldsymbol{x}_n) = (\boldsymbol{x}_m^{\mathrm{T}} \boldsymbol{x}_n + c)^d$$

for $c \geq 0$ and d is a positive integer.

Gaussian kernel, RBF kernel, or Gaussian RBF kernel

$$k(\boldsymbol{x}_m, \boldsymbol{x}_n) = e^{-\|\boldsymbol{x}_m - \boldsymbol{x}_n\|_2^2 / 2\sigma^2}$$

- Shift-invariant kernel (only depends on difference between two inputs)
- Corresponds to a feature space with *infinite* dimensions (but we can work directly with the original features)

'Kernel trick'

- Many learning methods rely on training and test data only in the form of *inner products*, e.g., regularized least squares, nearest neighbors, SVMs
- We can use a kernel function to introduce nonlinearity, i.e., *"kernelizing"* the methods
  - ➡ allows us to consider complex, high dimensional feature spaces
  - ➡ but, requires expressing training and inference in terms of inner products, e.g., in terms of the kernel matrix, K

$$\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} = \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & k(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_1) & k(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ \vdots & \vdots & \vdots & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_1) & k(\boldsymbol{x}_N, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{pmatrix}$$

TODAY

Kernel methods at scale

$$\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} = \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & k(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_1) & k(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ \vdots & \vdots & \vdots & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_1) & k(\boldsymbol{x}_N, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{pmatrix}$$

- Kernel methods allow us to easily encode data in high-dimensional feature spaces
- But, they come at a cost: storing and operating on the $O(n^2)$ kernel matrix
- This may make kernel methods infeasible for large values of *n*

Question: Can we approximate the kernel matrix with something simpler and more scalable?

Outline

1. Background: Kernel Methods
2. Kernel Ridge Regression
3. SVD
4. Large-Scale Kernel Methods (via Approximate SVD)

Motivation

How to choose nonlinear basis function for regression?

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$$

- $\boldsymbol{\phi}(\cdot)$ maps the original feature vector $\boldsymbol{x}$ to a *new* M-dimensional feature vector
- We can sidestep the issue of choosing which $\boldsymbol{\phi}(\cdot)$ to use by *equivalently* choosing a *kernel function*

Motivation

How to choose nonlinear basis function for regression?

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$$

- $\boldsymbol{\phi}(\cdot)$ maps the original feature vector $\boldsymbol{x}$ to a *new* M-dimensional feature vector
- We can sidestep the issue of choosing which $\boldsymbol{\phi}(\cdot)$ to use by *equivalently* choosing a *kernel function*

Regularized least squares

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_i (y_i - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i))^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

Its solution is given by

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = \sum_i (y_i - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i))(-\boldsymbol{\phi}(\boldsymbol{x}_i)) + \lambda \boldsymbol{w} = 0$$

Solution

The optimal parameter vector is a linear combination of features

$$\boldsymbol{w}^* = \sum_i \frac{1}{\lambda}(y_i - \boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i))\boldsymbol{\phi}(\boldsymbol{x}_i)$$

Solution

The optimal parameter vector is a linear combination of features

$$\boldsymbol{w}^* = \sum_i \frac{1}{\lambda}(y_i - \boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i))\boldsymbol{\phi}(\boldsymbol{x}_i) = \sum_i \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i) = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}$$

Solution

The optimal parameter vector is a linear combination of features

$$\boldsymbol{w}^* = \sum_i \frac{1}{\lambda}(y_i - \boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i))\boldsymbol{\phi}(\boldsymbol{x}_i) = \sum_i \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i) = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}$$

- $\alpha_i = \frac{1}{\lambda}(y_i - \boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i))$
- $\boldsymbol{\Phi}$ is the *design matrix* of *transformed* features
- Its transpose is made of column vectors and is given by

$$\boldsymbol{\Phi}^{\mathrm{T}} = (\boldsymbol{\phi}(\boldsymbol{x}_1)\ \boldsymbol{\phi}(\boldsymbol{x}_2)\ \cdots\ \boldsymbol{\phi}(\boldsymbol{x}_n)) \in \mathbb{R}^{M \times n}$$

Solution

The optimal parameter vector is a linear combination of features

$$\boldsymbol{w}^* = \sum_i \frac{1}{\lambda}(y_i - \boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i))\boldsymbol{\phi}(\boldsymbol{x}_i) = \sum_i \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i) = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}$$

- $\alpha_i = \frac{1}{\lambda}(y_i - \boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i))$
- $\boldsymbol{\Phi}$ is the *design matrix* of *transformed* features
- Its transpose is made of column vectors and is given by

$$\boldsymbol{\Phi}^{\mathrm{T}} = (\boldsymbol{\phi}(\boldsymbol{x}_1)\ \boldsymbol{\phi}(\boldsymbol{x}_2)\ \cdots\ \boldsymbol{\phi}(\boldsymbol{x}_n)) \in \mathbb{R}^{M \times n}$$

Of course, we don't know what $\boldsymbol{\alpha}$ (the vector of α_i) corresponds to, given its dependence on $\boldsymbol{w}^*$!

Dual formulation

Regularized least squares: $J(\boldsymbol{w}) = \frac{1}{2}\sum_n (y_n - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n))^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$

Substitute $\boldsymbol{w}^* = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}$ into $J(\boldsymbol{w})$ to obtain a function of $\boldsymbol{\alpha}$:

$$J(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} - (\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y})^{\mathrm{T}}\boldsymbol{\alpha} + \frac{\lambda}{2}\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}$$

Dual formulation

Regularized least squares: $J(\boldsymbol{w}) = \frac{1}{2}\sum_n (y_n - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n))^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$

Substitute $\boldsymbol{w}^* = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}$ into $J(\boldsymbol{w})$ to obtain a function of $\boldsymbol{\alpha}$:

$$J(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} - (\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y})^{\mathrm{T}}\boldsymbol{\alpha} + \frac{\lambda}{2}\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}$$

The *Gram matrix* or *kernel matrix* — $\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}$ — appears multiple times

$$\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}$$

$$= \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_1) & \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_2) & \cdots & \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_N) \\ \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_1) & \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_2) & \cdots & \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_N) \\ \cdots & \cdots & \cdots & \cdots \\ \boldsymbol{\phi}(\boldsymbol{x}_N)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_1) & \boldsymbol{\phi}(\boldsymbol{x}_N)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_2) & \cdots & \boldsymbol{\phi}(\boldsymbol{x}_N)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_N) \end{pmatrix} \in \mathbb{R}^{n \times n}$$

The derivation of $J(\boldsymbol{\alpha})$

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_n (y - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n))^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

The derivation of $J(\boldsymbol{\alpha})$

$$
\begin{aligned}
J(\boldsymbol{w}) &= \frac{1}{2}\sum_n (y - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n))^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 \\
&= \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 \qquad (\boldsymbol{y}, \boldsymbol{\Phi}\boldsymbol{w} \in \mathbb{R}^N)
\end{aligned}
$$

The derivation of $J(\boldsymbol{\alpha})$

$$
\begin{aligned}
J(\boldsymbol{w}) &= \frac{1}{2}\sum_n (y - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n))^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 \\
&= \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 && (\boldsymbol{y}, \boldsymbol{\Phi}\boldsymbol{w} \in \mathbb{R}^N) \\
&= \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}\|_2^2 && (\boldsymbol{w}^* = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha})
\end{aligned}
$$

The derivation of $J(\boldsymbol{\alpha})$

$$
\begin{aligned}
J(\boldsymbol{w}) &= \frac{1}{2}\sum_n (y - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n))^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 \\
&= \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 && (\boldsymbol{y}, \boldsymbol{\Phi}\boldsymbol{w} \in \mathbb{R}^N) \\
&= \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}\|_2^2 && (\boldsymbol{w}^* = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}) \\
&= \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha}\|_2^2 + \frac{\lambda}{2}\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} && (\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}, \|\boldsymbol{v}\|_2^2 = \boldsymbol{v}^{\top}\boldsymbol{v})
\end{aligned}
$$

The derivation of $J(\boldsymbol{\alpha})$

$$
\begin{aligned}
J(\boldsymbol{w}) &= \frac{1}{2}\sum_n (y - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n))^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 \\
&= \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 && (\boldsymbol{y}, \boldsymbol{\Phi}\boldsymbol{w} \in \mathbb{R}^N) \\
&= \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}\|_2^2 && (\boldsymbol{w}^* = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}) \\
&= \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha}\|_2^2 + \frac{\lambda}{2}\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} && (\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}, \|\boldsymbol{v}\|_2^2 = \boldsymbol{v}^{\top}\boldsymbol{v}) \\
&\propto \frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{K}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\alpha} - \boldsymbol{y}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\alpha} + \frac{\lambda}{2}\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\alpha}
\end{aligned}
$$

Optimal $\boldsymbol{\alpha}$

$$\frac{\partial J(\boldsymbol{\alpha})}{\partial \boldsymbol{\alpha}} = \boldsymbol{K}^2\boldsymbol{\alpha} - \boldsymbol{K}\boldsymbol{y} + \lambda \boldsymbol{K}\boldsymbol{\alpha} = 0$$

Optimal $\boldsymbol{\alpha}$

$$\frac{\partial J(\boldsymbol{\alpha})}{\partial \boldsymbol{\alpha}} = \boldsymbol{K}^2\boldsymbol{\alpha} - \boldsymbol{K}\boldsymbol{y} + \lambda \boldsymbol{K}\boldsymbol{\alpha} = 0$$

which leads to (assuming that $\boldsymbol{K}$ is invertible)

$$\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}$$

Optimal $\boldsymbol{\alpha}$

$$\frac{\partial J(\boldsymbol{\alpha})}{\partial \boldsymbol{\alpha}} = \boldsymbol{K}^2\boldsymbol{\alpha} - \boldsymbol{K}\boldsymbol{y} + \lambda \boldsymbol{K}\boldsymbol{\alpha} = 0$$

which leads to (assuming that $\boldsymbol{K}$ is invertible)

$$\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}$$

- We only need to know $\boldsymbol{K}$ in order to compute $\boldsymbol{\alpha}$!
- Solution doesn't involve $\boldsymbol{\phi}(\cdot)$, but instead inner products $\boldsymbol{\phi}(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_j)$
- This observation will give rise to the use of *kernel functions*

Optimal $\boldsymbol{\alpha}$

$$\frac{\partial J(\boldsymbol{\alpha})}{\partial \boldsymbol{\alpha}} = \boldsymbol{K}^2\boldsymbol{\alpha} - \boldsymbol{K}\boldsymbol{y} + \lambda \boldsymbol{K}\boldsymbol{\alpha} = 0$$

which leads to (assuming that $\boldsymbol{K}$ is invertible)

$$\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}$$

- We only need to know $\boldsymbol{K}$ in order to compute $\boldsymbol{\alpha}$!
- Solution doesn't involve $\boldsymbol{\phi}(\cdot)$, but instead inner products $\boldsymbol{\phi}(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_j)$
- This observation will give rise to the use of *kernel functions*

Note that the initial parameter vector does require knowledge of $\boldsymbol{\Phi}$

$$\boldsymbol{w}^* = \boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}$$

Computing prediction needs only inner products too!

Since $\boldsymbol{w}^* = \boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}$, at test time we compute:

$$\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}\boldsymbol{\phi}(\boldsymbol{x})$$

Computing prediction needs only inner products too!

Since $\boldsymbol{w}^* = \boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}$, at test time we compute:

$$\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}\boldsymbol{\phi}(\boldsymbol{x})$$

$$= \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) \\ \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_N)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) \end{pmatrix}$$

Computing prediction needs only inner products too!

Since $\boldsymbol{w}^* = \boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}$, at test time we compute:

$$\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}\boldsymbol{\phi}(\boldsymbol{x})$$

$$= \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) \\ \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_N)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) \end{pmatrix} = \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{K} + \lambda I)^{-1}\boldsymbol{k}_{\boldsymbol{x}}$$

- We used the property that $(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}$ is symmetric (as $\boldsymbol{K}$ is)
- $\boldsymbol{k}_{\boldsymbol{x}}$ is shorthand notation for the column vector of inner products between training set and test point

Computing prediction needs only inner products too!

Since $\boldsymbol{w}^* = \boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}$, at test time we compute:

$$\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}\boldsymbol{\phi}(\boldsymbol{x})$$

$$= \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) \\ \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_N)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) \end{pmatrix} = \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{K} + \lambda I)^{-1}\boldsymbol{k}_{\boldsymbol{x}}$$

- We used the property that $(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}$ is symmetric (as $\boldsymbol{K}$ is)
- $\boldsymbol{k}_{\boldsymbol{x}}$ is shorthand notation for the column vector of inner products between training set and test point
- To make a prediction we *only need* $\boldsymbol{\phi}(\boldsymbol{x}_n)^T\boldsymbol{\phi}(\boldsymbol{x})$!

Kernel PCA

$$\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} = \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & k(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_1) & k(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ \vdots & \vdots & \vdots & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_1) & k(\boldsymbol{x}_N, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{pmatrix}$$

Given: *n* × *k* matrix of uncentered raw data
Goal: Compute *r* ≪ *k* dimensional representation

Step 1: Compute kernel matrix, K

Step 2: Center K via $\tilde{K} = K - AK - KA + AKA$, where A is matrix containing $\frac{1}{n}$

Step 3: Find top *r* eigenvectors of $\tilde{K} = U\Lambda U^T$ and normalize to find P

Step 4: Compute PCA Scores $Z = \tilde{K}P$

'Kernel trick'

- Many learning methods rely on training and test data only in the form of *inner products*, e.g., regularized least squares, nearest neighbors, SVMs
- We can use a kernel function to introduce nonlinearity, i.e., *"kernelizing"* the methods

Why use kernel functions?

Can define kernel matrix without specifying $\phi(\cdot)$

$$\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} = \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & k(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_1) & k(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ \vdots & \vdots & \vdots & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_1) & k(\boldsymbol{x}_N, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{pmatrix}$$

Question: Can we approximate the kernel matrix with something simpler and more scalable?

Why do we care?

Kernel-based algorithms

- $n \times n$ similarity matrix ($\boldsymbol{K}$)
- flexibility in choice of kernel (similarity measure)
- e.g., SVMs, Kernel Ridge Regression, Kernel PCA
- Recall for KRR: $\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{y}$

Why do we care?

Kernel-based algorithms

- $n \times n$ similarity matrix ($\boldsymbol{K}$)
- flexibility in choice of kernel (similarity measure)
- e.g., SVMs, Kernel Ridge Regression, Kernel PCA
- Recall for KRR: $\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{y}$

Expensive for large-scale problems

- $O(n^2)$ space
- at least $O(n^2)$ time to operate on $\boldsymbol{K}$
- $O(n^3)$ to invert

Example: we want to invert a large matrix, e.g., $n = 18M$

- $\boldsymbol{K}$ requires roughly 1300 TB
- Requires 40K 32GB RAM machines
- Even if we can store it, $O(n^3)$ time is not possible

Possible solution: Encourage implicit sparsity via low-rank approximation

- Some data are naturally approximately low rank (e.g., images)
- Low-rankedness can be viewed as a form of regularization
- Ultimately it's a modeling assumption; common in practice, but not guaranteed to be a good one

Key technical tool: SVD!

Outline

1. Background: Kernel Methods
2. Kernel Ridge Regression
3. SVD
4. Large-Scale Kernel Methods (via Approximate SVD)

RECALL

Singular value decomposition (SVD)

Every matrix has the following decomposition:

SVD

Let $X \in \mathbb{R}^{n \times m}$ then

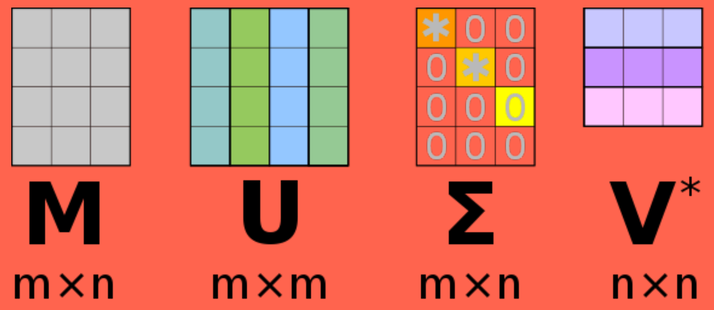
$$X = U \Sigma V^\top,$$

where $U \in \mathbb{R}^{n \times n}$, $V \in \mathbb{R}^{m \times m}$ are orthogonal matrices (*i.e.* $U^\top = U^{-1}$) and $\Sigma \in \mathbb{R}^{n \times m}$ is a diagonal matrix with *singular values* of X denoted by σ_i appearing by non-increasing order: $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{\min(m,n)} \geq 0$.

- The square singular values of X are the eigenvalues of the matrix $XX^\top$ or $X^\top X$, i.e., $\sigma_i(X) = \sqrt{\lambda_i(XX^\top)} = \sqrt{\lambda_i(X^\top X)}$

RECALL

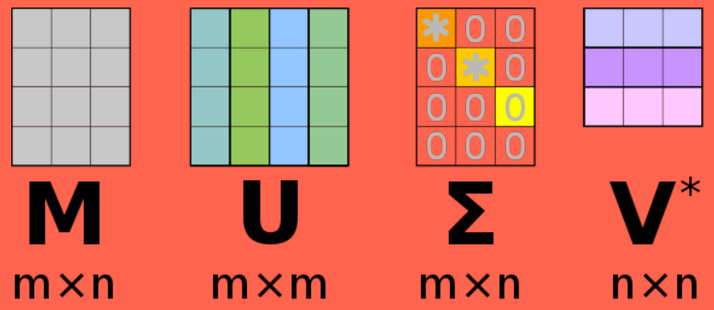
SVD vs Compact SVD vs Truncated SVD



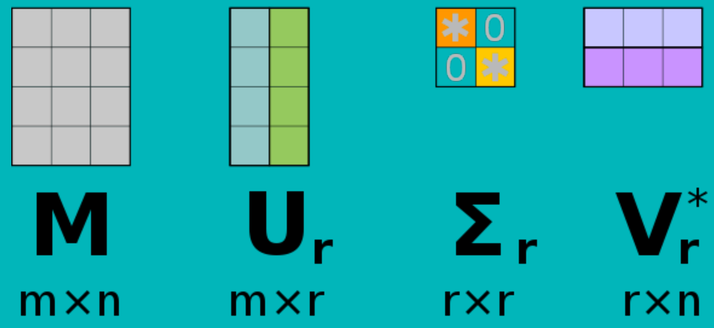
SVD: U, V orthogonal,
Σ rectangular diagonal matrix

RECALL

SVD vs Compact SVD vs Truncated SVD



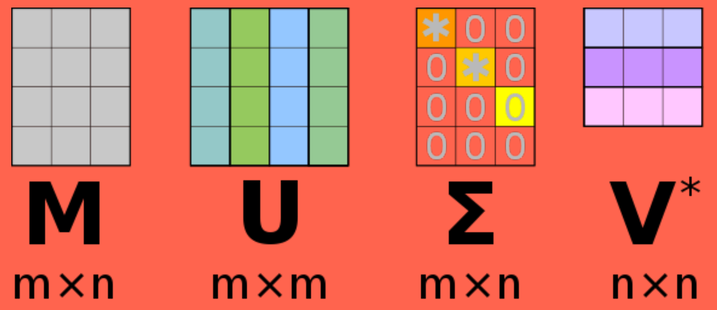
SVD: U, V orthogonal,
Σ rectangular diagonal matrix



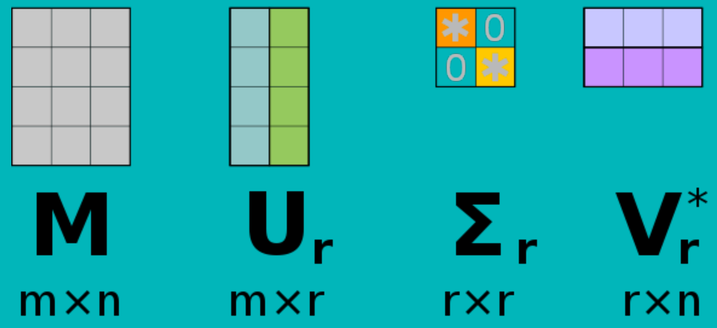
Compact SVD: U, V semi-orthogonal,
Σ square diagonal matrix with non-zero SVs

RECALL

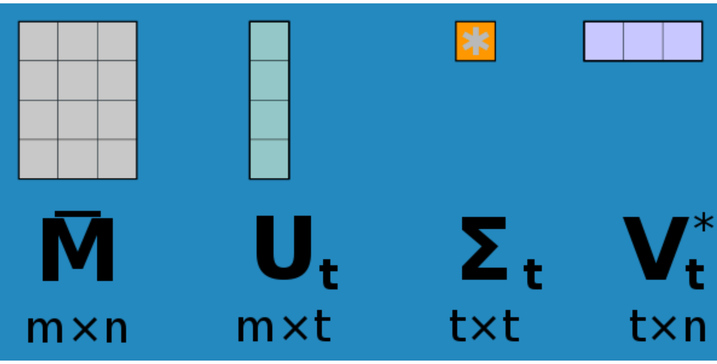
SVD vs Compact SVD vs Truncated SVD



SVD: U, V orthogonal,
Σ rectangular diagonal matrix



Compact SVD: U, V semi-orthogonal,
Σ square diagonal matrix with non-zero SVs



Truncated SVD: U, V semi-orthogonal,
Σ square diagonal matrix with *t* largest SVs

Properties of the matrix $\boldsymbol{K}$

- Symmetric $$K_{ij} = \phi(\mathbf{x_i})^\top \phi(\mathbf{x_j}) = \phi(\mathbf{x_j})^\top \phi(\mathbf{x_i}) = K_{ji}$$

- Positive semidefinite: for any vector $\boldsymbol{a}$

$$\boldsymbol{a}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{a} = (\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{a})^{\mathrm{T}} (\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{a}) \geq 0$$

Connections between $\boldsymbol{\Phi}$, C, and K

Assuming $\boldsymbol{\Phi} \in \mathbb{R}^{n \times M}$ is mean centered:

- Covariance matrix: $\mathbf{C} = \boldsymbol{\Phi}^\top \boldsymbol{\Phi} \in \mathbb{R}^{M \times M}$
- Kernel matrix: $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\top \in \mathbb{R}^{n \times n}$
- *Rank (number of non-zero singular values) of all three matrices are the same*

Why? Let's consider SVD

- $\boldsymbol{\Phi} = \mathbf{U}\boldsymbol{\Sigma}^{1/2}\mathbf{V}^\top$
- $\mathbf{C} = \boldsymbol{\Phi}^\top\boldsymbol{\Phi} = \mathbf{V}\boldsymbol{\Sigma}^{1/2}\mathbf{U}^\top\mathbf{U}\boldsymbol{\Sigma}^{1/2}\mathbf{V}^\top = \mathbf{V}\boldsymbol{\Sigma}\mathbf{V}^\top$
- $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\top = \mathbf{U}\boldsymbol{\Sigma}^{1/2}\mathbf{V}^\top\mathbf{V}\boldsymbol{\Sigma}^{1/2}\mathbf{U}^\top = \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^\top$
- C and K have the same non-zero singular values

Note: variants of the SVD do bookkeeping differently for zero-valued singular values (see HW2 for details on compact SVD)

Sanity Check: How does this relate to PCA?

SVD implications

- $\mathbf{\Phi} = \mathbf{U}\mathbf{\Sigma}^{1/2}\mathbf{V}^\top$
- $\mathbf{C} = \mathbf{\Phi}^\top\mathbf{\Phi} = \mathbf{V}\mathbf{\Sigma}^{1/2}\mathbf{U}^\top\mathbf{U}\mathbf{\Sigma}^{1/2}\mathbf{V}^\top = \mathbf{V}\mathbf{\Sigma}\mathbf{V}^\top$
- $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\top = \mathbf{U}\mathbf{\Sigma}^{1/2}\mathbf{V}^\top\mathbf{V}\mathbf{\Sigma}^{1/2}\mathbf{U}^\top = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^\top$

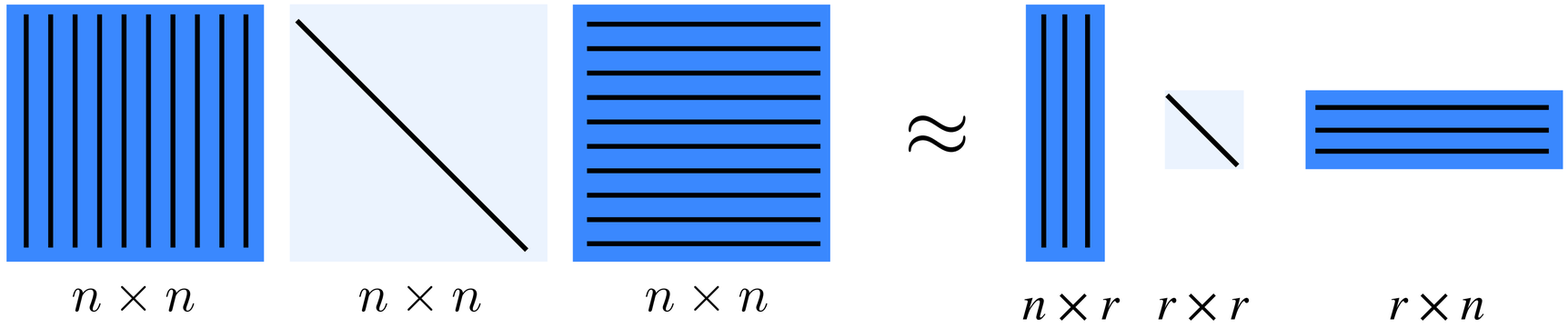
PCA implications

- Top r PCA scores (via C): $\mathbf{Z} = \mathbf{\Phi}\mathbf{V}_r \in \mathbb{R}^{n \times r}$
  - $\mathbf{V}_r$ are top r eigenvectors of C
- Top r PCA scores (via kernel matrix):

$$\mathbf{Z} = \mathbf{\Phi}\mathbf{V}_r = \mathbf{U}\mathbf{\Sigma}^{1/2}\mathbf{V}^\top\mathbf{V}_r = \mathbf{U}_r\mathbf{\Sigma}_r^{1/2}$$

Low-rank Approximation

- $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\top \in \mathbb{R}^{n \times n}$ is an SPSD matrix
- $\mathbf{K} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^\top$: Singular value decomposition (SVD)
- $\mathbf{K}_r = \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{U}_r^\top$: 'optimal' low-rank matrix ($r \ll n, M$)
  - $\mathbf{K}_r = \min_{rank(\tilde{\mathbf{K}})=r} \| \mathbf{K} - \tilde{\mathbf{K}} \|_F$



Low-rank Approximation

- $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\top \in \mathbb{R}^{n \times n}$ is an SPSD matrix
- $\mathbf{K} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^\top$: Singular value decomposition (SVD)
- $\mathbf{K}_r = \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{U}_r^\top$: 'optimal' low-rank matrix ($r \ll n, M$)

How does factorized $\mathbf{K}_r$ help speed up kernel methods?

- KRR: substituting it into $\alpha = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y}$ allows for linear time computation (instead of cubic) via "Woodbury Inversion Lemma"
- PCA: it directly gives us the PCA solution (see previous slide)

Low-rank Approximation

- $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\top \in \mathbb{R}^{n \times n}$ is an SPSD matrix
- $\mathbf{K} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^\top$: Singular value decomposition (SVD)
- $\mathbf{K}_r = \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{U}_r^\top$: 'optimal' low-rank matrix ($r \ll n, M$)

Goal: Fast (linear in n), Approximate SVD for large-scale kernel methods

- Minimize reconstruction error $\left\| \mathbf{K} - \tilde{\mathbf{K}} \right\|_F$
- Achieve $\mathscr{L}(\mathbf{K}) \approx \mathscr{L}(\tilde{\mathbf{K}})$ for learning algorithm $\mathscr{L}(\cdot)$

Low-rank Approximation

- $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\top \in \mathbb{R}^{n \times n}$ is an SPSD matrix
- $\mathbf{K} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^\top$: Singular value decomposition (SVD)
- $\mathbf{K}_r = \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{U}_r^\top$: 'optimal' low-rank matrix ($r \ll n, M$)

Can we just compute $\mathbf{K}_r$ and call it a day?

- No, SVD is expensive! — $O(n^2 r)$ for top r singular vectors/values
- Note that the distributed techniques from last lecture don't apply b/c we have access to $\mathbf{K}$ but not $\mathbf{\Phi}$ (and even storing $\mathbf{K}$ for very large n may be infeasible)

Outline

1. Background: Kernel Methods
2. Kernel Ridge Regression
3. SVD
4. Large-Scale Kernel Methods (via Approximate SVD)

Key Question: Can we approximate the SVD w/o looking at all the entries in the matrix?

What approximation should we use?

What entries should we look at?

Key Question: Can we approximate the SVD w/o looking at all the entries in the matrix?

What approximation should we use?

What entries should we look at?

Column-sampling Method

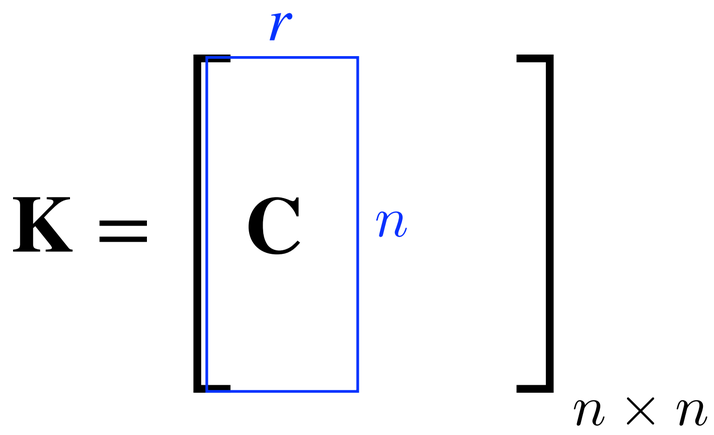
[Frieze et al., '98]

$$\mathbf{K} = \begin{bmatrix} & \\ & \end{bmatrix}_{n \times n}$$

Column-sampling Method

[Frieze et al., '98]

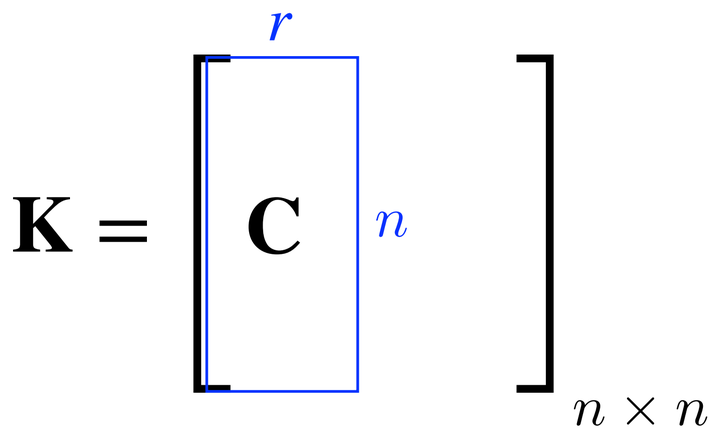
- Algorithm
  - Sample r columns of $\mathbf{K}$
  - SVD of $\mathbf{C}$: $O(nr^2)$



Column-sampling Method

[Frieze et al., '98]

- Algorithm
  - Sample r columns of $\mathbf{K}$
  - SVD of $\mathbf{C}$: $O(nr^2)$



- Approximation: $\tilde{\mathbf{K}}_{col} = \tilde{\mathbf{U}}_{col}\tilde{\mathbf{\Sigma}}_{col}\tilde{\mathbf{U}}_{col}^{\top}$
  - $\tilde{\mathbf{\Sigma}}_{col} = \sqrt{\frac{n}{r}}\mathbf{\Sigma}^{C}$
  - $\tilde{\mathbf{U}}_{col} = \mathbf{U}^{C}$

Intuition: SVD on $\mathbf{C}$ provides good approximation of full SVD

Nyström Method

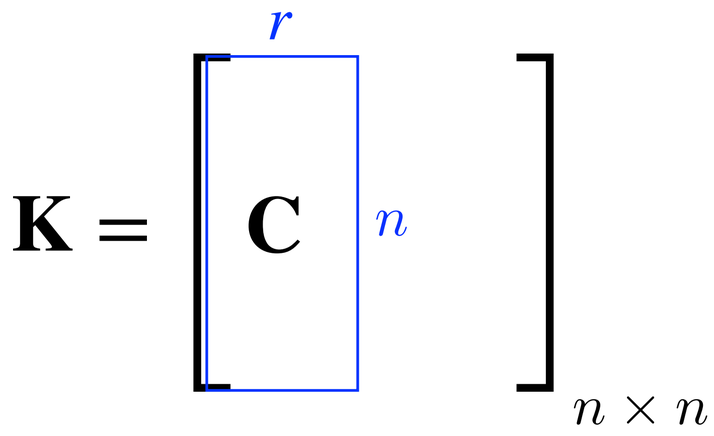
[Williams & Seeger, '00]
[Drineas & Mahoney, '05]

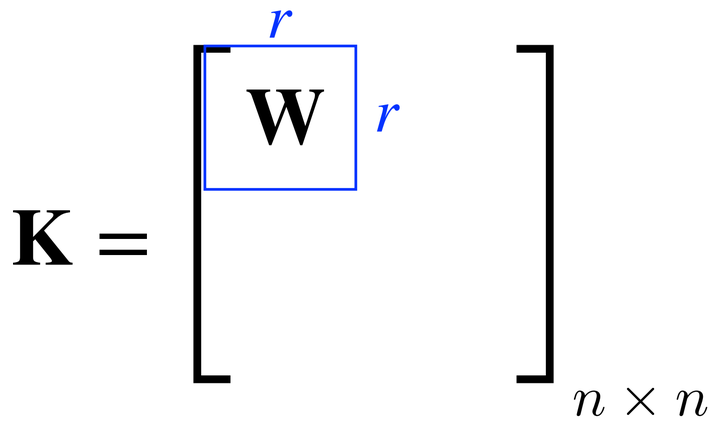
$$\mathbf{K} = \begin{bmatrix} & \\ & \end{bmatrix}_{n \times n}$$

Nyström Method

[Williams & Seeger, '00]
[Drineas & Mahoney, '05]

- Algorithm
  - Sample r columns of $\mathbf{K}$
  - Sample the same rows of $\mathbf{C}$
  - SVD / inverse of $\mathbf{W}$: $O(r^3)$

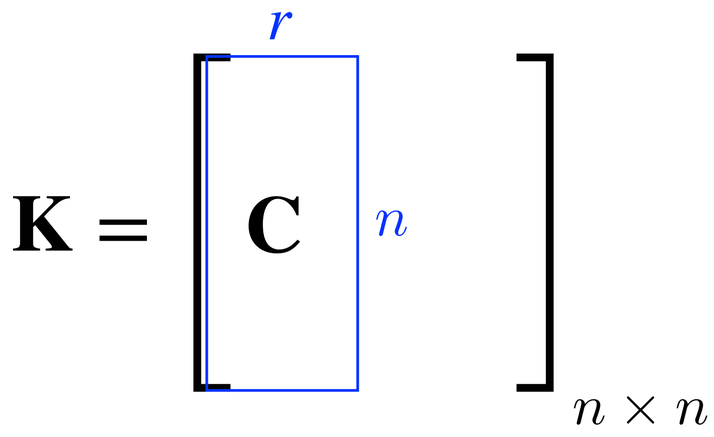




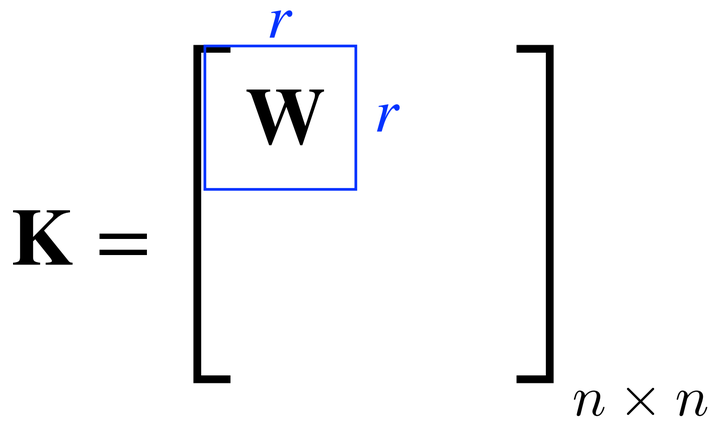
Nyström Method

[Williams & Seeger, '00]
[Drineas & Mahoney, '05]

- Algorithm
  - Sample r columns of $\mathbf{K}$
  - Sample the same rows of $\mathbf{C}$
  - SVD / inverse of $\mathbf{W}$: $O(r^3)$
  - Extrapolate: $O(nr^2)$



- Approximation: $\tilde{\mathbf{K}}_{nys} = \mathbf{C}\mathbf{W}^{-1}\mathbf{C}^\top$
  - What's going on here?



Nyström Method

[Williams & Seeger, '00]
[Drineas & Mahoney, '05]

$$\tilde{\mathbf{K}}_{nys} = \mathbf{C}\mathbf{W}^{-1}\mathbf{C}^{\top}$$

$$\mathbf{K} = \overbrace{\begin{bmatrix} \mathbf{C} \end{bmatrix}}^{r} \; n \qquad n \times n$$

$$\mathbf{K} = \begin{matrix} \\ r \\ \\ \end{matrix} \overbrace{\begin{bmatrix} \mathbf{W} & \mathbf{K}_{21}^{\top} \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix}}^{r} \qquad n \times n$$

Nyström Method

[Williams & Seeger, '00]
[Drineas & Mahoney, '05]

$$\tilde{\mathbf{K}}_{nys} = \mathbf{C}\mathbf{W}^{-1}\mathbf{C}^\top$$

$$= \overset{n \times r}{\begin{bmatrix} \mathbf{W} \\ \mathbf{K}_{21} \end{bmatrix}} \overset{r \times r}{\mathbf{W}^{-1}} \overset{r \times n}{\begin{bmatrix} \mathbf{W} & \mathbf{K}_{21}^\top \end{bmatrix}}$$

$$\mathbf{K} = \begin{bmatrix} \overset{r}{\boxed{\mathbf{C}}} \; n \\ \end{bmatrix}_{n \times n}$$

$$\mathbf{K} = \begin{bmatrix} \overset{r}{\boxed{\mathbf{W}}} & \mathbf{K}_{21}^\top \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix}_{n \times n} \quad r$$

Nyström Method

[Williams & Seeger, '00]
[Drineas & Mahoney, '05]

$$\tilde{\mathbf{K}}_{nys} = \mathbf{C}\mathbf{W}^{-1}\mathbf{C}^\top$$

$$= \overset{n\times r}{\begin{bmatrix} \mathbf{W} \\ \mathbf{K}_{21} \end{bmatrix}} \overset{r\times r}{\mathbf{W}^{-1}} \overset{r\times n}{\begin{bmatrix} \mathbf{W} & \mathbf{K}_{21}^\top \end{bmatrix}}$$

$$= \begin{bmatrix} \mathbf{W} & \mathbf{K}_{21}^\top \\ \mathbf{K}_{21} & \mathbf{K}_{21}\mathbf{W}^{-1}\mathbf{K}_{21}^\top \end{bmatrix}_{n\times n}$$

$$\mathbf{K} = \begin{bmatrix} \overset{r}{\boxed{\mathbf{C}}} \; n \end{bmatrix}_{n\times n}$$

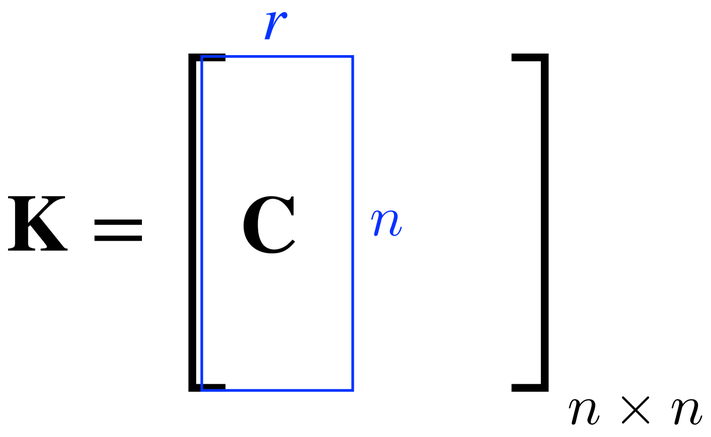
$$\mathbf{K} = \begin{bmatrix} \overset{r}{\boxed{\mathbf{W}}} & \mathbf{K}_{21}^\top \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix}_{n\times n} \quad r$$

note: $\mathbf{K}_{22} - \mathbf{K}_{21}\mathbf{W}^{-1}\mathbf{K}_{21}^\top$ is a well studied quantity called the "Schur complement"

How Do They Compare?

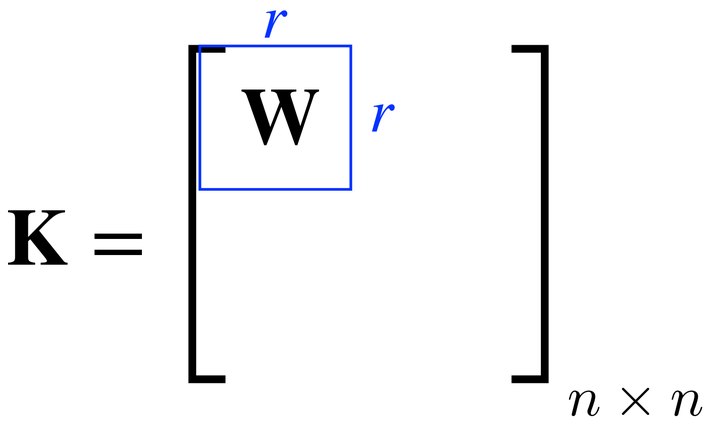
Column Sampling

- Estimate SVD via subset of columns
- $\tilde{\mathbf{K}}_{col} = \mathbf{C}\left((r/n)\mathbf{C}^\top\mathbf{C}\right)^{-1/2}\mathbf{C}^\top$



Nystrom Method

- Estimate SVD via smaller sub-matrix
- $\tilde{\mathbf{K}}_{nys} = \mathbf{C}\mathbf{W}^{-1}\mathbf{C}^\top$



How Do They Compare?

Assume $rank(\mathbf{W}) = rank(\mathbf{K})$

How Do They Compare?

Assume $rank(\mathbf{W}) = rank(\mathbf{K})$

- *Nyström is exact*
- *Column-sampling exact iff it trivially reduces to Nyström, i.e.,* $\mathbf{W} = \left((r/n)\mathbf{C}^\top\mathbf{C}\right)^{1/2}$

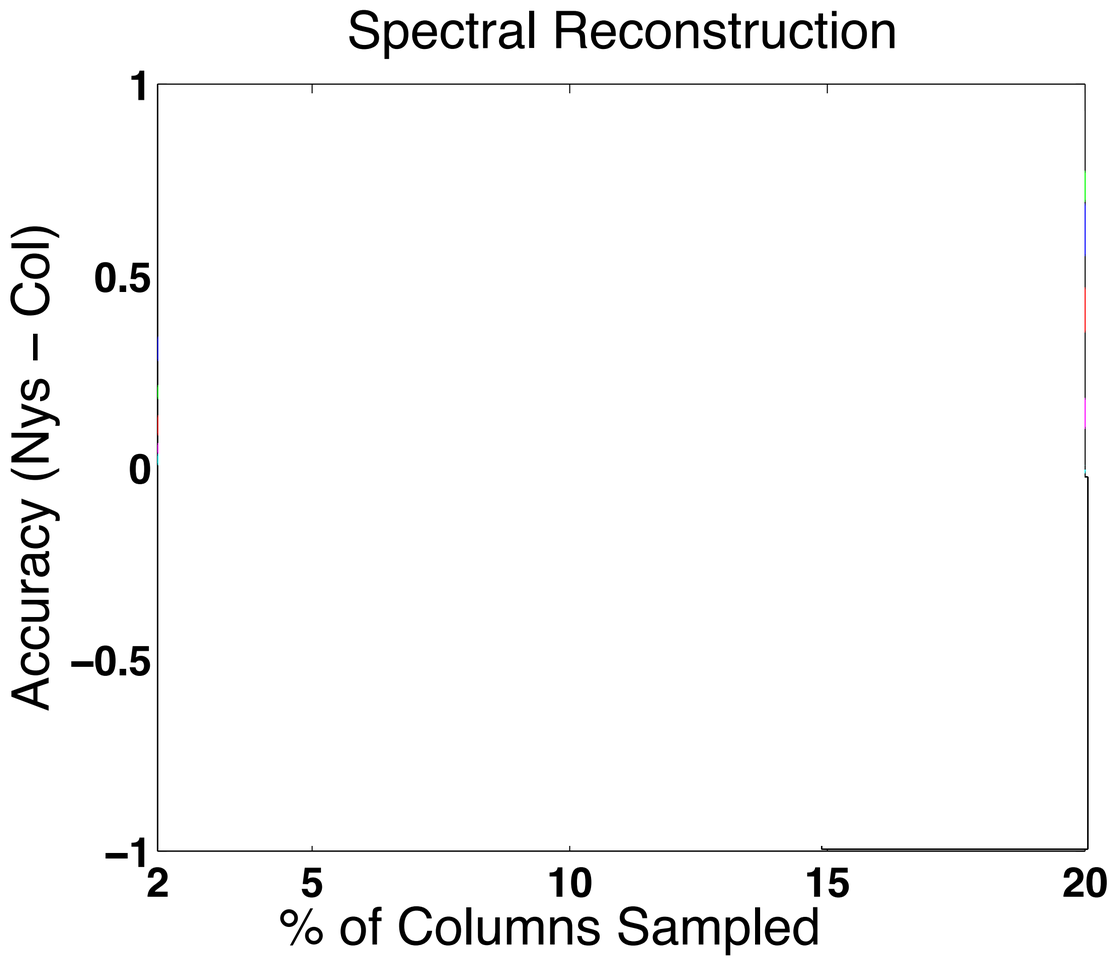
How Do They Compare?

Assume $rank(\mathbf{W}) = rank(\mathbf{K})$

- *Nyström is exact*
- *Column-sampling exact iff it trivially reduces to Nyström, i.e.,* $\mathbf{W} = \left((r/n)\mathbf{C}^\top\mathbf{C}\right)^{1/2}$

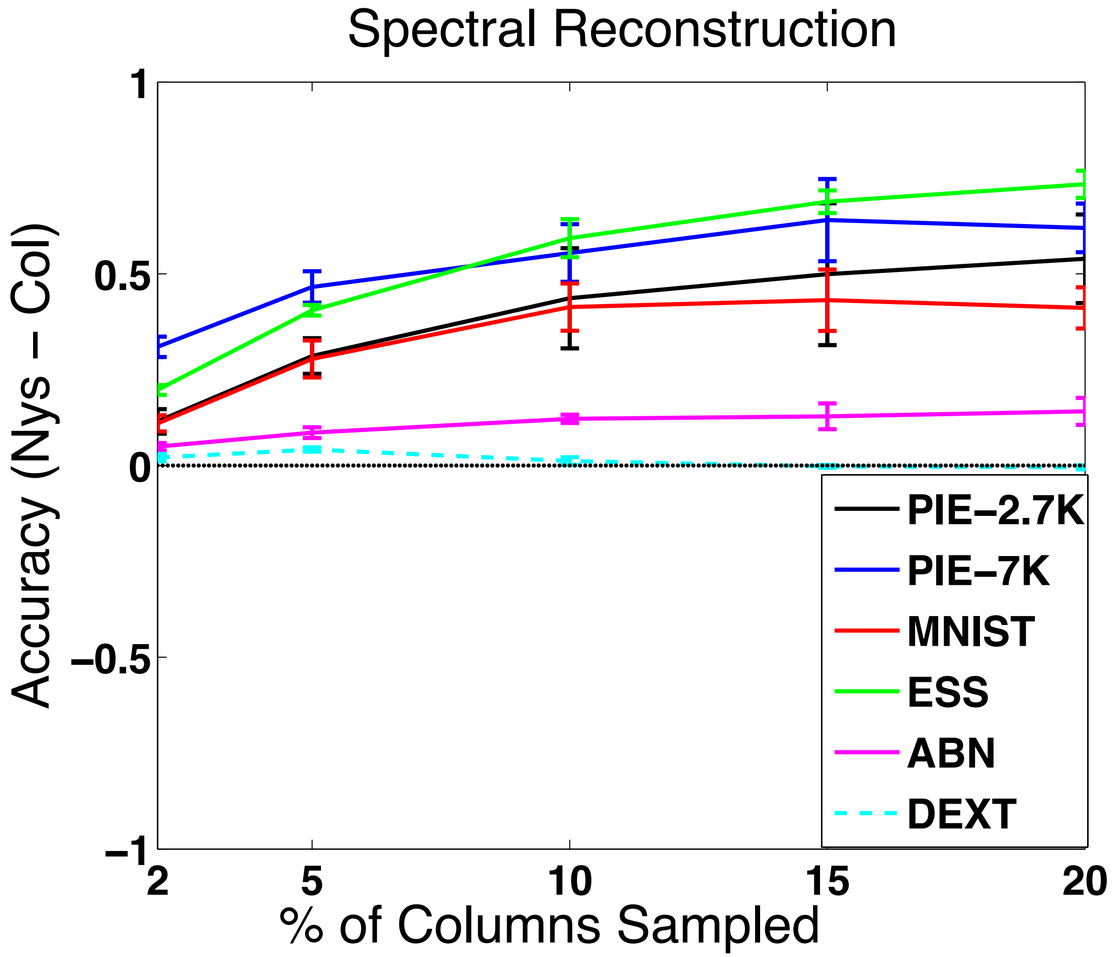
- Under certain conditions, Nyström is optimal, Column-sampling is suboptimal
- Suggests that if $\mathbf{K}$ has low-rank structure ($\mathbf{K} \approx \mathbf{K}_r$), then Nyström will be better

How Do They Compare?



- Accuracy defined as $\frac{\| \mathbf{K} - \mathbf{K}_r \|_F}{\| \mathbf{K} - \tilde{\mathbf{K}} \|_F}$
- Above 0 signifies Nyström is better

How Do They Compare?



- Accuracy defined as $\frac{\| \mathbf{K} - \mathbf{K}_r \|_F}{\| \mathbf{K} - \tilde{\mathbf{K}} \|_F}$
- Above 0 signifies Nyström is better
- Nyström better on all 5 dense matrices with quickly decaying eigenvalues

- Suggests that if $\mathbf{K}$ has low-rank structure ($\mathbf{K} \approx \mathbf{K}_r$), then Nyström will be better

Key Question: Can we approximate the SVD w/o looking at all the entries in the matrix?

What approximation should we use?

What entries should we look at?

Limitations of Random Sampling

$$\mathbf{K} = \begin{bmatrix} 1 & & & & & & \\ & 1 & & & & \mathbf{0} & \\ & & 1 & & & & \\ & & & 0 & & & \\ & & & & 0 & & \\ & \mathbf{0} & & & & 0 & \\ & & & & & & 0 \end{bmatrix}_{n \times n}$$

Optimal Choice

Leverage scores

- function of exact eigenvectors
- Very expensive: requires computing SVD!

Interesting theoretically, but not for practical, large-scale settings

How to sample in practice?

- Distribution over the columns remains fixed
  - uniform: $O(1)$
  - diagonal: $O(n)$
  - column-norm: $O(n^2)$
- Adaptively pick columns or perform pre-processing
  - eg. via *K*-means [Zhang et al., '08]
- Combine approximations (ensemble algorithm)

Take-aways

- Kernel methods are a popular approach for encoding complex features
- But, kernel methods don't scale well with *n*
- A natural approach is low-rank approximations of the kernel matrix, K
- SVD is a powerful tool
- But, calculating the optimal low-rank approximation (eg., via SVD) is expensive:
  - Column sampling or Nystrom method can make this feasible

Notes

- Random Fourier Features: another popular alternative for fast, approximate kernel methods, focusing on approximating $\phi(\cdot)$
- Theory connecting reconstruction error with generalization error
  - On the Impact of Kernel Approximation on Learning Accuracy. Cortes, Mohri, Talwalkar. AISTATS 2010.
  - Sharp analysis of low-rank kernel matrix approximations. Bach. COLT, 2013.
- Other References
  - Randomized algorithms for matrices and data. Mahoney. Foundations & Trends in Machine Learning, 2011.
  - Random Features for Large-Scale Kernel Machines. Rahimi and Recht. NeurIPS 2007.
  - Matrix Approximation for Large-scale Learning. Talwalkar. PhD Thesis. 2010.