Principal component analysis as a game with Nash Equilibrium

Elvira Plumite, Nina Konovalova, Ekaterina Radionova, Albina Klepach

{Elvira.Plumite,Nina.Konovalova,Ekaterina.Radionova,Albina.Klepach}@skoltech.ru

[1]Skolkovo Institute of Science and Technology, 2020

Abstract. *There are different ways to solve the problem of principal component analysis (PCA). In this project there was considered PCA as a competitive game in which each approximate eigenvector is controlled by a player whose goal is to maximize their own utility function. The algorithm, combined elements from Oja's rule and Gram-Schmidt orthogonalization, was implemented, checked for a convergence and compared with Oja's algorithm and* `sklearn.PCA` *from python library.*

1. Introduction

The principal components of data (PC) are the vectors that align with the directions of maximum variance. Principal components analysis (PCA) have two main purposes [3]:

1. Gain interpretive features
2. Data compression

The PCA solutions of the d-dimensional dataset $X \in \mathbb{R}^{n\times d}$ are eigenvectors of covariance matrix X^TX or, equivalently, right singular vectors of X.

Such solutions can be found in different ways. First of all, it can be done by computing the full **SVD**. But the cost of computing such **SVD** scales with $O(\min\{nd^2, n^2d\})$ - time and $O(nd)$-space. So for large datasets SVD-computation becomes very costly.

Also different iterative methods can be used, such as **Power method**. But we have some limitations for **Power method**, if there is no big gap between the first and the second element of the spectrum, the convergence will be rather slow. However, each iteration involves multiplication of one or more vectors by the covariance matrix X^TX [4].

2. Oja's algorithm

One of the options is to use Oja's algorithm [5].

Beginning with a random Gaussian matrix $Q_0 \in \mathbb{R}^{d\times k}$ entry i.i.d $\sim \mathbb{N}(0, 1)$, and the following computation repeatedly applies: $Q \leftarrow (I+\nu_t X^TX)Q$, where Q comes from QR-decomposition, $\nu_t > 0$ is some learning rate that may depend on t (in our simulation ν depends on $\frac{c}{\sqrt{t}}$. The most of numerical experiments and theoretical papers show that learning rate depends on parameters determined by the data set and it should be selected in each case individually [2].

So the first problem with Oja's algorithm is that the uncertainty in optimization of hyper-parameters can violate the streaming usage of this algorithm. In most approaches

Algorithm 1 Oja algorithm

Given: matrix $X \in \mathbb{R}^{n \times d}$, initial matrix $\hat{V}_i^0 \in \boldsymbol{S}^{d-1} \times \ldots \boldsymbol{S}^{d-1}$ and adaptive step size ν.
$\hat{V} \leftarrow \hat{V}^0$

for $t = 1 : T$ **do**
$\hat{V} \leftarrow \hat{V} + \nu X^T X \hat{V}$
$Q, R \leftarrow QR(\hat{V}$
$S = sign(sign(diag(R)) + 0.5)$
$\hat{V} = QS$
end for
return $\hat{V}$

we need to do several passes throw the data to estimate the hyper-parameters. The desired idea is to find a stable single-pass algorithm that requires only one single pass throw the data. The other problem with Oja is that this algorithm can not be obviously decentralized so we can only find PCA for centralized data.

However, this algorithm is still one the most used among other algorithms (such as Krasulina method or generalized Hebbian algorithm) for most streaming PCA because of its simplicity and asymptotic convergence.

3. PCA as EigenGame

Another option for PCA is to consider this problem as a game with eigenvectors as players and their objective is to maximize their own local utility function in controlled competition with other vectors.

As was already mentioned above the PCA solutions of the d-dimensional dataset $X \in R^{n \times d}$ are eigenvectors of covariance matrix $X^T X = M$ (right singular vectors of X). That means that the goal is to find a matrix of d orthonormal vectors such as

$$V^T M V = V^T V \Lambda = \Lambda \tag{1}$$

Let $\hat{V}$ - estimate of the true eigenvectors V. Then let's define $R(\hat{V}) = \hat{V}^T M \hat{V}$. Then the PCA problem considered as follow equation:

$$max \sum_i R_{ii} - \sum_{i \neq j} R_{ij}^2, \tag{2}$$

where there is maximizing the trace of R and minimizing off-diagonal elements of R.

However this equation ignores the natural hierarchy of the top-k eigenvectors. For example, $\hat{v}_1$ is penalized for aligning with $\hat{v}_k$ and vice versa, but $\hat{v}_1$, being the estimate of the largest eigenvector, should be free to search for the direction that captures the most variance independent of the locations of the other vectors.[1]

So there will be considered hierarchy structure and the utility functions u_i will be defined the following way:

- The top-1 eigenvector v_1: $u_1 = \langle \hat{v}_1, M\hat{v}_1 \rangle$ and $\hat{v}_1^T \hat{v}_1 = 1$

- The top-2 eigenvector v_2: $u_2 = \langle \hat{v}_2, M\hat{v}_2 \rangle - \frac{\langle \hat{v}_2, M\hat{v}_1 \rangle^2}{\langle \hat{v}_2, M\hat{v}_2 \rangle}$

  ...
- The top-i eigenvector v_i: $u_i = \langle \hat{v}_i, M\hat{v}_i \rangle - \sum_{j<i} \frac{\langle \hat{v}_i, M\hat{v}_j \rangle^2}{\langle \hat{v}_j, M\hat{v}_j \rangle}$

It is important to note that by incorporating knowledge of the natural hierarchy, we are immediately led to constructing asymmetric utilities. So, it allows to formulate PCA problem as a game, but not as a direct optimization problem.

A key concept in games is a **Nash equilibrium**. A Nash equilibrium specifies a variable for each player from which no player can unilaterally deviate and improve their outcome.

Theorem (PCA Solution is the Unique strict-Nash Equilibrium) Assume that the top-k eigenvalues of $X^T X$ are distinct. Then the top-k eigenvectors form the unique strict-Nash equilibrium of the proposed game in Equation $\max \left(u_i = \langle \hat{v}_i, M\hat{v}_i \rangle - \sum_{j<i} \frac{\langle \hat{v}_i, M\hat{v}_j \rangle^2}{\langle \hat{v}_j, M\hat{v}_j \rangle} \right)$.

Solving for the Nash of a game is difficult in general. However, because the game is hierarchical and each player's utility only depends on its parents, it is possible to construct a sequential algorithm that is convergent by solving each player's optimization problem in sequence [1].

4. Algorithm

Each eigenvector can be learned by maximizing its utility:

$$\nabla_{\hat{v}_i} u_i(\hat{v}_i | \hat{v}_{j-i}) = 2M\big[\hat{v}_i - \sum_{j<i} \frac{\hat{v}_i M \hat{v}_j}{\hat{v}_j^T M \hat{v}_j} \hat{v}_j\big] = 2X^T\big[X\hat{v}_i - \sum_{j<i} \frac{\langle X\hat{v}_i, X\hat{v}_j \rangle}{\langle X\hat{v}_j^T, X\hat{v}_j \rangle} X\hat{v}_j\big] \quad (3)$$

Algorithm 2 EigenGame^R - Sequential

Given: matrix $X \in \mathbb{R}^{n \times d}$, maximum error tolerance ρ_i, initial vector $\hat{v}_i^0 \in \boldsymbol{S}^{d-1}$, learned approximate parents $\hat{v}_{j<i}$, and step size α.

$\hat{v}_i \leftarrow \hat{v}_i^0$

$t_i = \left\lceil \frac{5}{4} \min(\|\nabla_{\hat{v}_i^0} u_i\|/2, \rho_i)^{-2} \right\rceil$

for $t = 1 : t_i$ **do**

  rewards $\leftarrow X\hat{v}_i$

  penalties $\leftarrow \sum_{j<i} \frac{\langle X\hat{v}_i, X\hat{v}_j \rangle}{\langle X\hat{v}_j^T, X\hat{v}_j \rangle} X\hat{v}_j$

  $\nabla_{\hat{v}_i} \leftarrow 2X^T$[rewards - penalties]

  $\nabla^R_{\hat{v}_i} \leftarrow \nabla_{\hat{v}_i} - \langle \nabla_{\hat{v}_i}, \hat{v}_i \rangle \hat{v}_i$

  $\hat{v}_i' \leftarrow \hat{v}_i + \alpha \nabla^R_{\hat{v}_i}$

  $\hat{v}_i \leftarrow \frac{v_i'}{\|v_i'\|}$

end for

return $\hat{v}_i$

5. Experiments

We compare EigenGame algorithm against Oja's algorithm. Code with all materials is available on GitHub: `https://github.com/eplumite/nla-skoltech-project`

Having Oja's algorithm implemented we were examining by plotting the convergence of this algorithm for each principal component (other graphs for Oja's implementation can be found in the Supplementary materials section). We observe that in Oja's algorithm are estimated simultaneously, additionally there is a tendency for the first principal components to converge faster than the last ones. However, we can see that the lines for each component are nearly parallel.

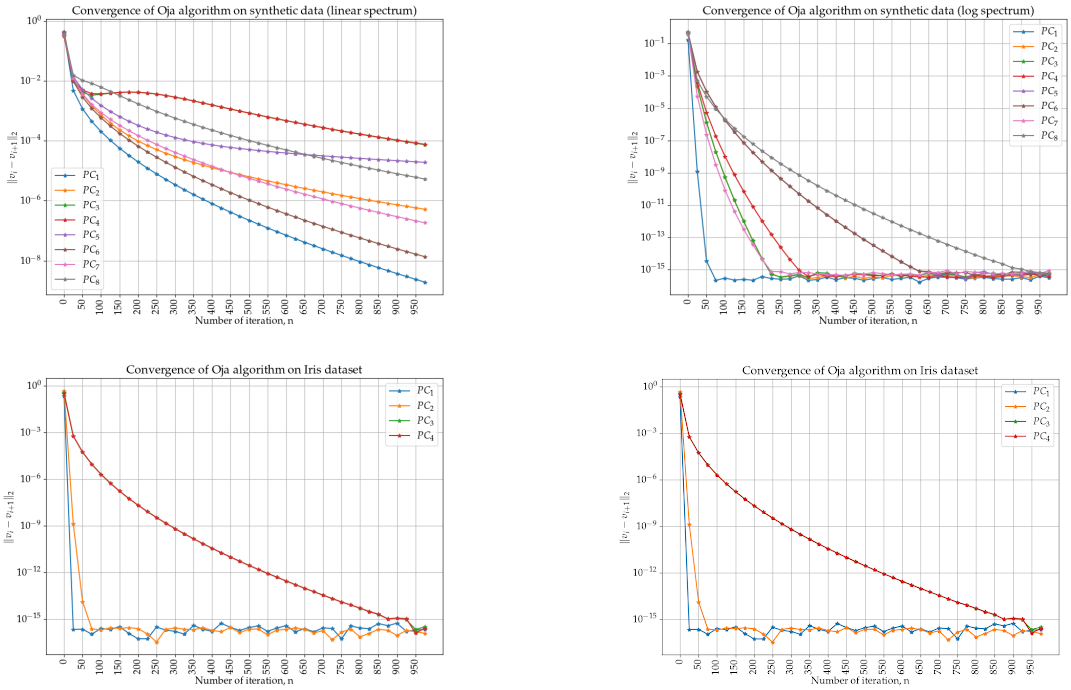


Figure 1. Convergence of Oja's algorithm on different datasets

After implementing EigenGame algorithm we also were examining it by plotting its convergence for each principal component. As you know from the previous section in this algorithm principal components are estimated one-by-one which can be observed on the figures. The fact that was aroused during our studies is that actually Oja's algorithm needs less steps to converge than EigenGame on all the datasets we applied.

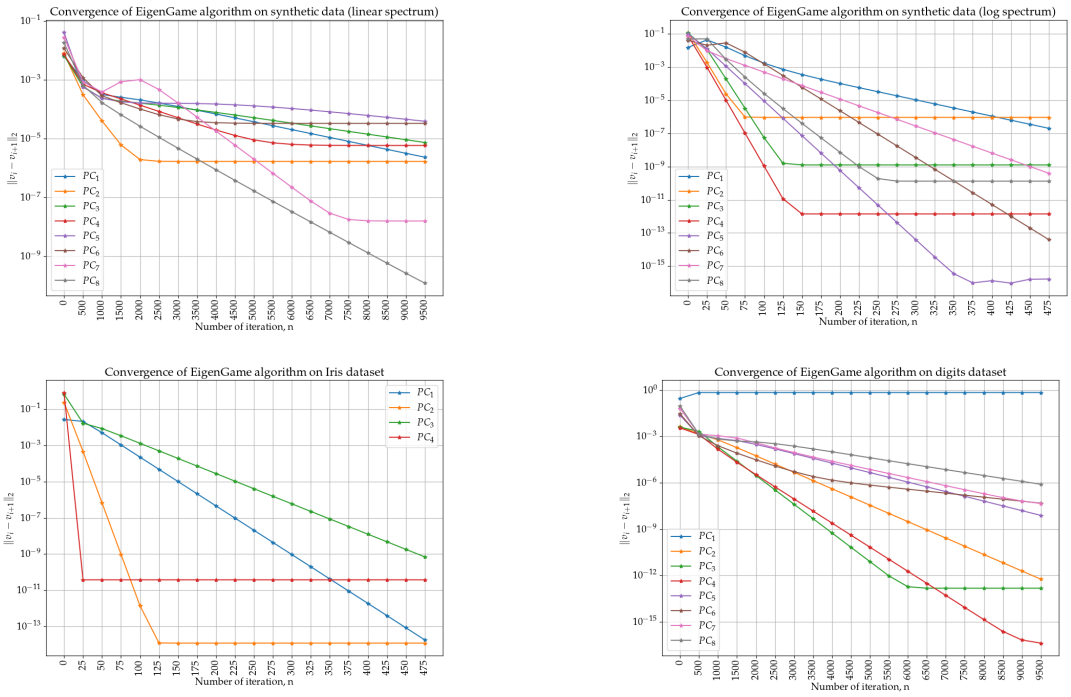


Figure 2. Convergence of EigenGame on different datasets

6. Team members contribution

Stage	Participants
Scientific and literature review of the project	Elvira, Nina, Ekaterina, Albina
Implementation of EigenGame algorithm	Elvira, Ekaterina
Implementation of Oja's algorithm	Nina, Albina
Experiments	Elvira, Nina, Albina
Presentation	Nina, Ekaterina, Albina
Report	Elvira, Nina, Ekaterina, Albina

References

[1] Anonymous. Eigengame: {PCA} as a nash equilibrium. In *Submitted to International Conference on Learning Representations*, 2021. under review.

[2] A. Henriksen and R. Ward. AdaOja: Adaptive Learning Rates for Streaming PCA. page 3, May 2019.

[3] M. K. . N. A. Jake Lever. Principal component analysis. *NATURE METHODS*, 14, 2017.

[4] O. Shamir. A stochastic pca and svd algorithm with an exponential convergence rate.

[5] P. Vlachas. Oja's rule: Derivation, properties. 2019.

7. Supplementary materials

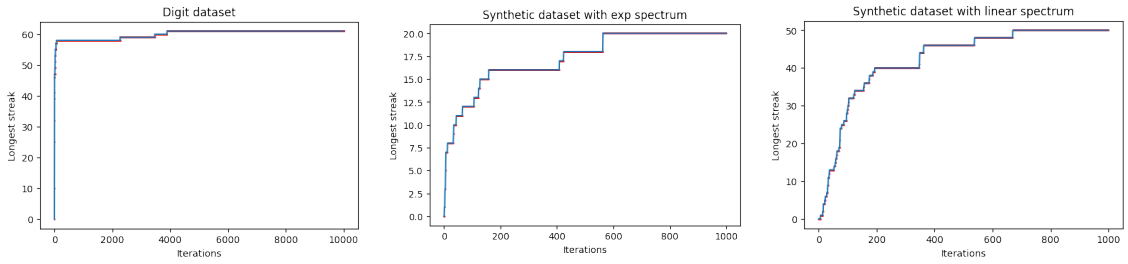
In this section we want to provide some experiments that weren't considered in the main part of our project.

7.1. Addition materials for Oja's algorithm

During implementing Oja's algorithm it was checked in terms of principal component accuracy and subspace distance [1]. This means that principal component accuracy was measured by the number of consecutive components, or longest streak, that were estimated as an angle defined as the angle between $\hat{v}_i$ (computed by Oja's algorithm) and v_i (true vectors in case of synthetic dataset or PCA components computed with sklearn library for digit dataset):

$$\theta_i = \sin^{-1}(\sqrt{1 - \langle v_i, \hat{v}_i \rangle^2} \tag{4}$$

Such angle was compared with $\frac{\pi}{8}$ and if it was less - the number of consecutive components was increased.

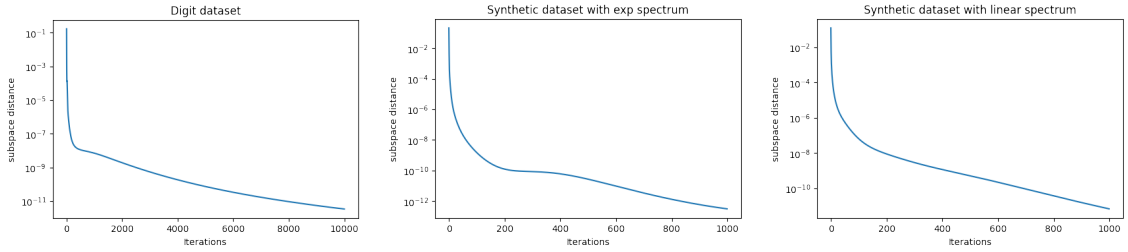


a) b) c)

Figure 3. Longest streak for a) Digit; b) synthetic dataset with exp spectrum c) synthetic dataset with linear spectrum

Normalized subspace distance was found using

$$1 - \frac{1}{k} Tr(U^* P) \in [0, 1], \text{ where } U^* = VV^\dagger \text{ and } P = \hat{V}\hat{V}^\dagger \tag{5}$$



a) b) c)

Figure 4. Subspace distance for a) Digit; b) synthetic dataset with exp spectrum c) synthetic dataset with linear spectrum

7.2. Visualization of algorithms

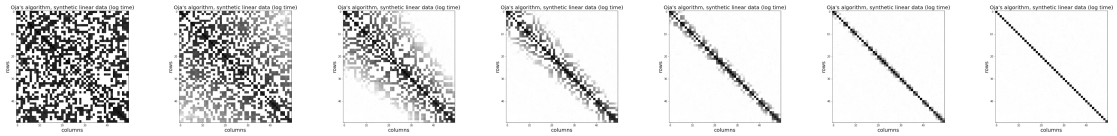


Figure 5. Time steps of Oja's algorithm on synthetic linear data

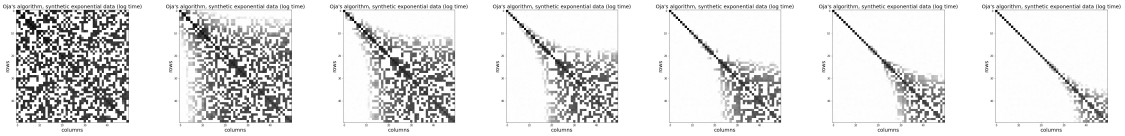


Figure 6. Time steps of Oja's algorithm on synthetic exponential data

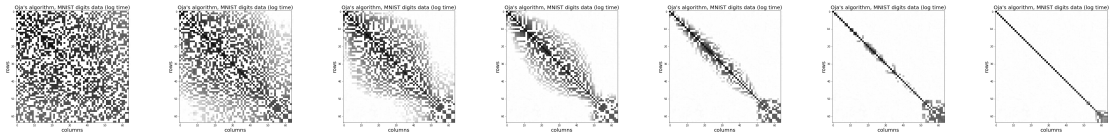


Figure 7. Time steps of Oja's algorithm on digits dataset

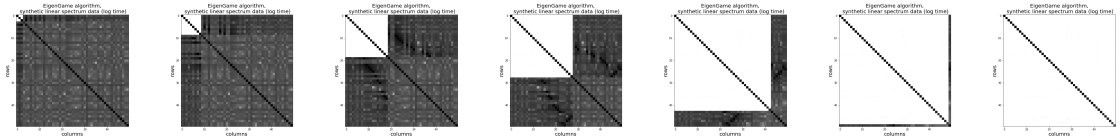


Figure 8. Time steps of EigenGame on synthetic linear data

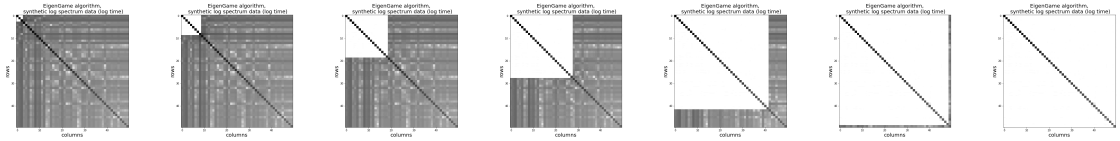


Figure 9. Time steps of EigenGame on synthetic exponential data

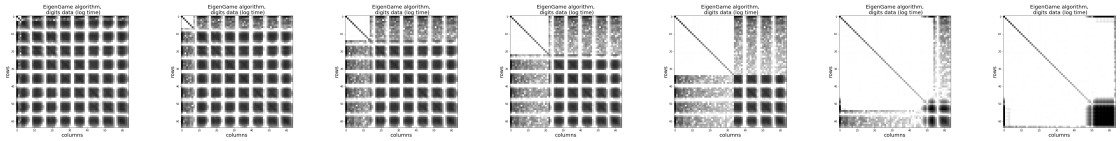


Figure 10. Time steps of EigenGame on digits dataset