Machine Learning for Data Science (CS4786) Lecture 6

Non-Linear Dimensionality Reduction

Course Webpage :
http://www.cs.cornell.edu/Courses/cs4786/2016fa/

Announcement

- Assignment 0 feedback available on cms.
- Assignment 1 helper code in matlab, ipython and R added, due on friday.

Recap

Pick a Random W

$$Y = X \times \left. \begin{array}{cc} \begin{bmatrix} +1 & \dots & -1 \\ -1 & \dots & +1 \\ +1 & \dots & -1 \\ & \vdots & \\ +1 & \dots & -1 \end{bmatrix} & d \\ K & \end{array} \right/ \sqrt{K}$$

Random Projection

Distances between all pairs of data-points in low dim. projection is roughly the same as their distances in the high dim. space.

Why should Random Projections even work?!

Say $K = 1$. Consider any vector $\tilde{\mathbf{x}} \in \mathbb{R}^d$ and let $\tilde{\mathbf{y}} = \tilde{\mathbf{x}}^\top W$.

We showed that: $\mathbb{E}\left[|\tilde{\mathbf{y}}|^2\right] = \|\tilde{\mathbf{x}}\|_2^2$

$K > 1$, $\tilde{\mathbf{y}}[j] = \tilde{\mathbf{x}}^\top W_j$ $\quad$ $\tilde{\mathbf{y}}[i]$ & $\tilde{\mathbf{y}}[j]$ are independent

(since we divide each entry of random matrix by $\sqrt{K}$ in W)

$$\mathbb{E}\left[|\tilde{\mathbf{y}}[j]|^2\right] = \frac{1}{K}\|\tilde{\mathbf{x}}\|^2$$

$$\text{Hence, } \mathbf{E}\|\tilde{\mathbf{y}}\|^2 = \sum_{j=1}^{K} \mathbf{E}\left[\mathbf{y}[j]^2\right] = \sum_{j=1}^{K} \frac{1}{K}\|\tilde{\mathbf{x}}\|^2 = \|\tilde{\mathbf{x}}\|^2$$

This is like taking an average of K independent measurements whose expectations are $\|\tilde{\mathbf{x}}\|_2^2$

WHY SHOULD RANDOM PROJECTIONS EVEN WORK?!

For large K, not only true in expectation but also with high probability

For any $\epsilon > 0$, if $K \approx \log(n/\delta)/\epsilon^2$, with probability $1 - \delta$ over draw of W, for all pairs of data points $i, j \in \{1, \ldots, n\}$,

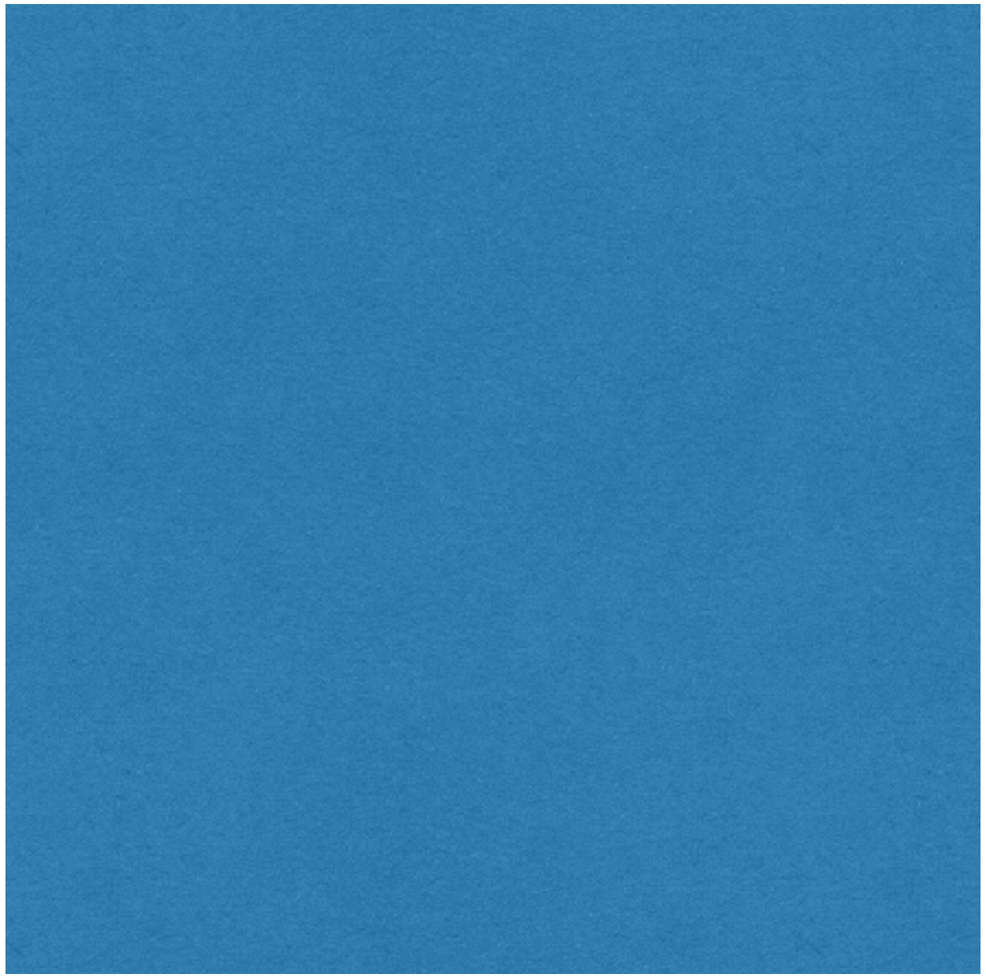
$$(1-\epsilon)\left\|\mathbf{y}_i - \mathbf{y}_j\right\|_2^{\mathbf{2}} \leq \left\|\mathbf{x}_i - \mathbf{x}_j\right\|_2 \leq (1+\epsilon)\left\|\mathbf{y}_i - \mathbf{y}_j\right\|_2^{\mathbf{2}}$$

Lets try on Matlab ...

This is called the Johnson-Lindenstrauss lemma or JL lemma for short.

Why is this so Ridiculously Magical?

n= 1000

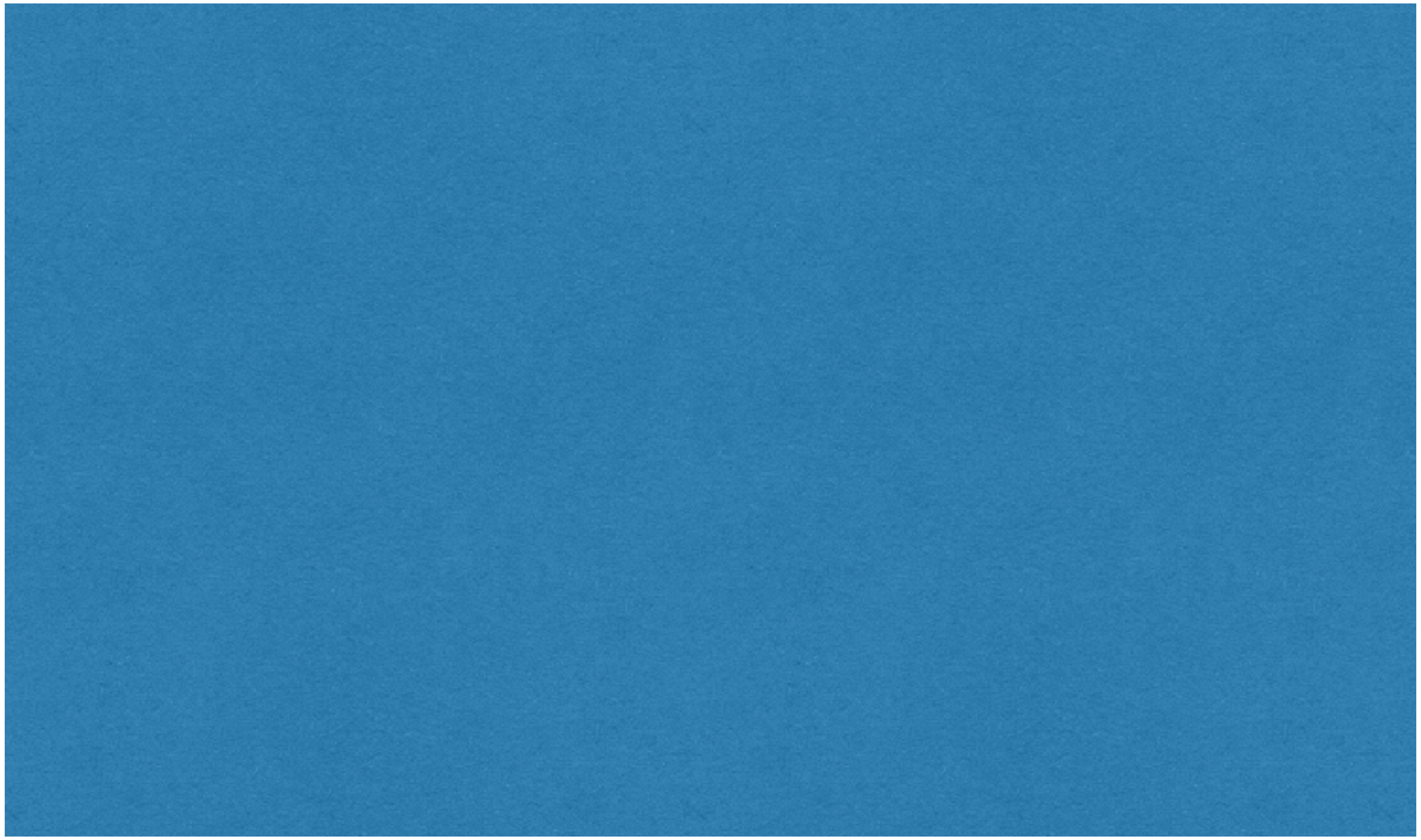


d = 1000

If we take $\epsilon = 1/4$, then taking $K \approx 185$ with probability 0.99 distances are preserved to factor $1/4$

WHY IS THIS SO RIDICULOUSLY MAGICAL?

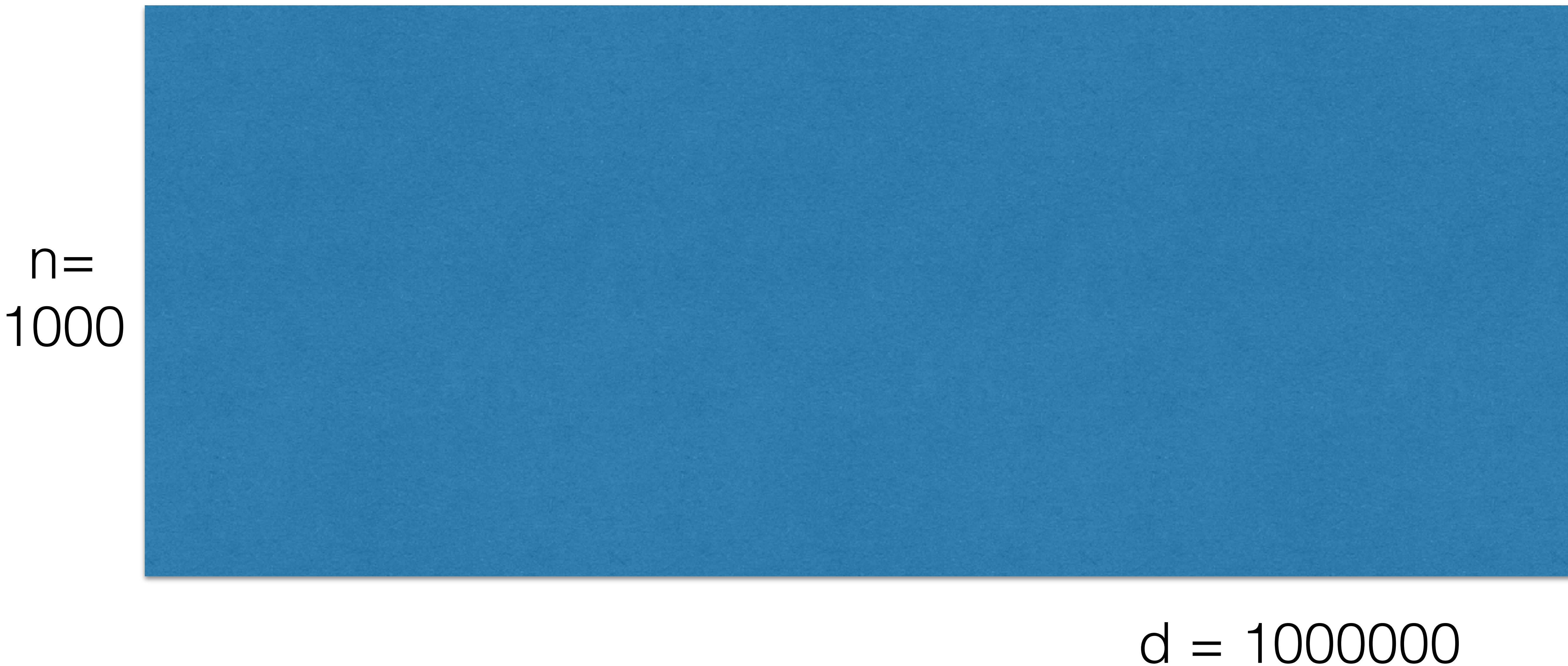
n= 1000



d = 10000

If we take $\epsilon = 1/4$, then taking $K \approx 185$ with probability 0.99 distances are preserved to factor $1/4$

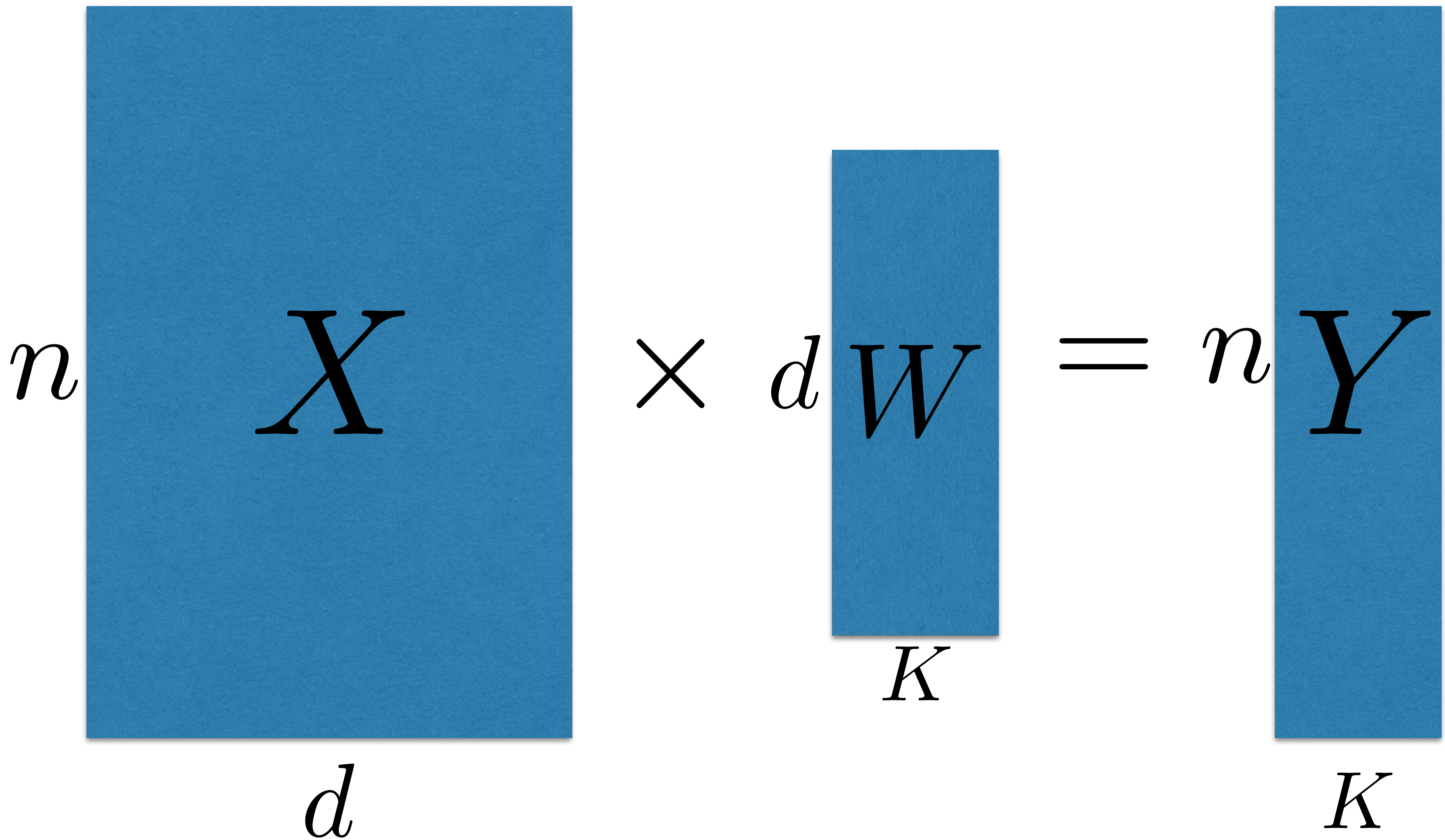
WHY IS THIS SO RIDICULOUSLY MAGICAL?



If we take $\epsilon = 1/4$, then taking $K \approx 185$ with probability 0.99 distances are preserved to factor $1/4$

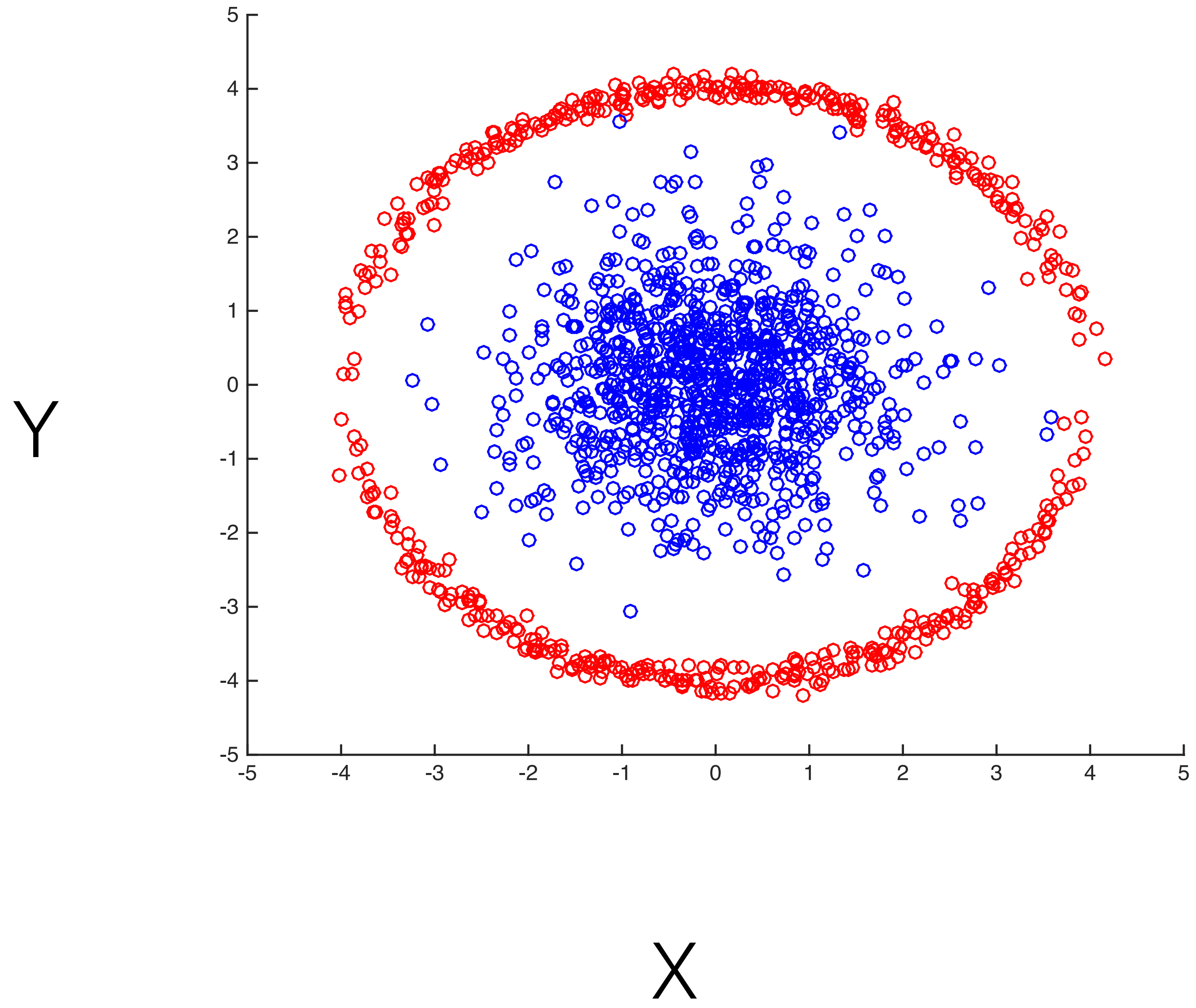
Kernel PCA
(non-linear projections)

Linear Projections



Works when data lies in a low dimensional linear sub-space

Example



Linear Projections (Right Co-ordinates)

Demo

A First Cut

- Given $\mathbf{x}_t \in \mathbb{R}^d$, the feature space vector is given by mapping

$$\Phi(\mathbf{x}_t) = (\mathbf{x}_t[1], \ldots, \mathbf{x}_t[d], \mathbf{x}_t[1] \cdot \mathbf{x}_t[1], \mathbf{x}_t[1] \cdot \mathbf{x}_t[2], \ldots, \mathbf{x}_t[d] \cdot \mathbf{x}_t[d], \ldots)^\top$$

- Enumerating products up to order K (ie. products of at most K coordinates) we can get degree K polynomials.
- However dimension blows up as d^K
- Is there a way to do this without enumerating Φ?

Kernel Trick

- Essence of Kernel trick:
  - **If** we can write down an algorithm only in terms of $\Phi(\mathbf{x}_t)^\top \Phi(\mathbf{x}_s)$ for data points $\mathbf{x}_t$ and $\mathbf{x}_s$
  - **Then** we don't need to explicitly enumerate $\Phi(\mathbf{x}_t)$'s but instead, compute $k(\mathbf{x}_t, \mathbf{x}_s) = \Phi(\mathbf{x}_t)^\top \Phi(\mathbf{x}_s)$ (even if Φ maps to infinite dimensional space)
- Example: RBF kernel $k(\mathbf{x}_t, \mathbf{x}_s) = \exp(-\sigma \|\mathbf{x}_t - \mathbf{x}_s\|_2^2)$, polynomial kernel $k(\mathbf{x}_t, \mathbf{x}_s) = \left(\mathbf{x}_t^\top \mathbf{y}_t\right)^p$
- Kernel function measures similarity between points.

Lets Rewrite PCA

- k^{th} column of W is eigenvector of covariance matrix
  That is, $\lambda_k W_k = \Sigma W_k$. Rewriting, for centered X

$$\lambda_k W_k = \frac{1}{n}\left(\sum_{t=1}^{n} \mathbf{x}_t \mathbf{x}_t^\top\right) W_k = \frac{1}{n}\sum_{t=1}^{n}\left(\mathbf{x}_t^\top W_k\right)\mathbf{x}_t$$

W_k's can be written as linear combination of $\mathbf{x}_t$'s, as

$$W_k = \sum_{t=1}^{n} \alpha_k[t]\mathbf{x}_t$$

where $\alpha_k[t] = \frac{1}{\lambda_k n}\left(\mathbf{x}_t^\top W_k\right)$

LETS REWRITE PCA

- We have that $W_k = \sum_{s=1}^{n} \alpha_k[s]\mathbf{x}_s$ and that $\alpha_k[t] = \frac{1}{\lambda_k n}\left(\mathbf{x}_t^\top W_k\right)$.
- Hence:

$$\alpha_k[t] = \frac{1}{\lambda_k n}\left(\mathbf{x}_t^\top\left(\sum_{s=1}^{n} \alpha_k[s]\mathbf{x}_s\right)\right) = \frac{1}{\lambda_k n}\sum_{s=1}^{n} \alpha_k[s]\mathbf{x}_t^\top \mathbf{x}_s$$

- Let $\tilde{K}$ be a matrix such that $\tilde{K}_{s,t} = \mathbf{x}_t^\top \mathbf{x}_s$. Hence, $\alpha_k[t] = \frac{1}{\lambda_k n}\alpha_k^\top \tilde{K}_t$ and

$$\alpha_k = \frac{1}{\lambda_k n}\tilde{K}\alpha_k$$

  where $\tilde{K}_t$ is the t'th column of $\tilde{K}$.
- Hence α_k is in the direction of eigen vector of $\tilde{K}$

Lets Rewrite PCA

- Further, since W_k is unit norm,

$$1 = \|W_k\|_2^2 = \left(\sum_{t=1}^{n} \boldsymbol{\alpha}_k[t]\mathbf{x}_t\right)^\top \left(\sum_{s=1}^{n} \boldsymbol{\alpha}_k[s]\mathbf{x}_s\right) = \boldsymbol{\alpha}_k{}^\top \tilde{K} \boldsymbol{\alpha}_k = n\gamma_k \boldsymbol{\alpha}_k{}^\top \boldsymbol{\alpha}_k$$

Hence $\|\boldsymbol{\alpha}_k\|^2 = \frac{1}{n\gamma_k}$ where γ_k is the k'th eigen value of matrix $\tilde{K}$

LETS REWRITE PCA

- However W_k itself is in feature space and has the same dimensionality of $\Phi(x)$ (which is possibly infinite)!
- However, the projections are in K dimensions and we can hope to directly compute these as:

$$\mathbf{y}_i[k] = \mathbf{x}_i^\top W_k = \sum_{t=1}^{n} \boldsymbol{\alpha}_k[t]\tilde{K}_{t,i}$$

Rewritting PCA

- We assumed centered data, what if its not,

$$
\begin{aligned}
\tilde{K}_{s,t} &= \left(\mathbf{x}_t - \frac{1}{n}\sum_{u=1}^{n}\mathbf{x}_u)\right)^\top \left(\mathbf{x}_s - \frac{1}{n}\sum_{u=1}^{n}\mathbf{x}_u\right) \\
&= \mathbf{x}_t^\top \mathbf{x}_s - \left(\frac{1}{n}\sum_{u=1}^{n}\mathbf{x}_u\right)^\top \mathbf{x}_s - \left(\frac{1}{n}\sum_{u=1}^{n}\mathbf{x}_u\right)^\top \mathbf{x}_t \\
&\quad + \frac{1}{n^2}\left(\sum_{u=1}^{n}\mathbf{x}_u\right)^\top \left(\sum_{v=1}^{n}\mathbf{x}_v\right) \\
&= \mathbf{x}_t^\top \mathbf{x}_s - \frac{1}{n}\sum_{u=1}^{n}\mathbf{x}_u^\top \mathbf{x}_s - \frac{1}{n}\sum_{u=1}^{n}\mathbf{x}_u^\top \mathbf{x}_t + \frac{1}{n^2}\sum_{u=1}^{n}\sum_{v=1}^{n}\mathbf{x}_u^\top \mathbf{x}_v
\end{aligned}
$$

Rewriting PCA

- Equivalently, if Kern is the matrix ($\mathrm{Kern}_{t,s} = x_t^\top x_s$),

$$\tilde{K} = \mathrm{Kern} - \frac{(\mathbf{1}_{n\times n} \times \mathrm{Kern})}{n} - \frac{(\mathrm{Kern} \times \mathbf{1}_{n\times n})}{n} + \frac{(\mathbf{1}_{n\times n} \times \mathrm{Kern} \times \mathbf{1}_{n\times n})}{n^2}$$

PCA REWRITTEN

- Compute $\tilde{K} = \mathrm{Kern} - \mathbf{1}\ \mathrm{Kern}/n - \mathrm{Kern}\ \mathbf{1}/n + \mathbf{1}\ \mathrm{Kern}\ \mathbf{1}/n^2$

- Compute top K eigen vectors $P_1, \ldots, P_K$ along with eigen values $\gamma_1, \ldots, \gamma_K$ for the matrix $\tilde{K}$

- Rescale each P_k by the inverse of the square-root of corresponding eigen values ie. $\alpha_k = P_k/\sqrt{n\gamma_k}$

- Compute projections by setting

$$\mathbf{y}_i[k] = \sum_{t=1}^{n} \boldsymbol{\alpha}_k[t]\tilde{K}_{t,i}$$

or in other words $Y = \tilde{K} \times [\alpha_1, \ldots, \alpha_K]$

Kernel PCA

All we need to be able to compute, to perform PCA are $\mathbf{x}_t^\top \mathbf{x}_s$

Replace $\mathbf{x}_t^\top \mathbf{x}_s$ with $\Phi(\mathbf{x}_t)^\top \Phi(\mathbf{x}_s) = k(x_t, x_s)$ to perform PCA in feature space

Kernel PCA

1.

$$\underbrace{\mathrm{Kern}}_{n \times n} = \begin{bmatrix} k(x_1,x_1) & k(x_1,x_2) & \dots & k(x_1,x_n) \\ k(x_2,x_1) & k(x_2,x_2) & \dots & k(x_2,x_n) \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ k(x_{n-1},x_1) & k(x_{n-1},x_2) & \dots & k(x_{n-1},x_n) \\ k(x_n,x_1) & k(x_n,x_2) & \dots & k(x_n,x_n) \end{bmatrix}$$

2.

$$\underbrace{\tilde{K}}_{n \times n} = \mathrm{Kern} - \frac{1}{n}\left(\mathbf{1}\ \mathrm{Kern} + \mathrm{Kern}\ \mathbf{1}\right) + \frac{1}{n^2}\mathbf{1}\ \mathrm{Kern}\ \mathbf{1}$$

Kernel PCA

3. $$\left[{}^{n}\underset{K}{P}, \gamma \right] = \text{eigs}\left(\tilde{K}, K \right)$$

4. $${}^{n}\underset{K}{\alpha} = {}^{n}\underset{K}{\left[\frac{P_1}{\sqrt{n\gamma_1}} \dots \frac{P_K}{\sqrt{n\gamma_K}} \right]}$$

5. $${}^{n}\underset{K}{Y} = {}^{n}\underset{n}{\tilde{K}} \times {}^{n}\underset{K}{\alpha}$$

Demo