Online PCA in High Dimension: Which Algorithm To Choose?

David Degras

UMass Boston

October 26, 2018

Talk based on article "Online PCA in High Dimension: Which Algorithm to Choose?" (*International Statistical Review*, 2018) with Hervé Cardot (Université de Bourgogne)

Outline

PCA

- Principal component analysis (PCA): a powerful tool for reducing the dimension of multivariate data.
- Idea: represent the data with a small number of linear combinations of the original variables that retain as much as possible of the data variation.
- Applications to exploratory data analysis, feature extraction, pattern recognition, data compression, tracking and more.

Example PCA

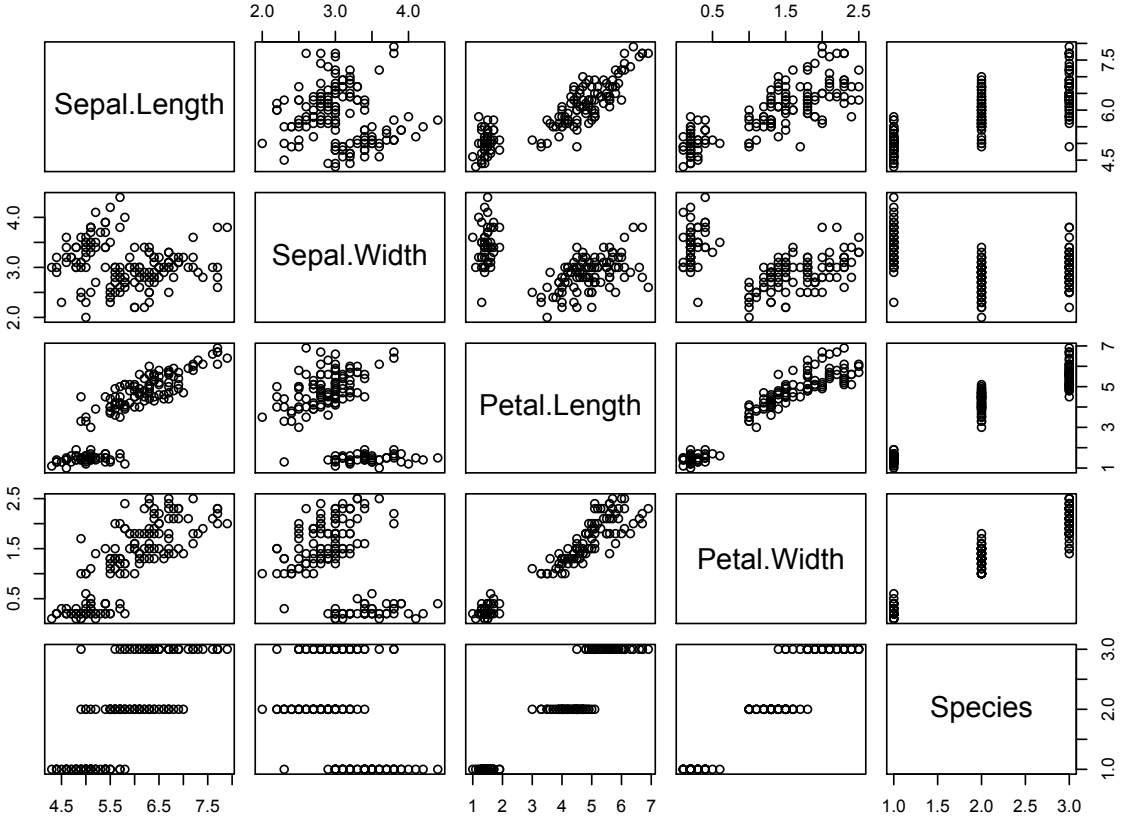
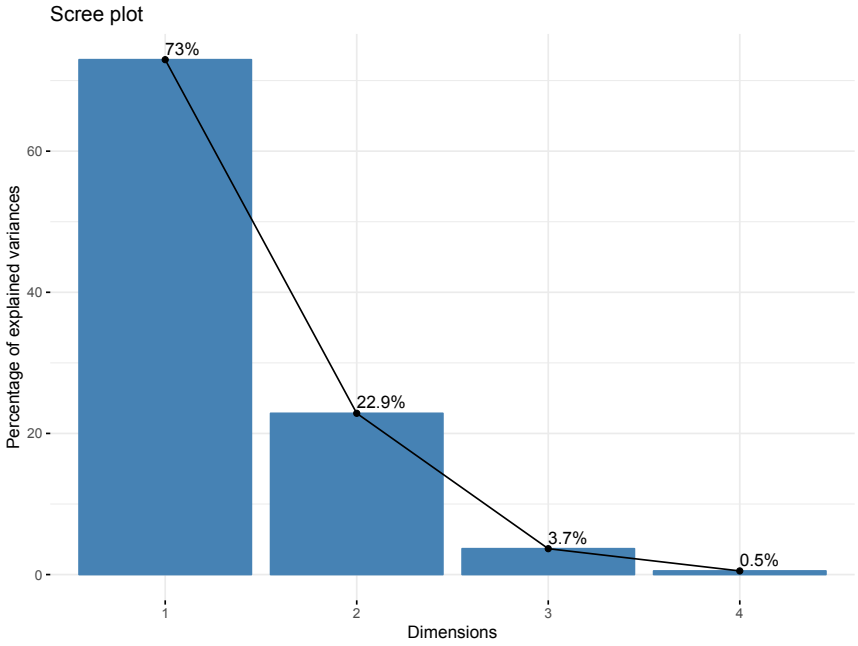


Figure: Iris dataset (Anderson, 1935; Fisher, 1936). Sepal length and width and petal length and width for 50 flowers from each of 3 species of iris (setosa, versicolor, and virginica). Measurements in centimeters.

Example PCA



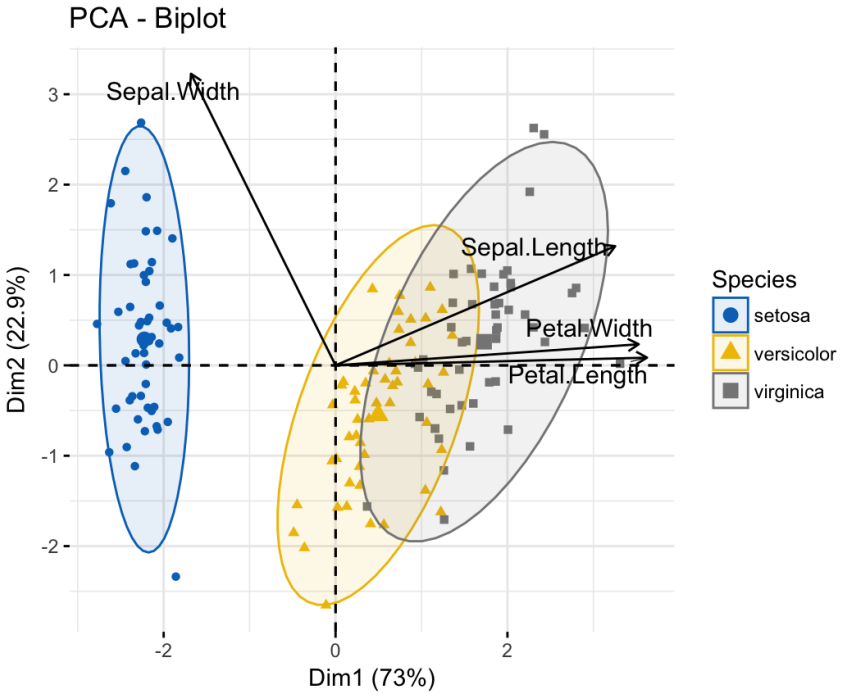


Figure: Left: scree plot of (scaled) eigenvalues of covariance matrix. Right: biplot of individuals and variables along first two PCs. Graphs obtained with R packages `FactoMineR` and `factoextra`.

Math Formalization

- Observe data vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$.
- PCA: find orthogonal set of vectors $\{\varphi_1, \ldots, \varphi_q\}$ $(q \leq p)$ that best approximates the (centered) data. Mathematically, minimize

$$\sum_{i=1}^{n} \left\| (\mathbf{x}_i - \bar{\mathbf{x}}) - \sum_{k=1}^{q} \langle \varphi_k, (\mathbf{x}_i - \bar{\mathbf{x}}) \rangle \varphi_k \right\|^2$$

under the constraints $\langle \varphi_k, \varphi_l \rangle = \delta_{kl}$, with $\| \cdot \|$ and $\langle \cdot, \cdot \rangle$ the Euclidean norm and scalar product in $\mathbb{R}^d$, δ_{kl} the Kronecker symbol, and $\bar{\mathbf{x}} = (1/n) \sum_{i=1}^{n} \mathbf{x}_i$.

Solution

- In matrix notations, the PCA problem expresses as

$$\min_{\mathbf{U}\in\mathbb{R}^{d\times q}} \left\|(\mathbf{I}_d - \mathbf{U}\mathbf{U}^T)\mathbf{X}\right\|_F^2$$

subject to the constraint $\mathbf{U}^T\mathbf{U} = \mathbf{I}_q$, where $\mathbf{U} = (\varphi_1, \ldots, \varphi_q)$, $\mathbf{X} = (\mathbf{x}_1 - \bar{\mathbf{x}}, \ldots, \mathbf{x}_n - \bar{\mathbf{x}})$, $\mathbf{I}$ is the identity matrix, and $\|\cdot\|_F$ is the Frobenius norm.

- One solution: take $\mathbf{U}$ as eigenvectors associated with q largest eigenvalues of data covariance matrix $\mathbf{C} = n^{-1}\mathbf{X}\mathbf{X}^T$.
That is, $\mathbf{C}\varphi_k = \lambda_k\varphi_k$ for $1 \leq k \leq q$ with $\lambda_1 \geq \cdots \lambda_d \geq 0$.

- The solution vectors $\varphi_1, \ldots, \varphi_q$ are called *principal components*. They are not unique ($\mathbf{U}_q$ can be rotated). However the linear space they span ($\mathbf{U}_q\mathbf{U}_q^T$) is unique.

Computation

- The eigenvectors and eigenvalues of $\mathbf{C}$ can be obtained by eigenvalue decomposition (EVD) of $\mathbf{C}$. Equivalently they can be obtained by singular value decomposition (SVD) of $\mathbf{X}$.

Reminder

if $\mathbf{X} = \widetilde{\mathbf{U}}\mathbf{D}\mathbf{V}$ is the SVD of $\mathbf{X}$, then $\mathbf{C} = n^{-1}\widetilde{\mathbf{U}}\mathbf{D}^2\widetilde{\mathbf{U}}^T$. Hence the eigenvalues and eigenvectors of $\mathbf{C}$ are the diagonal coefficients of $n^{-1}\mathbf{D}^2$ and the columns of $\widetilde{\mathbf{U}}$, respectively.

- Time complexity: $O(\min(n^2 d, nd^2))$ for standard SVD (see e.g. Golub and Van Loan, 1996), $O(\min(n^3, d^3))$ for full EVD ($q = d$). If $q \ll d$, time complexity can be significantly reduced with reduced-rank SVD/EVD (e.g. Arnoldi iteration, Krylov subspace, Lanczos algorithm...). R package: `RSpectra`.

Computation

- EVD: $O(\min(n^2, d^2))$ memory required to store $\mathbf{C}$ or $n^{-1}\mathbf{X}^T\mathbf{X}$.
- Because of limited computing and storage resources, standard SVD/EVD implementation of PCA (**batch PCA** or **offline PCA**) is impractical for massive data sets (large n and/or large d) and for time-varying data (e.g. data streams).
- Solution: for time-varying data, perform initial batch PCA on small data subset, then efficiently update current PCA when new data arrive. For massive datasets, divide the data into chunks and apply same idea. This is the principle of **online PCA**.

Online PCA

- Many online PCA algorithms studied in statistics, machine learning, signal processing, numerical analysis... but which one to use in practice?
- The answer depends on the analytic goal (eigenvector/eigenvalue estimation, subspace tracking, monitoring, ...) and the available resources (data, storage capacity, RAM, time).
- No dedicated software for online PCA and no systematic comparative numerical study found in the literature.

Goals of this work

1. Software implementation of standard online PCA algorithms. R package `onlinePCA` available on CRAN.
2. Comparison of online PCA algorithms in terms of statistical accuracy, memory usage, and computational speed.

Notation

Sample mean and sample covariance of $\mathbf{x}_1, \ldots, \mathbf{x}_n$:

$$\bar{\mathbf{x}}_n = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i\,, \qquad \mathbf{C}_n = \frac{1}{n}\sum_{i=1}^{n} (\mathbf{x}_i - \bar{\mathbf{x}}_n)(\mathbf{x}_i - \bar{\mathbf{x}}_n)^T$$

(Term $1/n$ in $\mathbf{C}_n$ can replaced by $1/(n-1)$ to get standard sample covariance)

Principle

- Recursive expression of sample mean and sample covariance:

$$\bar{\mathbf{x}}_{n+1} = \frac{n}{n+1}\,\bar{\mathbf{x}}_n + \frac{1}{n+1}\,\mathbf{x}_{n+1},$$
$$\mathbf{C}_{n+1} = \frac{n}{n+1}\mathbf{C}_n + \frac{n}{(n+1)^2}\left(\mathbf{x}_{n+1} - \bar{\mathbf{x}}_n\right)\left(\mathbf{x}_{n+1} - \bar{\mathbf{x}}_n\right)^T.$$

Similar formula available w.r.t. $\mathbf{x}_{n+1}, \ldots, \mathbf{x}_{n+r}$, $r \geq 2$.

- The fact that $\mathbf{C}_{n+1}$ is a rank 1 modification of $\mathbf{C}_n$ can be exploited to efficiently update an existing PCA.

Online PCA

- Suppose that $\mathbf{C}_n = \mathbf{U}_n \mathbf{D}_n \mathbf{U}_n'$ with $\mathbf{U}_n$ orthogonal (eigenvectors/PC) and $\mathbf{D}_n$ diagonal (eigenvalues). For a new data vector $\mathbf{x}_{n+1}$,

$$\begin{aligned}\mathbf{C}_{n+1} &= \mathbf{U}_{n+1}\mathbf{D}_{n+1}\mathbf{U}_{n+1}^T \\ &= \mathbf{U}_n \left(\mathbf{\Lambda}_n + \rho_n \mathbf{z}_{n+1}\mathbf{z}_{n+1}^T\right) \mathbf{U}_n^T\end{aligned}$$

with $\mathbf{\Lambda}_n = \frac{n}{n+1}\mathbf{D}_n$, $\rho_n = \frac{n}{(n+1)^2}$ and $\mathbf{z}_n = \mathbf{U}_n^T(\mathbf{x}_{n+1} - \bar{\mathbf{x}}_{n+1})$.

- The PCA update reduces to finding the EVD of $\mathbf{\Lambda}_n + \rho_n \mathbf{z}_n \mathbf{z}_n^T$, say $\tilde{\mathbf{U}}_n \tilde{\mathbf{D}}_n \tilde{\mathbf{U}}_n^T$. Then $\mathbf{U}_{n+1} = \mathbf{U}_n \tilde{\mathbf{U}}_n$ and $\mathbf{D}_{n+1} = \tilde{\mathbf{D}}_n$.

Perturbation methods

- For n large, the diagonal matrix $\mathbf{\Lambda}_n$ dominates $\rho_n \mathbf{z}_n \mathbf{z}_n^T$. In this case, perturbation theory provides approximate EVD for $\mathbf{\Lambda}_n + \rho_n \mathbf{z}_n \mathbf{z}_n^T$.
- For convenience, drop n from notations when possible.

Hegde et al. (2006)

Write $\tilde{\mathbf{D}} \approx \mathbf{\Lambda} + \mathbf{P}_{\mathbf{\Lambda}}$ and $\tilde{\mathbf{U}} \approx \mathbf{I} + \mathbf{P}_{\tilde{\mathbf{U}}}$, with $\mathbf{P}_{\mathbf{\Lambda}}$ and $\mathbf{P}_{\tilde{\mathbf{U}}}$ first-order perturbation matrices. Then

$$\left[\mathbf{P}_{\mathbf{\Lambda}}\right]_{ij} = \delta_{ij} \rho z_i^2,$$

$$\left[\mathbf{P}_{\tilde{\mathbf{U}}}\right]_{ij} = \frac{(1-\delta_{ij})\rho z_i z_j}{d_j + z_j^2 - d_i^2 - z_i}.$$

Secular equations

- Extensive work on secular equations in numerical analysis: Golub (1973), Bunch et al. (1976), Gu and Eisenstadt (1994), Li et al. (2000)...

If the diagonal coefficients of $\mathbf{\Lambda}$ are distinct, say $d_1 < \ldots < d_n$, then the eigenvalues $\lambda_1 \leq \ldots \leq \lambda_n$ of $\mathbf{\Lambda} + \rho \mathbf{z}\mathbf{z}^T$ are the roots of the equation

$$1 + \rho^2 \sum_{i=1}^{n} \frac{z_i^2}{d_i - \lambda} = 0$$

and $d_1 < \lambda_1 < d_2 < \lambda_2 < d_3 < \ldots < d_n < \lambda_n$ (interlacing property).

The corresponding eigenvectors are $(\mathbf{\Lambda} - \lambda_i \mathbf{I})^{-1}\mathbf{z}$, $i = 1, \ldots, n$.

Incremental PCA

- Number of required PCs (q) usually much smaller than data dimension d.
- Significant speedup can be achieved in online PCA by computing only q PCs instead of all d. Tradeoff with statistical accuracy.
- Schemes for updating approximate reduced rank PCA: Brand (2002), Arora et al. (2012), ...

Incremental PCA

- Write $\mathbf{C}_n \approx \mathbf{U}_n \mathbf{D}_n \mathbf{U}_n^T$ where $\mathbf{U}_n$ and $\mathbf{D}_n$ approximate the first q eigenvectors and eigenvalues of $\mathbf{C}_n$, respectively. Matrix $\mathbf{U}_n$ of size $d \times q$ with $\mathbf{U}_n^T \mathbf{U}_n = \mathbf{I}_q$, matrix $\mathbf{D}_n$ diagonal of size $q \times q$.
- Now

$$\begin{aligned}\mathbf{C}_{n+1} &\approx \frac{n}{n+1}\left(\mathbf{U}_n \mathbf{D}_n \mathbf{U}_n^T\right) + \frac{1}{n}\widetilde{\mathbf{x}}_{n+1}\widetilde{\mathbf{x}}_{n+1}^T \\ &= \frac{1}{n}\begin{bmatrix}\mathbf{U}_n & \dfrac{\widetilde{\mathbf{x}}_{n+1}^{\perp}}{\left\|\widetilde{\mathbf{x}}_{n+1}^{\perp}\right\|}\end{bmatrix}\begin{pmatrix}\frac{n}{n+1}\mathbf{D}_n + \mathbf{c}_{n+1}\mathbf{c}_{n+1}^T & \left\|\widetilde{\mathbf{x}}_{n+1}^{\perp}\right\|\mathbf{c}_{n+1} \\ \left\|\widetilde{\mathbf{x}}_{n+1}^{\perp}\right\|\mathbf{c}_{n+1}^T & \left\|\widetilde{\mathbf{x}}_{n+1}^{\perp}\right\|^2\end{pmatrix}\begin{bmatrix}\mathbf{U}_n & \dfrac{\widetilde{\mathbf{x}}_{n+1}^{\perp}}{\left\|\widetilde{\mathbf{x}}_{n+1}^{\perp}\right\|}\end{bmatrix}^T\end{aligned}$$

- EVD of the middle matrix provides best rank $q+1$ approximation to $\frac{n}{n+1}\left(\mathbf{U}_n \mathbf{D}_n \mathbf{U}_n^T\right) + \frac{1}{n}\widetilde{\mathbf{x}}_{n+1}\widetilde{\mathbf{x}}_{n+1}^T$.

Stochastic gradient algorithms

- **Gradient algorithms:** minimize empirical loss function $\sum_{i=1}^{n} l_i(\theta)$ by solving $\sum_i \nabla l_i(\theta) = 0$. Computationally intractable for large n.

- **Stochastic gradient algorithms:** consider vector instances sequentially. At each iteration i, move the current solution $\hat{\theta}_i$ along the gradient $\nabla l_i(\theta)$.

- Stochastic gradient algorithms converge to roots of the gradient of a suitable loss function $l(\theta)$. For PCA, $l(\theta) = E(\|(\mathbf{I} - \mathbf{U}\mathbf{U}^T)\mathbf{x}\|^2)$ $= \mathrm{tr}(\mathbf{\Gamma}) - \mathrm{tr}(\mathbf{U}^T\mathbf{\Gamma}\mathbf{U})$ where $\theta = \mathbf{U} \in \mathbb{R}^{d\times q}$, $\mathbf{U}^T\mathbf{U} = \mathbf{I}_q$, and $\mathbf{x}$ is a random vector with mean zero and covariance matrix $\mathbf{\Gamma}$.
  They can be efficiently implemented as **neural networks** (Sanger, 1989; Oja, 1992).

Stochastic gradient algorithms

- Setup: for sample size n, PC matrix $\mathbf{U}_n$ of dimensions $d \times q$.
- For new vector $\mathbf{x}_{n+1}$, update $\mathbf{U}_n$ as follows:

Oja (1992)

1. $\tilde{\mathbf{U}}_n = \mathbf{U}_n + \gamma_n \widetilde{\mathbf{x}}_{n+1} \widetilde{\mathbf{x}}_{n+1}^T \mathbf{U}_n$, with $\gamma_n > 0$ a gain parameter ($\gamma_n = c/n^a$).
2. $\mathbf{U}_{n+1} = \tilde{\mathbf{U}}_n \mathbf{S}_n^{-1}$ where the matrix $\mathbf{S}_n^{-1}$ orthogonalizes $\tilde{\mathbf{U}}_n$.

$\mathbf{S}_n$ can be chosen to perform the Graham-Schmidt procedure (**Stochastic Gradient Ascent**) or as $\mathbf{S}_n = \left(\tilde{\mathbf{U}}_n^T \tilde{\mathbf{U}}_n\right)^{1/2}$ (**Subspace Network Learning**), among others.

Other stochastic algorithms

- Generalized Hebbian Algorithm (Sanger, 1989). Similar to SGA, with convenient NN implementation.
- Candid Covariance-Free Incremental PCA (Weng et al., 2003). Similar to SGA & GHA but uses deflation technique. Does not require choosing gain parameters.
- Randomized online PCA (Warmuth and Kuzmin, 2008). Bounds on expected compression loss. Very slow: $O(d^2)$ per new instance.

Summary

Table: Comparison of online PCA algorithms: time/space required per data vector, necessity to select tuning parameters, orthogonality of eigenvectors.

Method	Complexity	Memory	Tuning	Orthogonality
Batch	$O(\min(nd^2, n^2d))$	$O(nd + d^2)$	No	Exact
Perturbation	$O(d^2)$	$O(d^2)$	No	Approx.
Secular	$O(d^2)$	$O(d^2)$	No	Approx.
SGA,SNL,GHA	$O(qd)$	$O(qd)$	Yes	Exact
SGA,SNL,GHA (NN)	$O(qd)$	$O(qd)$	Yes	Approx.
Candid cov.-free	$O(qd)$	$O(qd)$	No	Approx.
Incremental	$O(q^2d)$	$O(qd)$	No	Exact
Incremental (no $\perp$)	$O(qd)$	$O(qd)$	No	Approx.

R package

Online Principal Component Analysis

Documentation for package 'onlinePCA' version 1.3.1

- DESCRIPTION file.
- Package NEWS.

Help Pages

onlinePCA-package	Online Principal Component Analysis
batchpca	Batch PCA
bsoipca	Block Stochastic Orthononal Iteration (BSOI)
ccipca	Candid Covariance-Free Incremental PCA
coef2fd	Recover functional data from their B-spline coefficients
create.basis	Create a smooth B-spline basis
fd2coef	Compute the coefficients of functional data in a B-spline basis
ghapca	Generalized Hebbian Algorithm for PCA
impute	BLUP Imputation of Missing Values
incRpca	Incremental PCA
incRpca.block	Incremental PCA with Block Update
incRpca.rc	Incremental PCA With Reduced Complexity
perturbationRpca	Recursive PCA using a rank 1 perturbation method
secularRpca	Recursive PCA Using Secular Equations
sgapca	Stochastic Gradient Ascent PCA
snlpca	Subspace Network Learning PCA
updateCovariance	Update the Sample Covariance Matrix
updateMean	Update the Sample Mean Vector

Features

- Batch PCA (data- or covariance-based)
- Online PCA algorithms: BSOI, CCI, GHA, IPCA, SGA, SNL, secular equations, perturbation method
- Missing data imputation method (BLUP)
- Updating functions for sample mean and sample covariance
- Functional data methods (B-spline basis,...)

Usage

- Most online PCA functions `ccipca`, `ghapca`, `incRpca`, `perturbationRpca`, `secularRpca`, `sgapca`, `snlpca` take the same first three arguments: `lambda` (eigenvalues), `U` (eigenvectors/PCs), and `x` (new data). The outputs of these functions can be used as inputs to the same function to update the PCA when new data arrive.
- Next function arguments (number of PCs, centering flag, sample size, ...) are method-specific parameters.
- Package available at
  `https://cran.r-project.org/package=onlinePCA`
  `https://github.com/ddegras/Online-PCA.git`

Simulation 1: computation time

- Generate n trajectories of standard Wiener process on uniform grid of size d in $[0, 1]$ (covariance matrix $\mathbf{\Gamma} = (\min(k/d, l/d)))$.
- Run online PCA on all $n = 100$ vectors with arbitrary initialization.

	$d = 10$		$d = 100$		$d = 1000$		
	$q = 2$	$q = 5$	$q = 5$	$q = 20$	$q = 5$	$q = 20$	$q = 100$
SGA.ex	0.015	0.014	0.015	0.019	0.023	0.077	2.235
SGA.nn	0.014	0.014	0.014	0.016	0.019	0.049	0.293
SNL.ex	0.021	0.023	0.025	0.044	0.030	0.111	1.006
SNL.nn	0.014	0.013	0.014	0.015	0.017	0.051	0.334
GHA	0.009	0.009	0.010	0.011	0.013	0.039	0.264
CCIPCA	0.009	0.009	0.010	0.012	0.017	0.048	0.190
IPCA	0.014	0.014	0.016	0.028	0.024	0.073	0.505
Perturbation	0.016	0.014	0.259	0.132	15.273	13.379	13.365
Secular	0.120	0.103	1.361	1.291	57.241	53.846	54.280

Table: Computation time (seconds)

Simulation 2: statistical accuracy

- Same setup as simulation 1 (standard Wiener process).
- Goal: estimate eigenspace generated by first $q = 5$ eigenvectors/PC.
- Initialize online PCA with batch PCA of $n_0 = 250$ units. For accurate estimation, use $q_{work} = 2q = 10$ and discard the 5 smallest PCs.
- For SGA & GHA, gain parameter taken as $\gamma = c/n^a$ with $a \in \{2/3, 1\}$ and best constant $c = c(n, d, a)$.
- Error measure: normalized squared distance between projectors onto estimated and true eigenspace: $\frac{1}{q}\|\hat{\mathbf{P}}_q - \mathbf{P}_q\|_F^2$

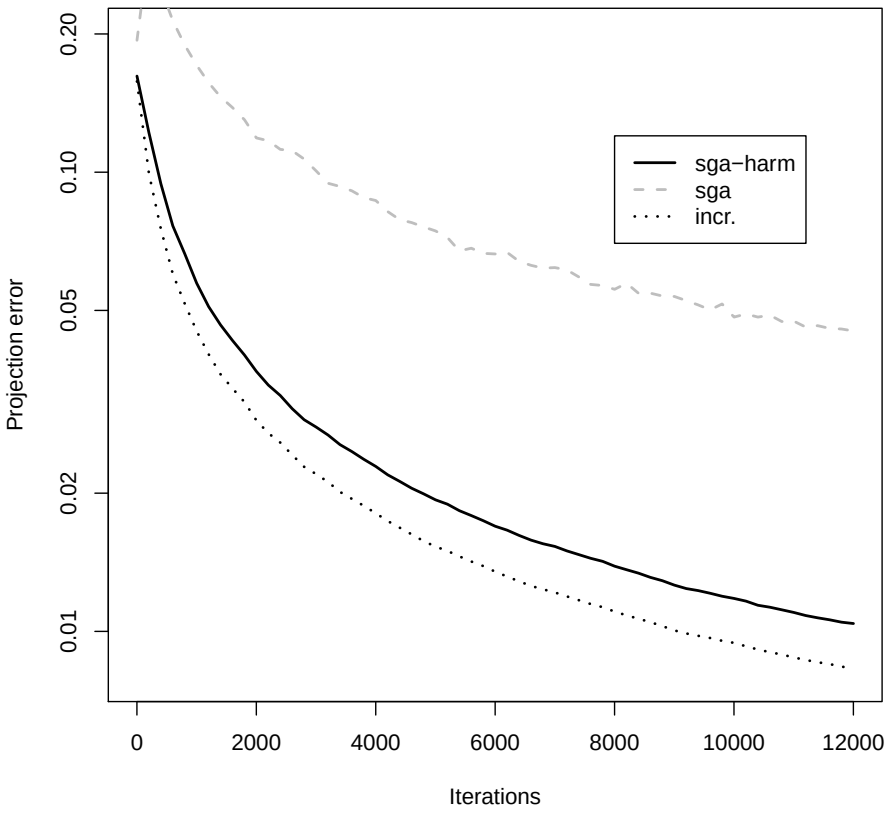
Simulation 2: statistical accuracy

	$n = 500$			$n = 1000$	
	$d = 10$	$d = 100$	$d = 1000$	$d = 10$	$d = 100$
Batch (n_0)	0.044	0.050	0.029	0.041	0.032
Batch (n)	0.019	0.014	0.018	0.009	0.007
SGA ($a = 1$)	0.033	0.042	0.022	0.024	0.017
SGA ($a = 2/3$)	0.035	0.043	0.023	0.026	0.018
GHA ($a = 1$)	0.033	0.042	0.021	0.023	0.016
GHA ($a = 2/3$)	0.035	0.043	0.022	0.025	0.018
CCIPCA	0.028	0.018	0.017	0.018	0.011
IPCA	0.019	0.014	0.018	0.009	0.007
Perturbation	0.522	1.742	1.997	0.560	1.781
Secular	0.019	0.014	0.012	0.009	0.007

Table: Statistical accuracy of online PCA methods (average over 20 replications).

Simulation 3: convergence

- Same setup as simulations 1 & 2. Examine eigenspace estimation error $\frac{1}{q}\|\hat{\mathbf{P}}_q(i) - \mathbf{P}_q\|_F^2$ and compression loss $\sum_{i=1}^n \|\left(\hat{\mathbf{P}}_q(i) - \mathbf{I}_d\right)(\mathbf{x}_i - \bar{\mathbf{x}}_i)\|^2$.



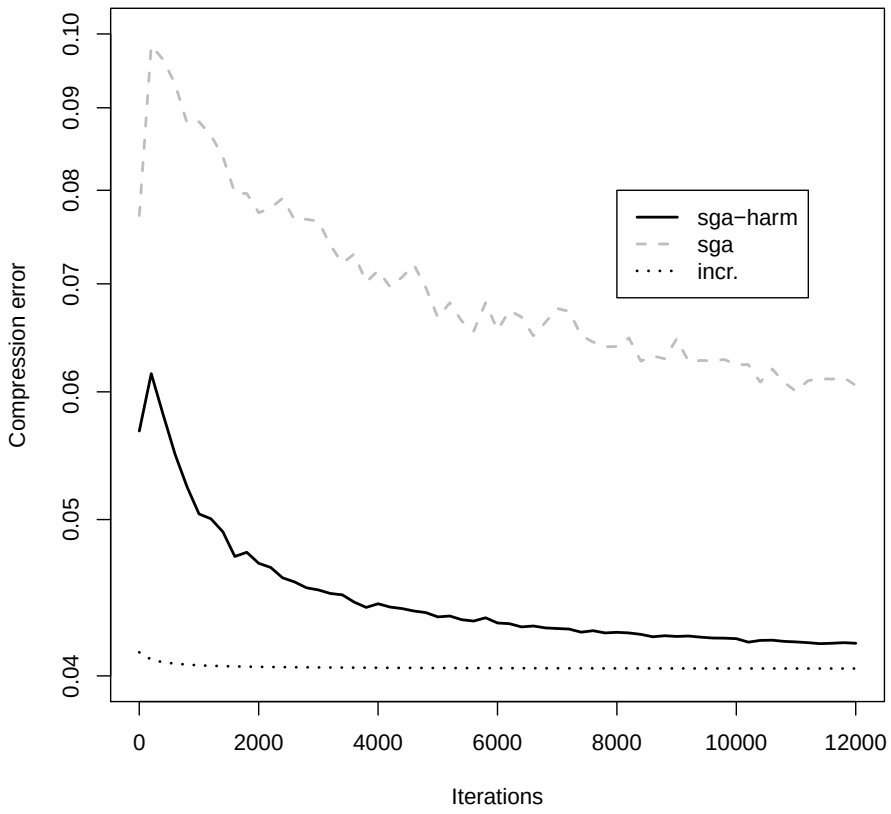


Figure: Left: Eigenspace error for $d = 100$ and $q = 5$. Right: Compression loss for $d = 1000$ and $q = 5$. Methods: incremental PCA (`inc.`) and SGA (`sga-harm` when $a = 1$ and `sga` when $a = 2/3$).

AT&T Database of Faces

- Database of Faces of the AT&T Laboratories Cambridge (`http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html`)
- 400 face images of dimensions 92×112 pixels in 256 gray levels.
- For each of 40 subjects, 10 different images featuring various facial expressions and facial details.

Methodology

- Database randomly split in training set and test set by stratified sampling. For each subject, 9 training images and one test image.
- The IPCA, SGA and CCIPCA algorithms were applied to the (vectorized) training images using $q = 20$ or $q = 40$ PCs as in Levy & Lindenbaum (2000). No image centring (uncentred PCA).
- PCs used for two tasks: compression and classification of the test images.
- Batch PCA taken as benchmark.

Results: computation

Method	Time	Memory
Batch PCA	7.13	924.4
IPCA	7.09	73.3
CCIPCA	3.93	74.6
SGA	9.45	67.8

Table: Computation time (seconds) and memory usage (MB).

Results: data compression

	$q = 20$		$q = 40$	
	Training	Test	Training	Test
Batch	0.0323	0.0363	0.0224	0.0286
IPCA	0.0327	0.0367	0.0229	0.0290
SGA	0.0523	0.0551	0.0388	0.0431
CCIPCA	0.0335	0.0373	0.0257	0.0312

Table: Compression loss

Results: face recognition

	$q = 20$		$q = 40$	
	Training	Test	Training	Test
Batch	0.9897	0.9580	0.9986	0.9880
IPCA	0.9915	0.9635	0.9995	0.9875
CCIPCA	0.9920	0.9655	0.9988	0.9837
SGA	0.9788	0.9340	0.9963	0.9710

Table: Classification rates

Conclusions

- **Perturbation method:** extremely inaccurate, not recommended.
- **Secular equations:** too slow for large d, very accurate for small d.
- **Stochastic methods:** SGA, SNL & GHA very fast BUT sensitive to choice of gain parameter. Trial-and-error required.
- **Candid covariance-free incremental PCA:** fast and accurate.
- **Incremental PCA:** slightly more accurate but slightly slower than CCIPCA.

Conclusions

Topics not discussed today (see paper for details):

- Adaptation to nonstationnary processes: memory/forgetting parameters.
- Imputation of missing data: BLUP.
- Computational speedup: orthogonalize PC matrix every q-th update or so.
- Functional data: work with functional coefficients, not raw data.
- Generalization of vector updates: block updates.

References

- Golub (1973). Some modified matrix eigenvalue problems,
- Hegde et al. (2006). Perturbation-based eigenvector updates for on-line principal components analysis and canonical correlation analysis.
- Oja (1992). Principal components, minor components, and linear neural networks.
- Sanger (1989). Optimal unsupervised learning in a single-layer linear feedforward neural network.