Efficient non-linear analysis of large data sets

Daniel Perry, Shireen Elhabian, Ross Whitaker

The Plan

- **Introduce Kernel PCA**
- **Difficulty at scale – Nystrom method**
- **Lasso Dictionary Selection**
  - Note: Submitted to ICML (anonymous review)
  - Feedback welcome

Kernel PCA

- **Review PCA**
  - Linear dimension reduction
- **Kernel PCA**
  - Non-linear dimension reduction
  - Kernel trick
- **Relationship to other kernel methods**
  - SVM
  - Etc.

Principal Component Analysis (PCA)

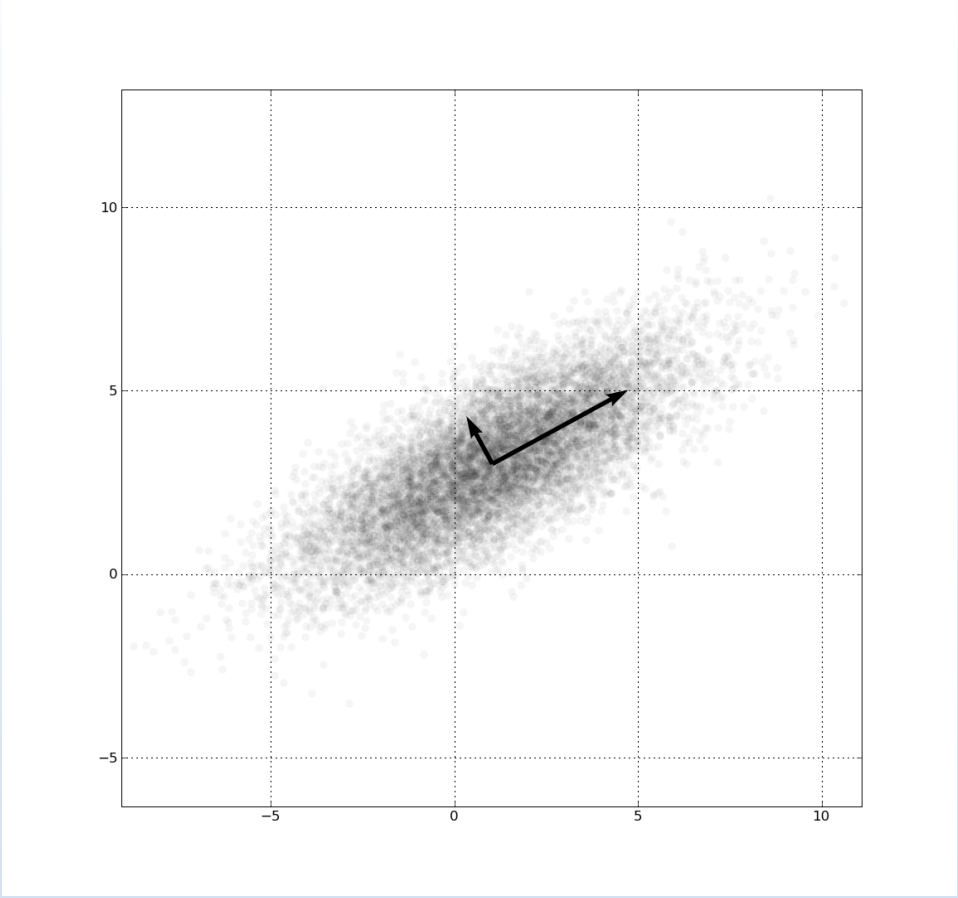
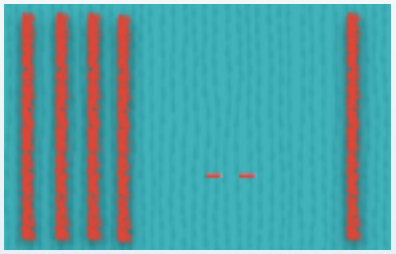


Image: wikipedia

- **Find a new basis where the first direction preserves the most variance,**
- **Second basis, second most, etc.**

PCA: review

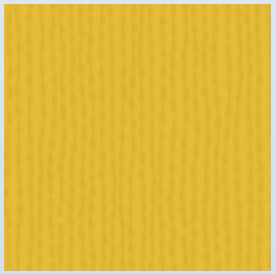
Centered data matrix



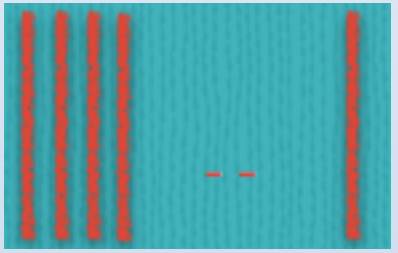
$$X \in R^{r \times n}, X = [x_1, \ldots, x_i] \in R^r$$

Compute covariance

$$C = XX^T \in R^{r \times r}$$
$$C = E[x_i x_i^T]$$



C $=$



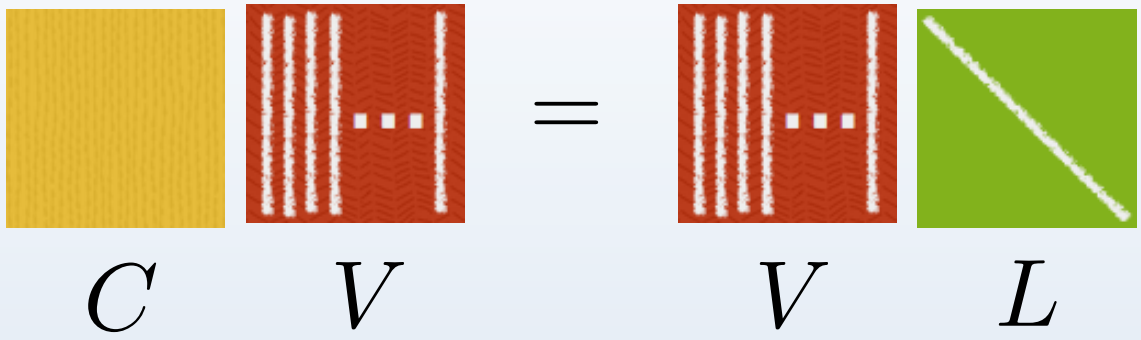
X

X^T

PCA: review

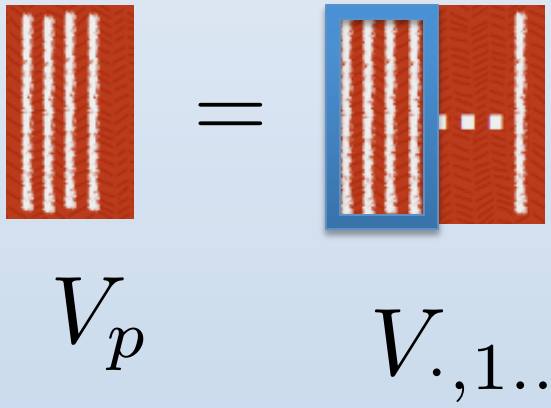
Eigen Decomposition

$$
\begin{aligned}
CV &= VL \\
V^T V &= VV^T = I \\
L &\quad \text{diagonal}
\end{aligned}
$$



Select p dominant eigen vectors

$$V_p = V_{\cdot,1\ldots p}$$

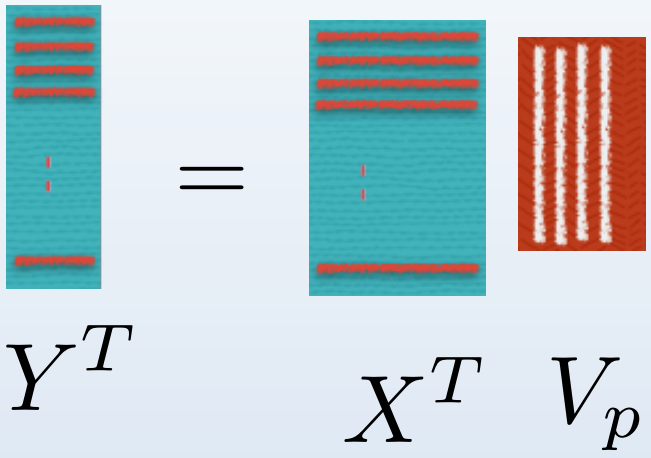


PCA: review

Project data into p-subspace

$$Y^T = X^T V_p$$

$$Y \in R^{p \times n}$$



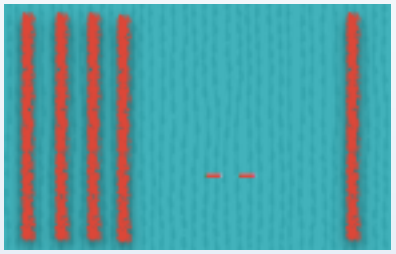
- **Optimal p-dimensional subspace in the sense of preserving the variance of data in r-dimensional space**

PCA: Alternative Computation

- **Alternative computation (important later)**
- **Normally use covariance**
  - Eigen decomposition is $O(r^3)$
    - (note: theoretically can do it in $O(r^{2.3})$)
- **Gramian matrix can also be used**
  - Eigen decomposition is $O(n^3)$
  - Useful when r > n (high dimensional data)
- **Aside: PCA can also be done using SVD of X**
  - SVD is $O(nr^2)$ or $O(rn^2)$

PCA: Gramian computation

Centered data matrix

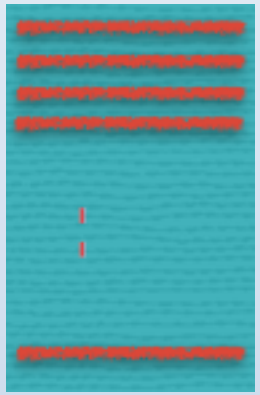


$$X \in R^{r \times n}, X = [x_1, \ldots, x_i] \in R^r$$

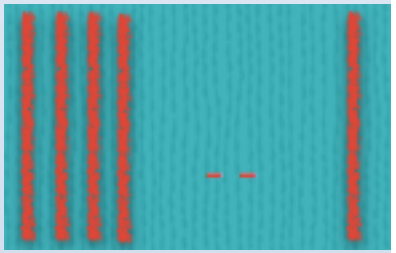
Compute Gramian

$$\begin{aligned} G &= X^T X \in R^{n \times n} \\ G_{i,j} &= x_i^T x_j \end{aligned}$$

G $=$



X^T

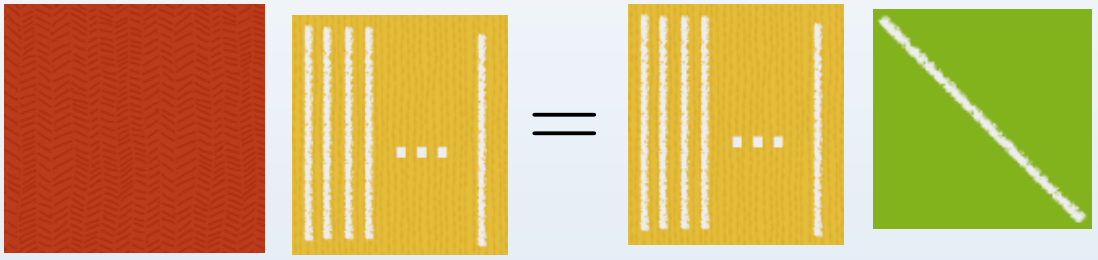


X

PCA: Gramian Computation

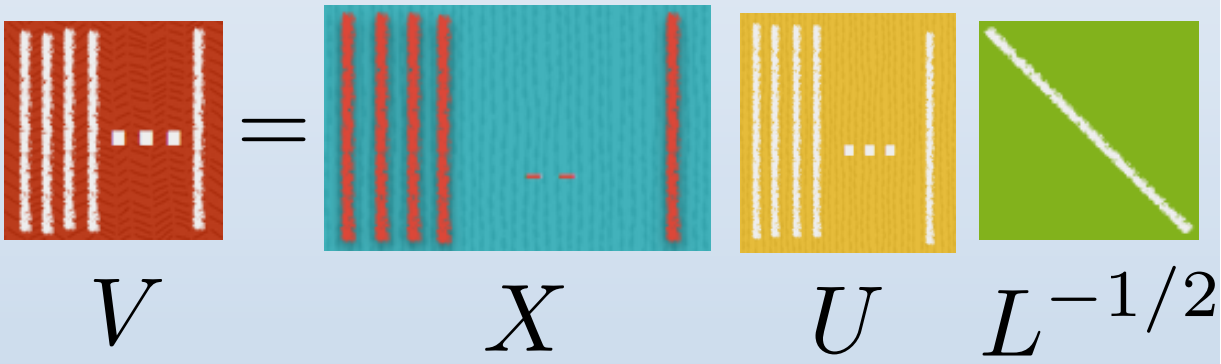
Eigen decomposition

$$
\begin{aligned}
GU &= UL \\
U^T U &= UU^T = I \\
L &\quad \text{diagonal}
\end{aligned}
$$



Recover PC Vectors

$$
\begin{aligned}
GU &= UL \\
X^T XU &= UL \\
XX^T(XU) &= (XU)L \\
C(XU) &= (XU)L
\end{aligned}
$$



PCA: Gramian computation

Important things to notice

- **Only require Gramian (inner products)**
- **Decomposition of n x n matrix**
  - Not dependent on dimension d
- **This is important if d > n**

PCA: Why?

- **Understanding data**
  - Eg, look at how data changes along first principal component
- **Dimension reduction**
  - Visualization
  - Pre-process data (ie for clustering, etc)

PCA: assumptions

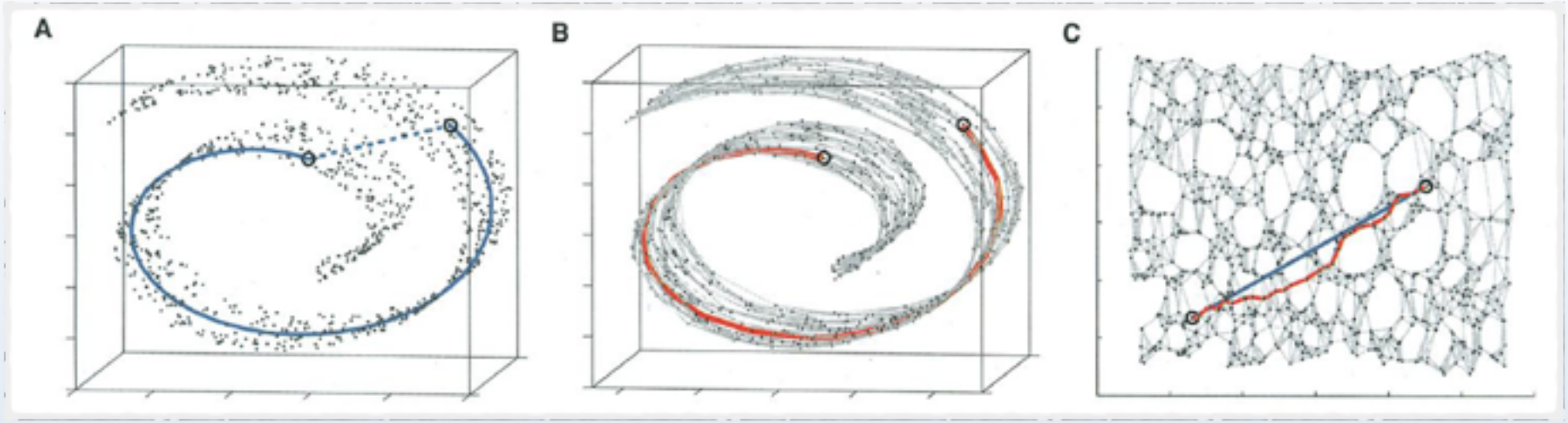
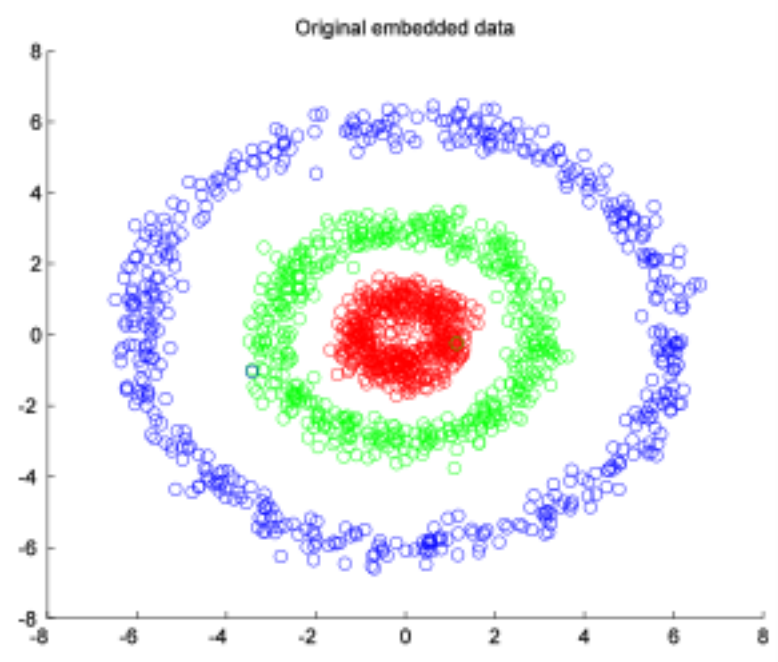
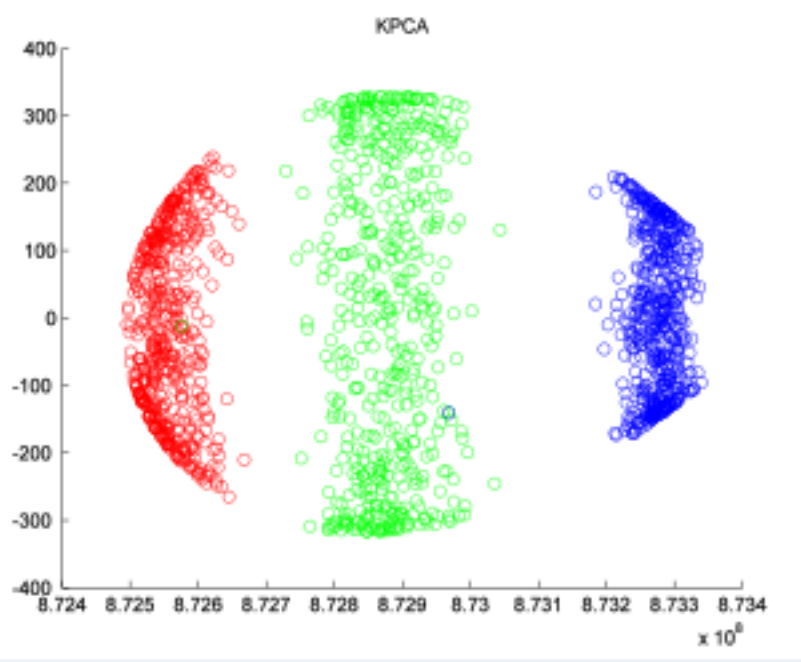


Image: Tannenbaum, et al. “A global geometric framework for nonlinear dimension reduction”, Science 2000

- **PCA assumes the data can be described “accurately” by a linear space**
- **Is the data better described by a non-linear subspace?**

PCA: assumptions





- **PCA assumes the data can be described "accurately" by a linear space**
- **Is the data better described by a non-linear subspace?**

PCA: non-linear alternatives

- **Manifold Learning**
  - Isomap
  - Locally Linear Embedding (LLD)
  - Etc
- **Principal Geodesic Analysis (PGA)**
- **Kernel PCA**
- **Etc**

PCA: non-linear alternatives

- **Manifold Learning**
  - Isomap
  - Locally Linear Embedding (LLD)
  - Etc
- **Principal Geodesic Analysis (PGA)**
- ***Kernel PCA***
- **Etc**

Kernel PCA: general formulation

- **General formulation of Kernel PCA is attractive for non-linear analysis**
  - [Bengio, et al. 2004] showed that Isomap, LLE, and other manifold learning can be formulated as Kernel PCA
  - [Snape and Zafeiriou 2014] showed that PGA can be formulated as Kernel PCA
  - More generally, can perform Kernel PCA using any **Mercer kernel** (discussed later)
- **Note: more efficient to compute Isomap, PGA, etc directly, than in kPCA form**

Kernel PCA: Introduction

- **Instead of PCA on the elements of X, PCA on a (non-linear) transformation, $\phi(X_i)$**
- **$\phi(X_i)$ can be potentially infinite in length**
- **non-linear analysis we are doing depends on choice of $\phi(X_i)$**
  - Isomap
  - LLE
  - PGA
  - Polynomial, etc.

Kernel PCA: Introduction

Kernel PCA as intuitive progression from PCA:

- PCA on (non-linear) features
  - Non-linear PCA
- PCA on BIG (non-linear) features
- Kernel trick
- Kernel PCA

PCA on (non-linear) features

- Instead of PCA on X
- Run PCA on a (non-linear) transformation of X
- Example: quadratic expansion

$$\begin{aligned}\phi(X_i) &= [\phi_1(X_i), \phi_2(X_i)] \\ &= [X_{i,1}, X_{i,1}^2, \ldots X_{i,r}, X_{i,r}^2]\end{aligned}$$

- Feature matrix, each column: $\Phi_i = \phi(X_i)$
- Covariance: $\Phi\Phi^T$
- Proceed with PCA as usually on $\Phi \in R^{2*r,n}$

PCA on BIG features

- **Now imagine an arbitrarily large polynomial (or other) expansion:**

$$\begin{aligned} \phi(X_i) &= [\phi_1(X_i), \ldots] \\ \Phi &\in R^{q \times n} \end{aligned}$$

- **Covariance :** $\Phi\Phi^T \in R^{q \times q}$
- **Gramian:** $\Phi^T\Phi \in R^{n \times n}$

- **If q > n, use Gramian computation of PCA!**

Kernel Trick

- **For Gramian computation**
  - Expand each $\Phi_i = \phi(X_i)$
  - compute inner products
  - Expensive!
- **Image you can define a "kernel" function such that**
  - $k(X_i, X_i) = \phi(X_i)^T \phi(X_i)$
  - Costs less than expanding and computing dot product
- **Mercer's theorem (roughly) says**
  - Any symmetric semi-positive matrix is the Gramian matrix of inner products for some feature space

Kernel Trick

- **Using the kernel trick we can handle cases where**
  - $q \to \infty$
  - We know $k(X_i, X_i) = \phi(X_i)^T \phi(X_i)$
  - But we don't know $\phi(\cdot)$
- **We can't recover** $V = \Phi U L^{-1/2}$
  - Because we can't compute $\Phi_i = \phi(X_i)$ directly
- **But we can compute projections,**

$$\begin{aligned} y &= \phi(X_i)^T \Phi U L^{-1/2} \\ &= k(X_i, X) U L^{-1/2} \end{aligned}$$

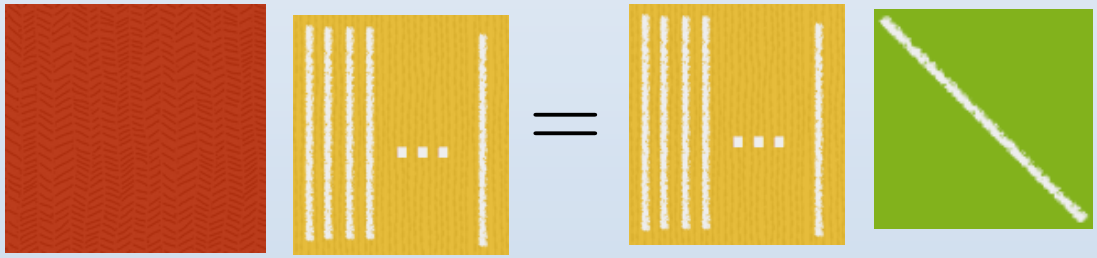
Kernel PCA

- **[Scholkopf, 1997] introduced Kernel PCA**
- **Compute Gramian**

$$G_{i,j} = k(X_i, X_j)$$

G

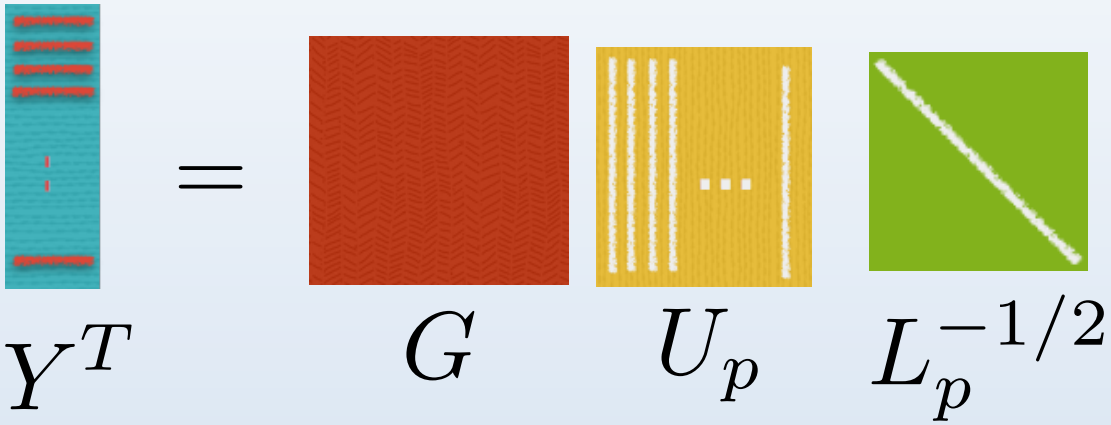
- **Eigen decomposition**



$G \quad U = U \quad L$

Kernel PCA

Project onto p-dimensional subspace



$$Y^T = G \quad U_p \quad L_p^{-1/2}$$

Kernel Examples

- **Gaussian kernel:**

$$k(x, y) = \exp(||x - y||_2^2/\sigma)$$

- **Polynomial kernel:**

$$k(x, y) = (\alpha x^T y + \beta)^\gamma$$

- **Sigmoid kernel**

$$k(x, y) = \tanh(\alpha x^T y + \beta)$$

General Kernel Methods

- **Today I'll be focusing on Kernel PCA (kPCA)**
- **Methods can be used on any other method using the kernel trick**
  - Support Vector Machines (SVM)
  - Gaussian Processes (GP)
  - Etc
- **Because discussion is on efficient approximation of Gramian (Kernel) matrix**

Kernel PCA on large data sets

- **Problems at scale (n is large)**
- **Nystrom method**
- **Previous work in dictionary selection**

Kernel PCA: the ugly

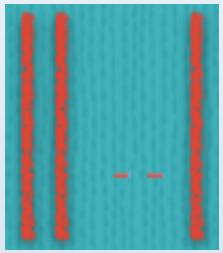
- **Kernel PCA (and other kernel methods) are difficult when n is large**
- **Kernel PCA uses the Gramian**
  - n x n matrix
  - What if n is BIG?
  - $O(n^3)$ decomposition become infeasible

Kernel PCA on large data sets

- **Nystrom method**
  - [Williams and Seeger, 2001]
  - [Drineas and Mahoney, 2005]
  - Maintain a subset of the data as a “dictionary”
  - (Current work)
- **Online Kernel PCA**
  - [Kim et al. 2005]
  - [Kimura et al. 2005]
  - Maintain a rank-k approximation to Gramian
  - Compute pre-image of bases as “dictionary”
  - (Potential future work)
- **Random projection Approximations**
  - [Rahimi and Recht, 2007]
  - [Lopez-Paz, et al. 2014]
  - Use random projection approximation to Gramian
  - (Currently working on this with Jeff Phillips and Mina Ghashami)

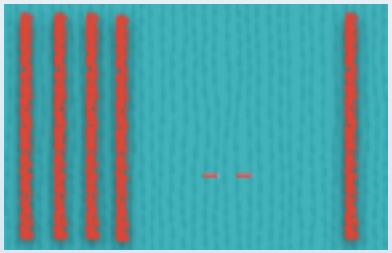
Nystrom method

- **[Williams and Seegers 2001] introduced to Kernel SVM**
  - Use a smaller subset of the data, a “dictionary” to approximate the Gramian



D

$\subset$



X

$$D \in R^{r \times d}$$

$$d \ll n$$

$$\hat{G} = k(X, D)k(D, D)^{-1}k(D, X)$$

$$k(X, D)_{i,j} = k(X_i, D_j)$$

Nystrom method

$$\hat{G} = k(X, D)k(D, D)^{-1}k(D, X)$$

- **This is equivalent to projecting n data onto space spanned by D with basis, V_D**

$$\begin{aligned}
& (V_D \Phi_X)^T (V_D \Phi_X) \\
= \quad & \Phi_X^T \Phi_D U_D^T L_D^{-1/2} L_D^{-1/2} U_D \Phi_D^T \Phi_X \\
= \quad & \Phi_X^T \Phi_D (\Phi_D^T \Phi_D)^{-1} \Phi_D^T \Phi_X \\
= \quad & k(X, D)k(D, D)^{-1}k(D, X)
\end{aligned}$$

Kernel PCA: Nystrom Method

- **Projected data points live in R^d – $O(nd^2)$**

$$Y_D = (V_D \Phi_X)$$
$$Y_D \in R^{d \times n}$$

- **Run PCA on projected data – $O(d^3)$**

$$Y_D Y_D^T W = W S$$

- **$O(nd^2) + O(d^3)$ instead of $O(n^3)$**
  - Good when d << n

Dictionary selection

- **How to select the dictionary ?**
- **Uniform random**
  - [Drineas and Mahoney 2005]
- **Greedy-random methods**
  - [Smola, et al. 2001]
  - [Ouimet, et al. 2005]
- **Sparse Kernel PCA**
  - [Tipping 2001]

Uniform Random Dictionary

- **Select a uniform random subset**
  - Without replacement
  - With replacement
- **[Williams and Seegers, 2001]**
  - Uniform random dictionary
- **[Drineas and Mahoney, 2005]**
  - Analyzed within more general framework of matrix sketching
  - Importance sampling based on norm of data points
    - For stationary kernels (Gaussian) norm is 1,

$$||\phi(x)||_2^2 = k(x,x) = 1$$

Greedy Random

- **Greedily select points randomly (should) improve projection error**
- **[Smola, et al. 2001] "Sparse Greedy Matrix"**
  - Select a uniform random subset, R
  - From R select point z that improves projection error of R onto current dictionary D
  - Add z to D
  - Repeat until convergence
- **[Ouimet, et al. 2005] "Greedy Spectral Embedding"**
  - D = first point
  - Consider every point, z, in X
    - If projection error of z onto current D is large
      - Add z to D

Sparse Kernel PCA

- **[Tipping 2001] introduced Sparse Kernel PCA**
- **Based on Probabilistic PCA [Tipping and Bishop 1999]**
- **Model the covariance in feature space as,**

$$C = \sigma^2 I + \Phi W \Phi^T$$

  - σ - controls sparsity (large value => more sparse)
  - W - diagonal weight matrix, weight corresponds to each data point
- **Compute model using expectation-maximization (EM)**
- **Large σ "soaks up" ambient variance leaving only the most impactful data points with non-zero weight**
- **Elements with non-zero weights are dictionary**

Dictionary selection - problems

- **[Tipping 2001] – takes advantage of the Kernel PCA subspace**
  - Papers using this approach have to subdivide problem because of speed (EM convergence)
  - Non-intuitive interaction between sparsity and σ parameter
- **[Smola, et al. 2001], [Ouimet, et al. 2005] – doesn't use kernel PCA subspace to select dictionary (greedy)**
  - Greedy nature leads to larger dictionaries
  - Build dictionary first, then compute kernel PCA

Proposed approach: Lasso Dictionary Selection

- **New sparse kernel PCA**
  - Based on sparse PCA by [Zou, et al. 2006]
  - Uses L1 regularization to induce sparsity
  - Make use of kernel PCA subspace for dictionary selection (optimal instead of greedy)
  - Intuitive parameterization (size of dictionary)
  - Fixed number of iterations (size of dictionary)
- **Propose using a two-stage dictionary selection**
  - First rough pass (using random or greedy methods)
  - Second careful pass selecting dictionary based on kernel PCA subspace

Sparse PCA

- **Proposed by [Zou, et al. 2006]**
- **Motivation**
  - running PCA on high dimensional data (analysis of genes)
  - Sparse PCA means less data points needed for analysis
- **Run PCA, then make the loadings sparse**
- **Formulation as L1 regularized optimization problem,**

$$b_i = \arg\min_b ||X^T(v_i - b)||_2^2 + \lambda||b||_2 + \lambda||b||_1$$

- **Elastic-net problem**
- **Solved using Least Angles Regression (LARS)**
- **Each principal vector independently**

L1 regularization inducing sparsity

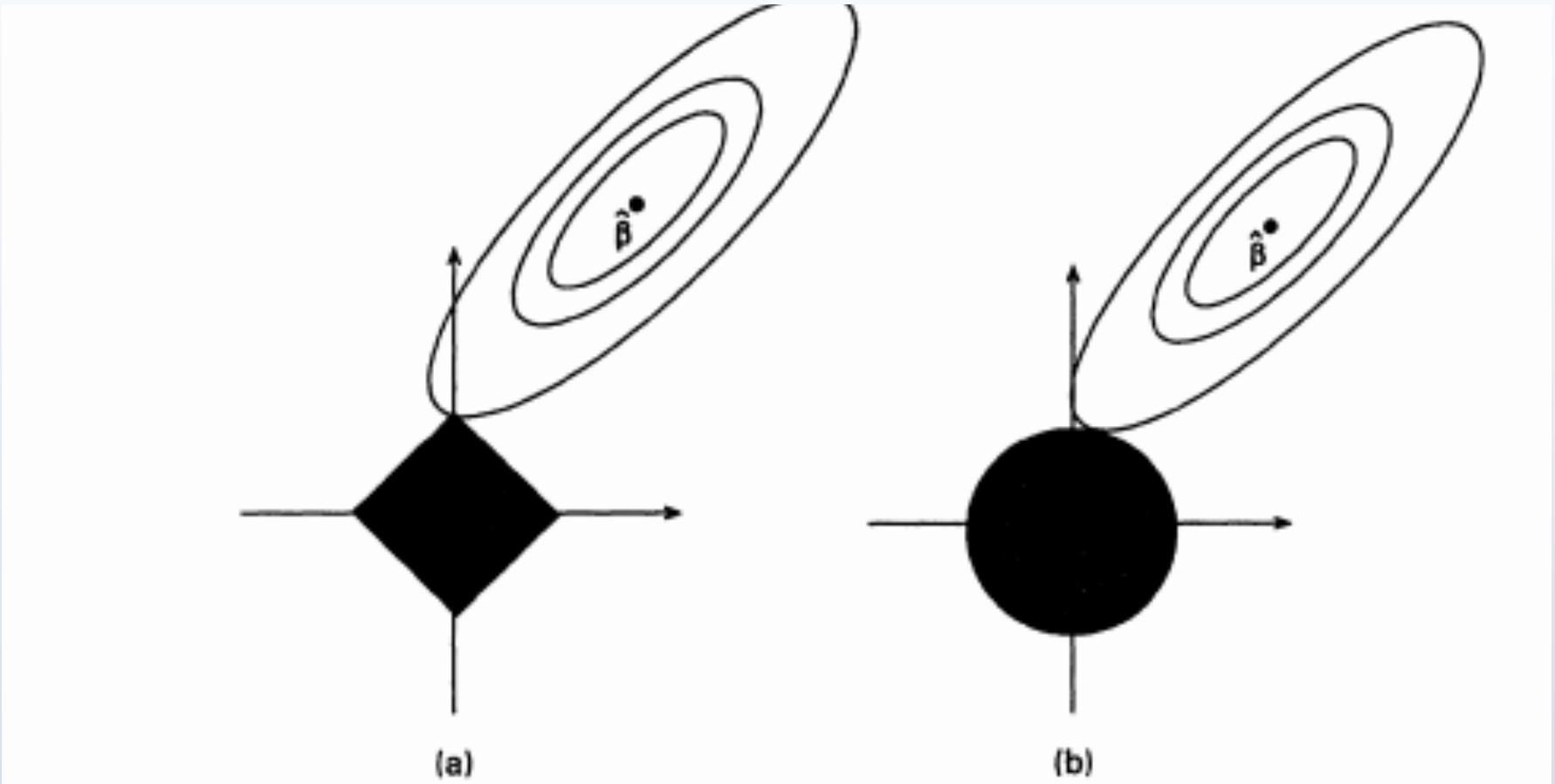


Fig. 2. Estimation picture for (a) the lasso and (b) ridge regression

[image from quora.com]

Kernel Sparse PCA

We modified [Zou, et al. 2006] to use the kernel trick,

- **Run kernel PCA**
- **Make each of the loadings sparse,**

$$b_i = \arg\min_b ||\Phi^T(v_i - b)||_2^2 + \lambda||b||_1$$

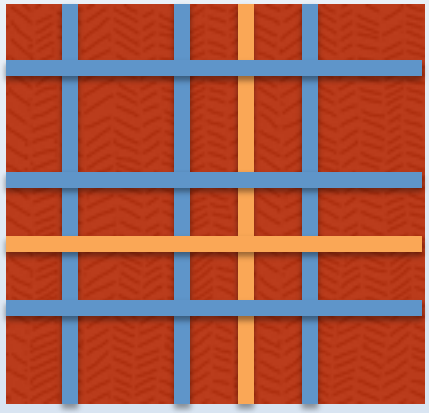
$$w_i, s_i = \arg\min_{w,l} ||\Phi^T(\Phi u_i l_i^{-1/2} - \Phi w s^{-1/2})||_2^2 + \lambda||w||_1$$

$$w_i, s_i = \arg\min_{w,l} ||G(u_i l_i^{-1/2} - w s^{-1/2})||_2^2 + \lambda||w||_1$$

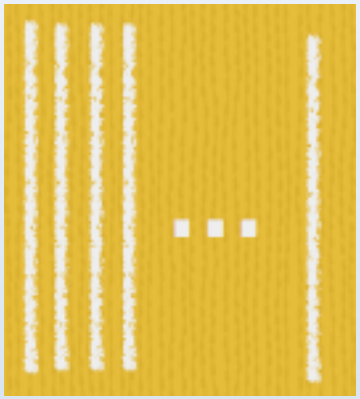
- **In kernel case r = n, so LARS solution to elastic net becomes the same as Lasso solution**
- **Make loadings sparse for each PCA vector independently**

Kernel Sparse PCA: problems

- **Sparse solution is found for each loading vector independently**
  - Zero-loadings in each vector may not match up!



G



W

$S^{-1/2}$

- **Suppose orange components only have one loading vector with non-zero weight?**

Kernel Sparse PCA: problems

- **This results in a sub-optimal dictionary**
- **Can we modify the optimization to consider all loading vectors at once?**

Kernel Sparse PCA: problems

- **This results in a sub-optimal dictionary**
- **Can we modify the optimization to consider all loading vectors at once?**

$$\arg\min_{W,S} ||K(UL^{-1/2} - WS^{-1/2})||_2^2 + \lambda \sum_{i=1}^{n} ||W^i||_2$$

- **W^i is the i^{th} row of W**

Lasso Dictionary Selection

Input:

- **G (Gramian),**
- **U,S (eigen decomp of G)**
- **Find optimum using LARS:**

$$\arg\min_{W,S} ||K(UL^{-1/2} - WS^{-1/2})||_2^2 + \lambda \sum_{i=1}^{n} ||W^i||_2$$

Output:

- **D – non-zero entries of W**

Two-step approach

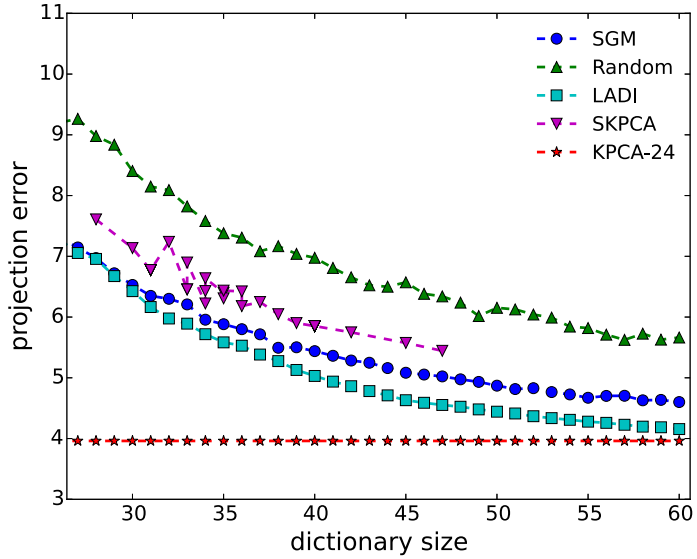
1. **Find a subset of size m << n that preserves statistics of full data set**
   - Small enough to perform kernel PCA
   - Large enough to captures statistics of full dat aset
2. **Run Lasso dictionary selection**
   - Optimal dictionary of size d for subset of size m

Use dictionary D to approximate full Gramian

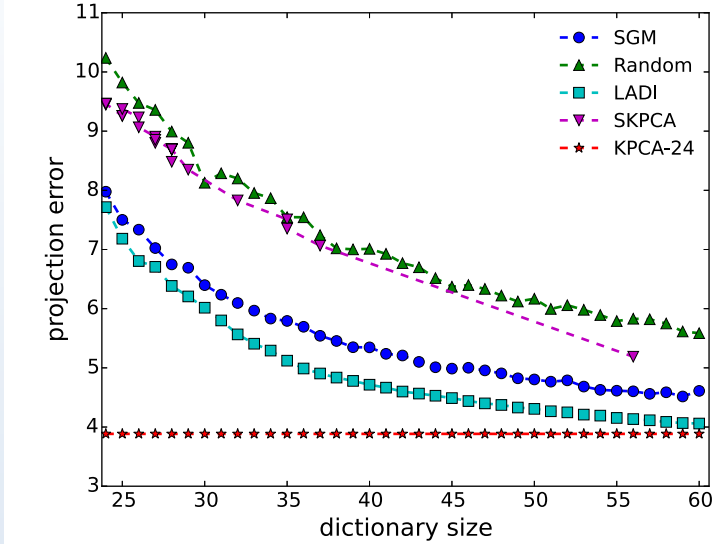
Comparison

- **Data sets:**
  - UCI data sets using Gaussian kernel
  - σ chosen according to average inter-point distance
- **For fairness chose the same first stage subset for each method**
- **Compare projection error of resulting dictionaries**

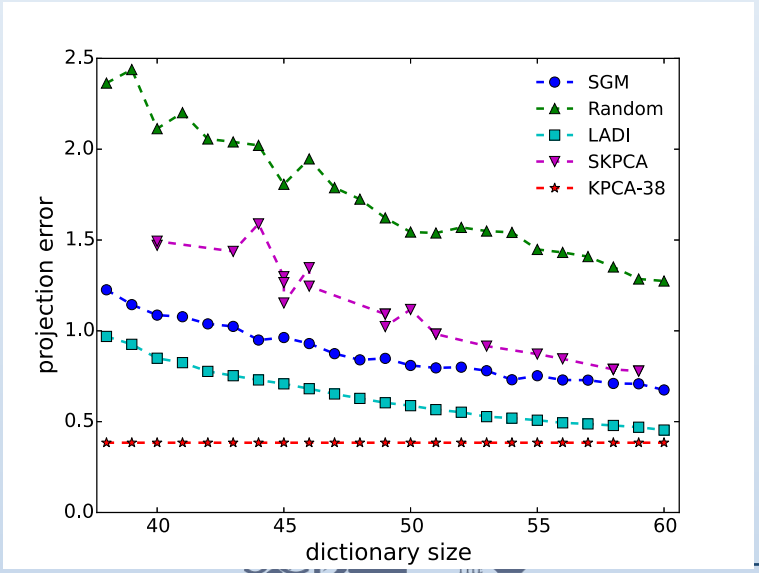
Synthetic Gaussian (varying SNR)



SNR 0.1

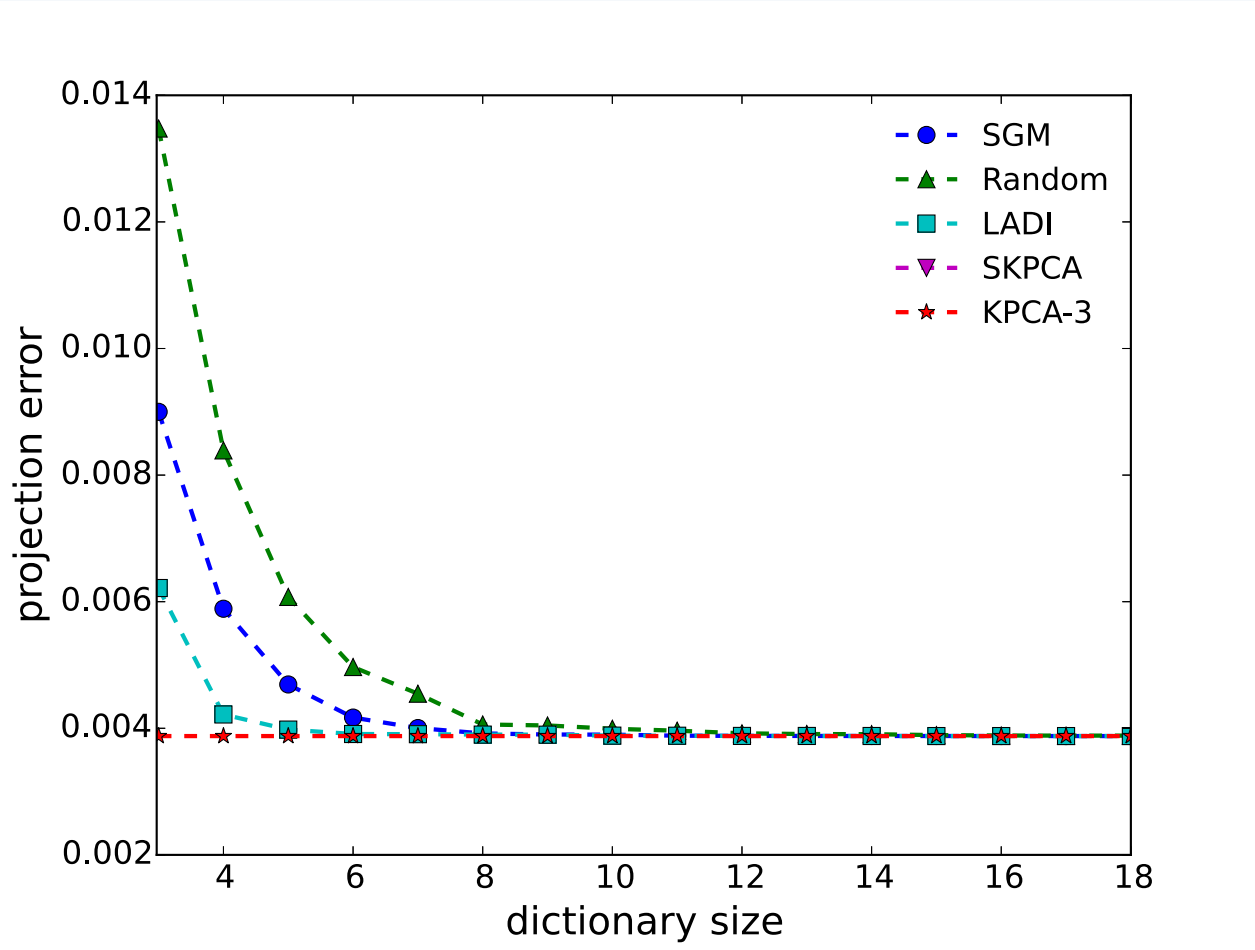


SNR 1.0

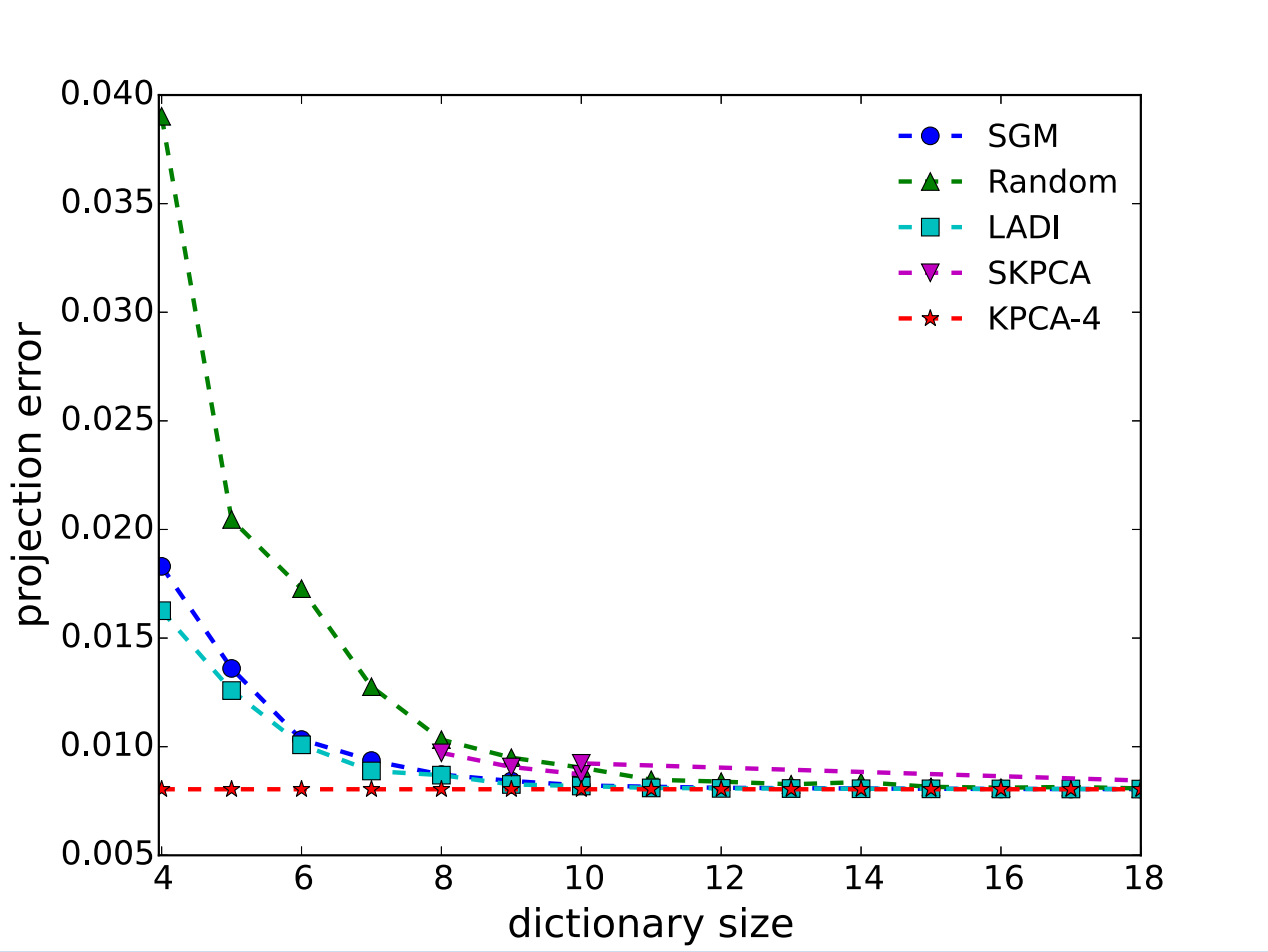


SNR 10.0

UCI Cloud data set



UCI Forest cover type



UCI Solar flare