THE UNIVERSITY OF TEXAS AT AUSTIN
Department of Statistics and Data Sciences
College of Natural Sciences

SDS 385: Stat Models for Big Data

Lecture 12: PCA and random projections

Purnamrita Sarkar
Department of Statistics and Data Science
The University of Texas at Austin

https://psarkar.github.io/teaching

Principal Component Analysis - why

- Lots of high dimensional data:
  - Documents - Can have about a million words.
  - Recommender systems - If a datapoint is an user, there are around tens of thousands of movies
  - Images – each image is represented using many pixel values – more for higher resolution

Principal Component Analysis - why

- Interpretation/visualization is difficult
- Storage and computation is difficult
- Many features are useless, and may lead to bad generalization error

Principal Component Analysis - why

- Interpretation/visualization is difficult
- Storage and computation is difficult
- Many features are useless, and may lead to bad generalization error
- Solution:
  - Do feature selection
  - Represent data as a linear combination of *important* features

Principal Component Analysis

- Goal: Find the direction of the most variance.
- Say X is the data matrix
- The average is $\bar{\boldsymbol{x}} = \frac{\sum_{i=1}^{n} \boldsymbol{x}_i}{n}$
- Let $\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i - \bar{\boldsymbol{x}}$

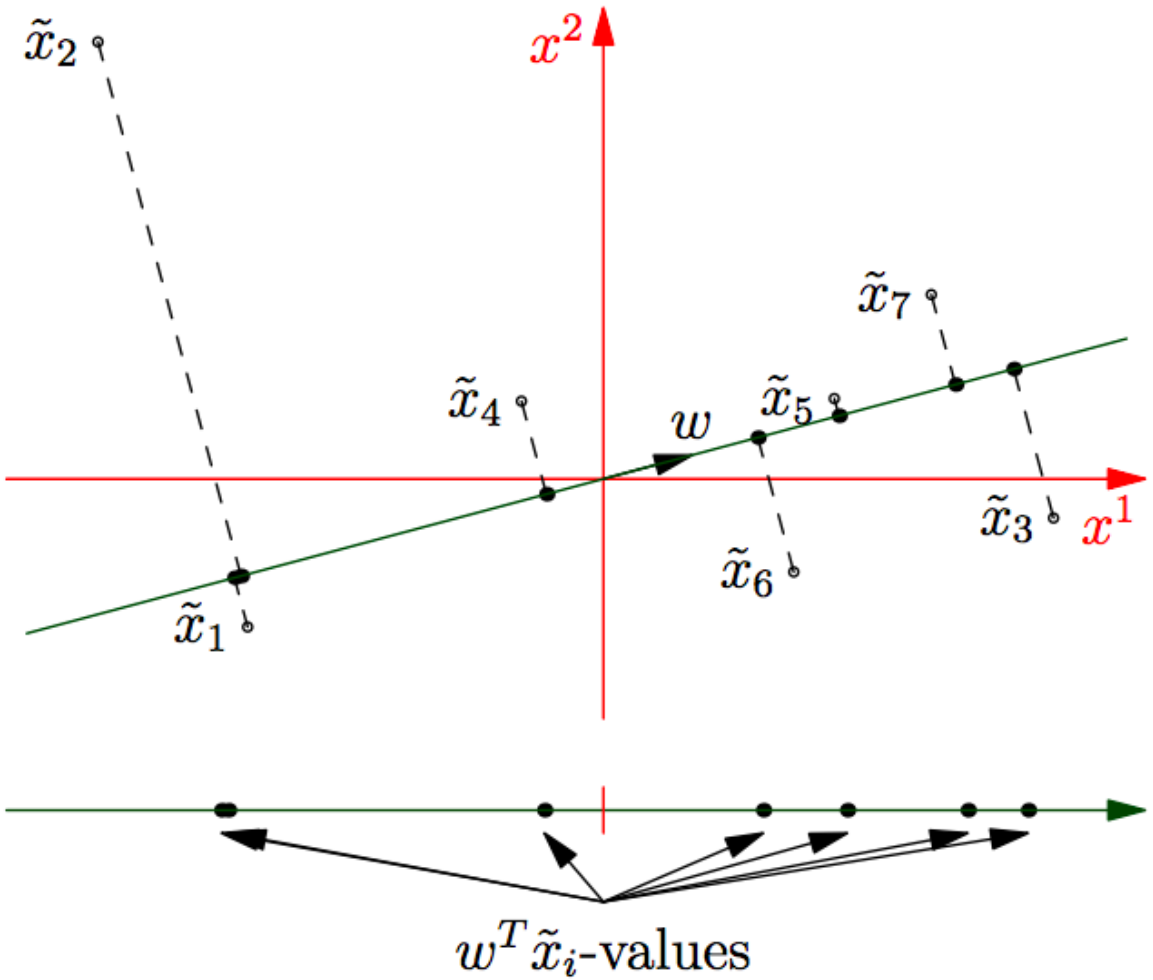
Principal Component Analysis

- Goal: Find the direction of the most variance.
- Say X is the data matrix
- The average is $\bar{\boldsymbol{x}} = \frac{\sum_{i=1}^{n} \boldsymbol{x}_i}{n}$
- Let $\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i - \bar{\boldsymbol{x}}$
- The sample variance of $(\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_n)$ *along a direction* w is give by:

$$\frac{1}{n} \sum_{i=1}^{n} (\tilde{\boldsymbol{x}}_i^T w)^2$$

- What is the sample variance of $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ *along a direction* $\boldsymbol{w}$?

Principal Component Analysis



First component

- So the first PC direction is:

$$\boldsymbol{w}_1 = \arg \max_{\|\boldsymbol{w}\|=1} \frac{1}{n} \sum_{i=1}^{n} (\tilde{\boldsymbol{x}}_i^T \boldsymbol{w})^2$$

- And the first PC component of $\tilde{\boldsymbol{x}}_i$ is $\tilde{\boldsymbol{x}}_i^T \boldsymbol{w}_1$

First component

- So the k^{th} PC direction is:

$$\boldsymbol{w}_k = \arg \max_{\substack{\|\boldsymbol{w}\|=1 \\ \boldsymbol{w} \perp \boldsymbol{w}_1, \ldots, \boldsymbol{w}_{k-1}}} \frac{1}{n} \sum_{i=1}^{n} (\tilde{\boldsymbol{x}}_i^T \boldsymbol{w})^2$$

- And the k^{th} PC component of $\tilde{\boldsymbol{x}}_i$ is $\tilde{\boldsymbol{x}}_i^T \boldsymbol{w}_k$
- Note that $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k$ form an orthogonal basis.

Simple algorithm

- Let W is a matrix with $\boldsymbol{w}_k$ along its columns
- $\tilde{X}W$ gives a **low dimensional** representation of $\tilde{X}$

Simple algorithm

- Let W is a matrix with $\boldsymbol{w}_k$ along its columns
- $\tilde{X}W$ gives a **low dimensional** representation of $\tilde{X}$
- We can frame the optimization problem also as

$$\boldsymbol{w}_1 = \arg \max_{\|\boldsymbol{w}\|=1} \boldsymbol{w}^T \tilde{X}^T \tilde{X} \boldsymbol{w}$$

Simple algorithm

- Let W is a matrix with $\boldsymbol{w}_k$ along its columns
- $\tilde{X}W$ gives a **low dimensional** representation of $\tilde{X}$
- We can frame the optimization problem also as

$$\boldsymbol{w}_1 = \arg \max_{\|\boldsymbol{w}\|=1} \boldsymbol{w}^T \tilde{X}^T \tilde{X} \boldsymbol{w}$$

- This is the first eigenvector of $S = \tilde{X}^T \tilde{X}$

Eigenvector and eigenvalues

- Any square symmetrix matrix S has real eigenvalues
- The i^{th} eigenvalue,vector pair satisfy $S\boldsymbol{w}_i = \lambda_i \boldsymbol{w}_i$
- The eigenvectors are orthogonal to each other, and normalized to have length 1.

Eigenvector and eigenvalues

- Any square symmetrix matrix S has real eigenvalues
- The i^{th} eigenvalue,vector pair satisfy $S\boldsymbol{w}_i = \lambda_i \boldsymbol{w}_i$
- The eigenvectors are orthogonal to each other, and normalized to have length 1.
- In matrix terms, we can write:

$$S = U\Sigma U^T, \text{ where}$$

  - columns of U are the orgonal eigenvectors, and
  - Σ is a diagonal matrix with eigenvalues on the diagonal
- The larger the magnitude of the eigenvalue, more important the eigenvector

Back to PCA: Simple algorithm

- Let W is a matrix with $\boldsymbol{w}_k$ along its columns
- $\tilde{X}W$ gives a **low dimensional** representation of $\tilde{X}$

Back to PCA: Simple algorithm

- Let W is a matrix with $\boldsymbol{w}_k$ along its columns
- $\tilde{X}W$ gives a **low dimensional** representation of $\tilde{X}$
- We can frame the optimization problem also as

$$\boldsymbol{w}_1 = \arg \max_{\|\boldsymbol{w}\|=1} \boldsymbol{w}^T \tilde{X}^T \tilde{X} \boldsymbol{w}$$

Back to PCA: Simple algorithm

- Let W is a matrix with $\boldsymbol{w}_k$ along its columns
- $\tilde{X}W$ gives a **low dimensional** representation of $\tilde{X}$
- We can frame the optimization problem also as

$$\boldsymbol{w}_1 = \arg \max_{\|\boldsymbol{w}\|=1} \boldsymbol{w}^T \tilde{X}^T \tilde{X} \boldsymbol{w}$$

- This is the first eigenvector of $S = \tilde{X}^T \tilde{X}$

Back to PCA: Simple algorithm

- Let W is a matrix with $\boldsymbol{w}_k$ along its columns
- $\tilde{X}W$ gives a **low dimensional** representation of $\tilde{X}$
- We can frame the optimization problem also as

$$\boldsymbol{w}_1 = \arg \max_{\|\boldsymbol{w}\|=1} \boldsymbol{w}^T \tilde{X}^T \tilde{X} \boldsymbol{w}$$

- This is the first eigenvector of $S = \tilde{X}^T \tilde{X}$
- What is S?

Back to PCA: Simple algorithm

- Let W is a matrix with $\boldsymbol{w}_k$ along its columns
- $\tilde{X}W$ gives a **low dimensional** representation of $\tilde{X}$
- We can frame the optimization problem also as

$$\boldsymbol{w}_1 = \arg \max_{\|\boldsymbol{w}\|=1} \boldsymbol{w}^T \tilde{X}^T \tilde{X} \boldsymbol{w}$$

- This is the first eigenvector of $S = \tilde{X}^T \tilde{X}$
- What is S?
- Its the scalar multiple of the sample covariance matrix

$$\hat{\Sigma} = \frac{1}{n} \sum_i \tilde{\boldsymbol{x}}_i \tilde{\boldsymbol{x}}_i^T = \frac{S}{n}$$

Back to PCA: Simple algorithm

- Let W is a matrix with $\boldsymbol{w}_k$ along its columns
- $\tilde{X}W$ gives a **low dimensional** representation of $\tilde{X}$
- We can frame the optimization problem also as

$$\boldsymbol{w}_1 = \arg \max_{\|\boldsymbol{w}\|=1} \boldsymbol{w}^T \tilde{X}^T \tilde{X} \boldsymbol{w}$$

- This is the first eigenvector of $S = \tilde{X}^T \tilde{X}$
- What is S?
- Its the scalar multiple of the sample covariance matrix

$$\hat{\Sigma} = \frac{1}{n} \sum_i \tilde{\boldsymbol{x}}_i \tilde{\boldsymbol{x}}_i^T = \frac{S}{n}$$

- So, all you have to do is to calculate eigenvectors of the covariance matrix.

Back to PCA: Simple algorithm

- So, all you have to do is to calculate eigenvectors of the covariance matrix.
- But, do I even need to do that?

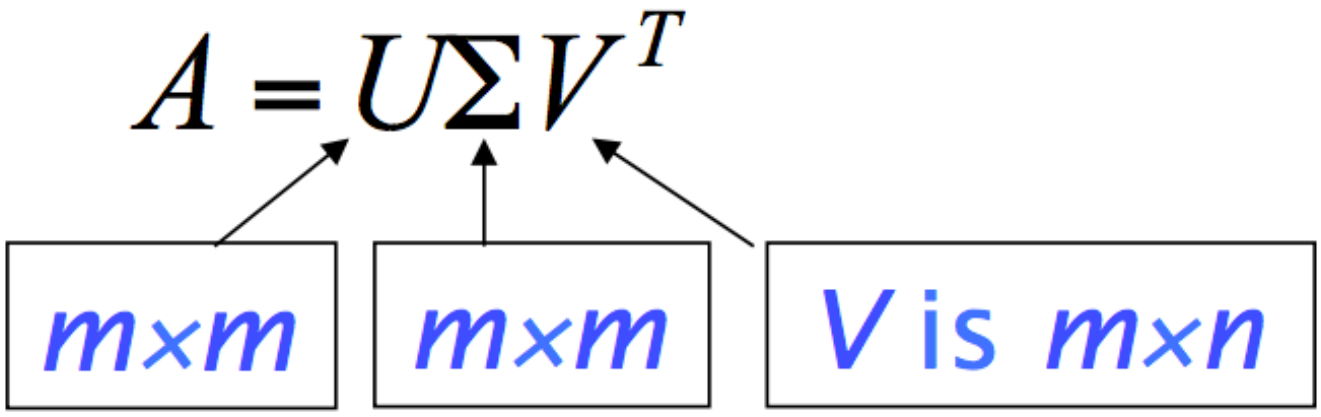
Back to PCA: Simple algorithm

- So, all you have to do is to calculate eigenvectors of the covariance matrix.
- But, do I even need to do that?
- The right singular vectors of $\tilde{X}$ is just fine.

Back to PCA: Simple algorithm

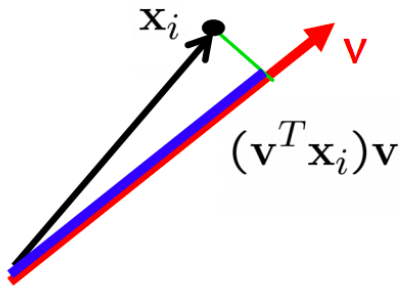
- So, all you have to do is to calculate eigenvectors of the covariance matrix.
- But, do I even need to do that?
- The right singular vectors of $\tilde{X}$ is just fine.
- How many PC's? (more of a dissertaiton question)

Singular value decomposition



- The columns of U are orthogonal eigenvectos of AA^T
- The columns of V are orthogonal eigenvectos of $A^T A$
- $A^T A$ and AA^T have the same eigenvalues

Second interpretation



- Minimum reconstruction error:

$$(\boldsymbol{x}_i - (\boldsymbol{x}_i^T \boldsymbol{w})\boldsymbol{w})^T(\boldsymbol{x}_i - (\boldsymbol{x}_i^T \boldsymbol{w})\boldsymbol{w}) = \boldsymbol{x}_i^T \boldsymbol{x}_i - (\boldsymbol{x}_i^T \boldsymbol{w})^2$$

- So, the first PC direction gives the direction projecting on which has the **minimum reconstruction error**.

Low rank approximation

- Take the centered data matrix $\tilde{X}$ with SVD

$$\tilde{X} = USV^T$$

- Project on the top k PC's $W \in \mathbb{R}^{p \times k}$

Low rank approximation

- Take the centered data matrix $\tilde{X}$ with SVD

$$\tilde{X} = USV^T$$

- Project on the top k PC's $W \in \mathbb{R}^{p \times k}$
- You get $\tilde{X}W = US_kV^T$, where S_k has zeroed out all singular values $\leq \sigma_k$

Low rank approximation

- Take the centered data matrix $\tilde{X}$ with SVD

$$\tilde{X} = USV^T$$

- Project on the top k PC's $W \in \mathbb{R}^{p \times k}$
- You get $\tilde{X}W = US_kV^T$, where S_k has zeroed out all singular values $\leq \sigma_k$
- So $W = \arg \min_{\text{rank}(B)=k, B \in \mathbb{R}^{n \times p}} \|\tilde{X} - B\|_F^2$ and the reconstruction error is $\sum_{i=k+1}^{p} \sigma_i^2$

Low rank approximation

- Take the centered data matrix $\tilde{X}$ with SVD

$$\tilde{X} = USV^T$$

- Project on the top k PC's $W \in \mathbb{R}^{p \times k}$
- You get $\tilde{X}W = US_kV^T$, where S_k has zeroed out all singular values $\leq \sigma_k$
- So $W = \arg \min_{\text{rank}(B)=k, B \in \mathbb{R}^{n \times p}} \|\tilde{X} - B\|_F^2$ and the reconstruction error is $\sum_{i=k+1}^{p} \sigma_i^2$
- This explains why you want to take large k to reduce approx. error.

Lets do some coding

- We will make a covariance matrix and generate independent multivariate gaussian random variables

```
d=1000
Sigma=np.zeros([d,d])
Sigma[0:200,0:200]=.3;
np.fill_diagonal(Sigma,1)
```

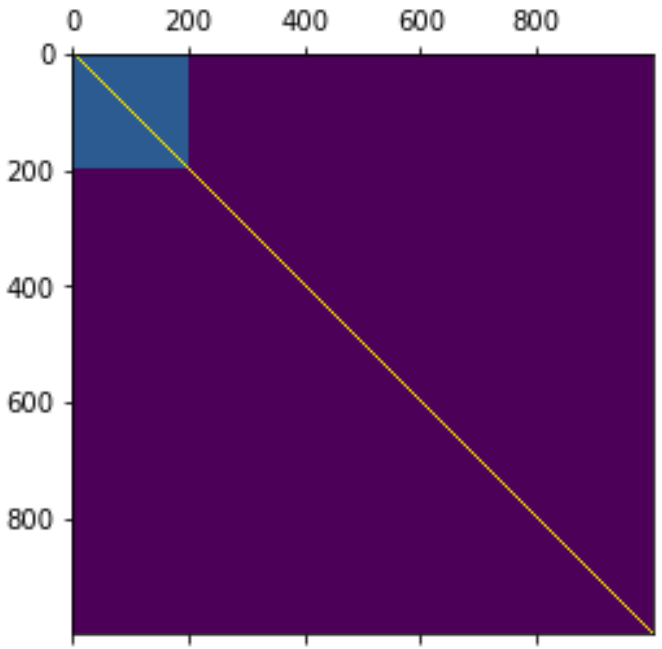
Lets do some coding

- We will make a covariance matrix and generate independent multivariate gaussian random variables

```
d=1000
Sigma=np.zeros([d,d])
Sigma[0:200,0:200]=.3;
np.fill_diagonal(Sigma,1)
```

```
plt.matshow(Sigma)
```

```
<matplotlib.image.AxesImage at 0x188c373d0>
```



Lets do some coding

- Now lets compute eigenvectors of the covariance matrix.

```
X=np.random.multivariate_normal(np.zeros(d),Sigma,5000)
```

```
S=np.cov(np.transpose(X))
u,s,vt=svd(S)
```

Lets do some coding

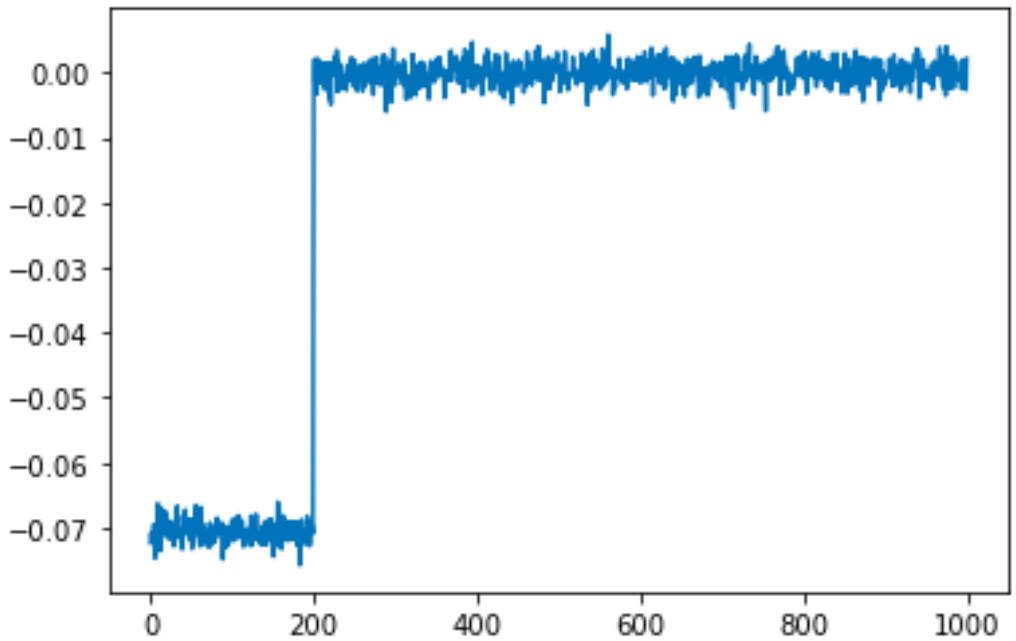
- Now lets compute eigenvectors of the covariance matrix.

```
X=np.random.multivariate_normal(np.zeros(d),Sigma,5000)
```

```
S=np.cov(np.transpose(X))
u,s,vt=svd(S)
```

```
plot(u[:,0])
```

```
[<matplotlib.lines.Line2D at 0x167e5ae50>]
```



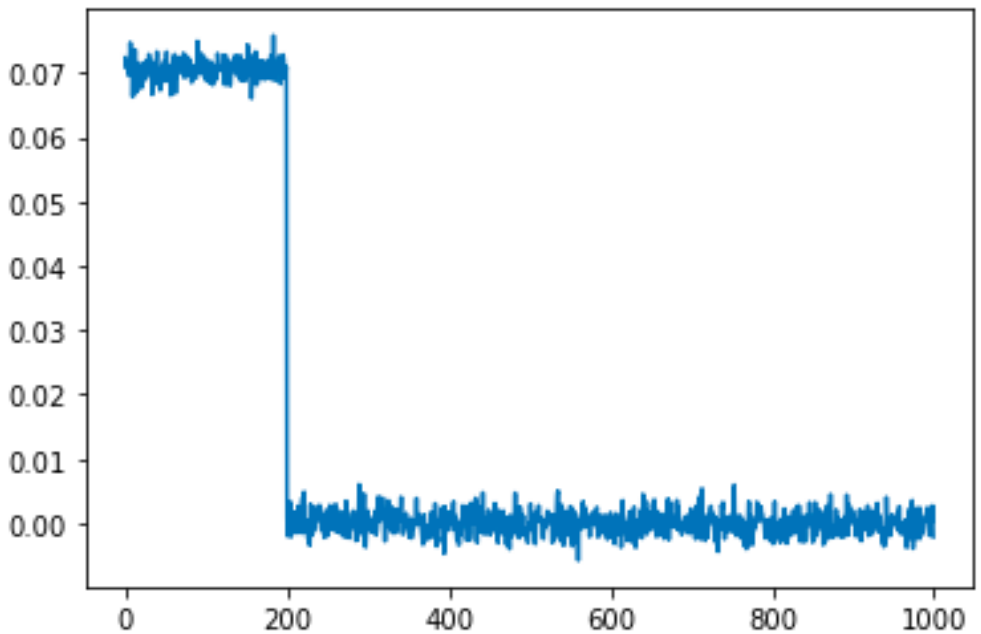
Lets do some coding

- Now lets do SVD on the data matrix X.

```
u,s,vt=svd(X)
```

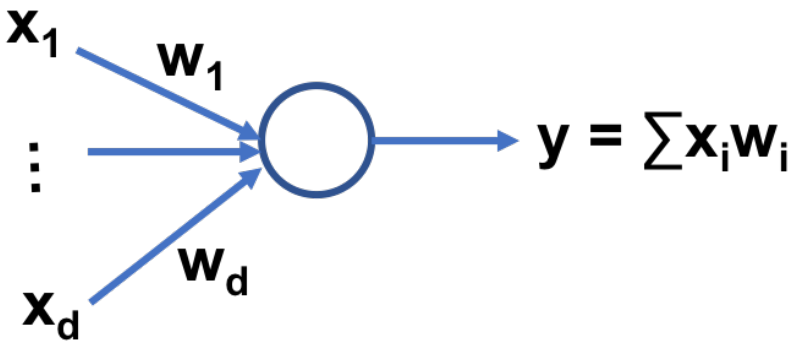
```
plot(vt[0,:])
```

```
[<matplotlib.lines.Line2D at 0x1680810a0>]
```



Online PCA - Oja's algorithm

- Erikki Oja wrote a seminal paper in 1982 about a simple neural network model.
- He was inspired by the Hebbian principle (1949, "The organization of behavior", Donald Hebb) which claims that the synaptic energy increases from presynaptic cells stimulating post-synaptic cells.



Online PCA - Oja's algorithm

- For each data-point, you do:

$$w_{t+1} \leftarrow w_t + \eta_t (x_t^T w_t) x_t$$

$$w_{t+1} \leftarrow w_{t+1} / \|w_{t+1}\|$$

Online PCA - Oja's algorithm

- For each data-point, you do:

$$w_{t+1} \leftarrow w_t + \eta_t (x_t^T w_t) x_t$$

$$w_{t+1} \leftarrow w_{t+1} / \|w_{t+1}\|$$

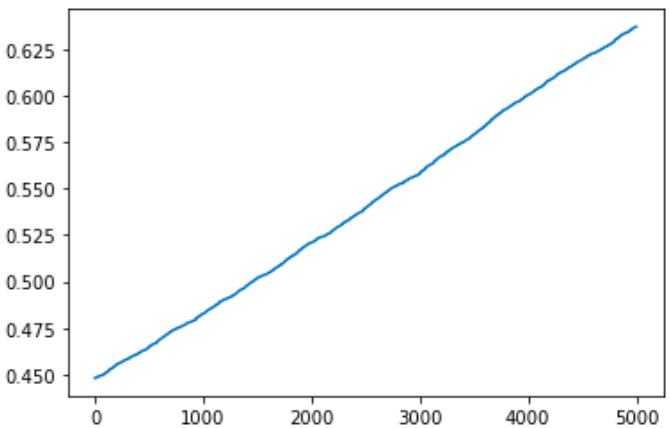
- Note that here, you do not need to construct the covariance matrix explicitly, which is extremely useful, when d is much larger than n, i.e. in high dimensional settings.
- Step size η_t can be set as $c \log n/n$ or $\eta_t \propto 1/t$
- Sharp error bounds show that the final solution converges to the principal component and the error has weak dependence on dimensionality d

Lets do some coding

- Now lets do Oja's algorithm for X.
- Set $\eta = 0.001 \log n/n$

```
plot(dot_prod)
```

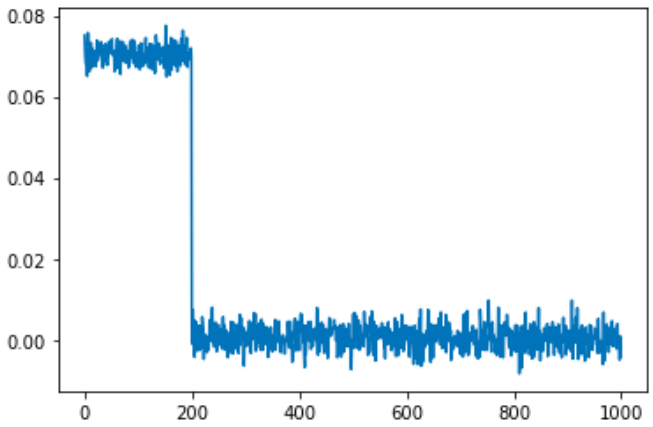
```
[<matplotlib.lines.Line2D at 0x165092100>]
```



Dot product with truth

```
plot(z)
```

```
[<matplotlib.lines.Line2D at 0x1679e1100>]
```



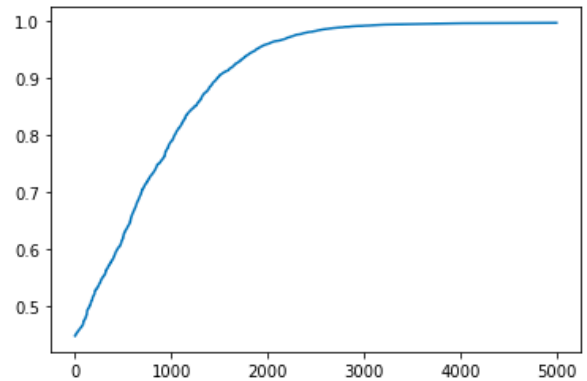
final vector

Lets do some coding

- Now lets do Oja's algorithm for X.
- Set $\eta = 0.01 \log n/n$

```
plot(dot_prod)
```

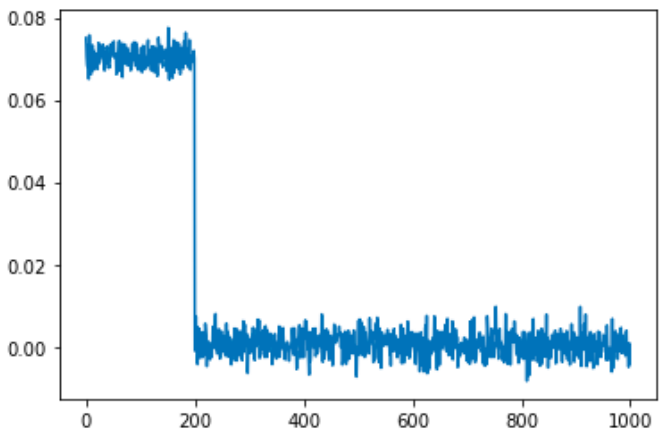
```
[<matplotlib.lines.Line2D at 0x168107130>]
```



Dot product with truth

```
plot(z)
```

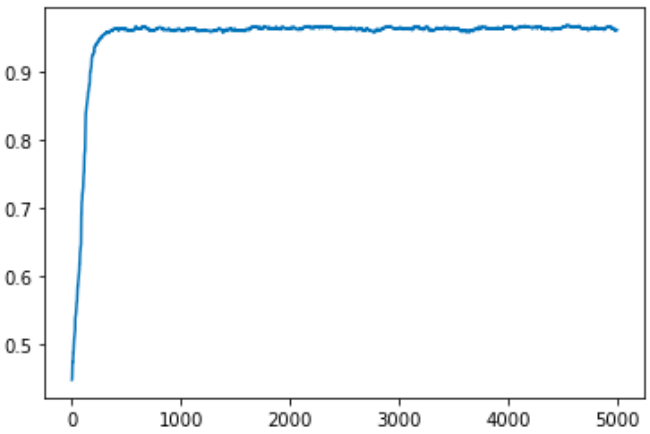
```
[<matplotlib.lines.Line2D at 0x1679e1100>]
```



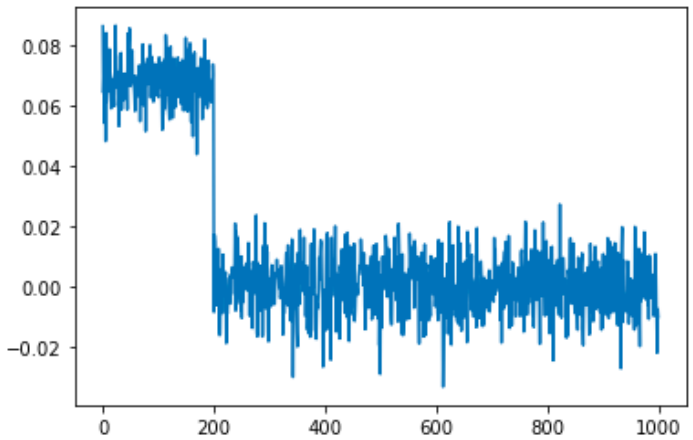
final vector

Lets do some coding

- Now lets do Oja's algorithm for X.
- Set $\eta = 0.1 \log n/n$



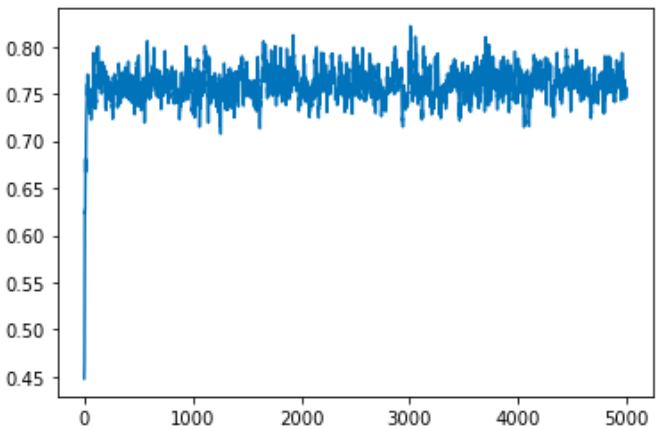
Dot product with truth



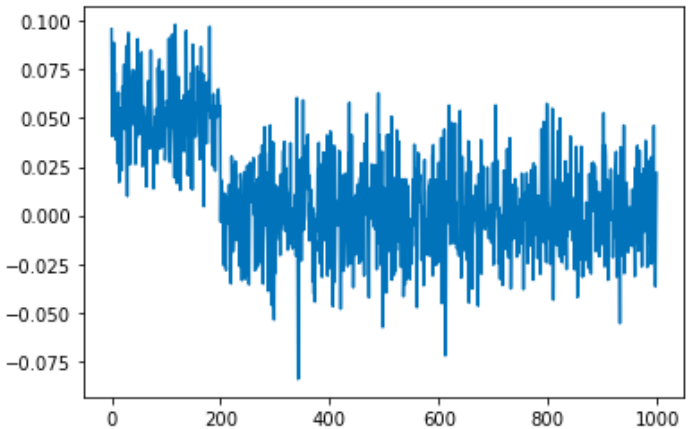
final vector

Lets do some coding

- Now lets do Oja's algorithm for X.
- Set $\eta = 1 \log n/n$



Dot product with truth



final vector

Online PCA - Oja's algorithm - (thanks to Yijun Dong)

- Oja's algorithm maximizes $w^T \text{cov}(X) w$ over $\|w\| = 1$
- Why is it nonconvex?
  - First, the constraint is non-convex.
  - But you can change the optimization to do $\max_{\|w\| \leq 1} w^T \text{cov}(X) w$ and $\text{cov}(X)$ is PSD.
  - Sure, but even then, you are maximizing a convex function here, not minimizing it.

Random projections

- What if we did something crazy and projected the data on a random vector.

$$\tilde{x}_i = x_i^T R$$

Random projections

- What if we did something crazy and projected the data on a random vector.

$$\tilde{x}_i = x_i^T R$$

  - R could be Gaussian random variables:

$$R_i \sim N(0, 1)$$

  - R could be centered bernoullis:

$$R_i \sim \begin{cases} 1 & \text{with prob } 1/2 \\ -1 & \text{with prob } 1/2 \end{cases}$$

  - R could be sparse

$$R_i \sim \begin{cases} 1/\sqrt{s} & \text{with prob } 1/2s \\ 0 & \text{with prob } 1 - 1/s \\ -1/\sqrt{s} & \text{with prob } 1/2s \end{cases}$$

Random projections

- Why will this work?
- Lets take a unit vector u and see if RP preserves the norm.
- Take d dimensional standard normal vector R

Random projections

- Why will this work?
- Lets take a unit vector u and see if RP preserves the norm.
- Take d dimensional standard normal vector R
- $Y = u^T R = \sum_i u_i R_i \sim N(0, 1)$

Random projections

- Why will this work?
- Lets take a unit vector u and see if RP preserves the norm.
- Take d dimensional standard normal vector R
- $Y = u^T R = \sum_i u_i R_i \sim N(0, 1)$
- $Y^2 \sim \chi^2$, $EY^2 = 1$

Random projections

- Why will this work?
- Lets take a unit vector u and see if RP preserves the norm.
- Take d dimensional standard normal vector R
- $Y = u^T R = \sum_i u_i R_i \sim N(0, 1)$
- $Y^2 \sim \chi^2$, $EY^2 = 1$
- On an average the length is preserved

Random projections

- Why will this work?
- Lets take a unit vector u and see if RP preserves the norm.
- Take d dimensional standard normal vector R
- $Y = u^T R = \sum_i u_i R_i \sim N(0, 1)$
- $Y^2 \sim \chi^2$, $EY^2 = 1$
- On an average the length is preserved
- But how about the variance?

Random projections

- Now take $R \in N(0,1)^{d \times m}/\sqrt{m}$

Random projections

- Now take $R \in N(0,1)^{d \times m}/\sqrt{m}$
- $u^T R = [u^T R(:,1), \ldots u^T R(:,m)]/\sqrt{m}$

Random projections

- Now take $R \in N(0,1)^{d \times m}/\sqrt{m}$
- $u^T R = [u^T R(:,1), \ldots u^T R(:,m)]/\sqrt{m}$

$$\|u^T R\|^2 = \sum_i (u^T R(:,i))^2/m$$

Average of m chi-squared RVs

concentrates around 1

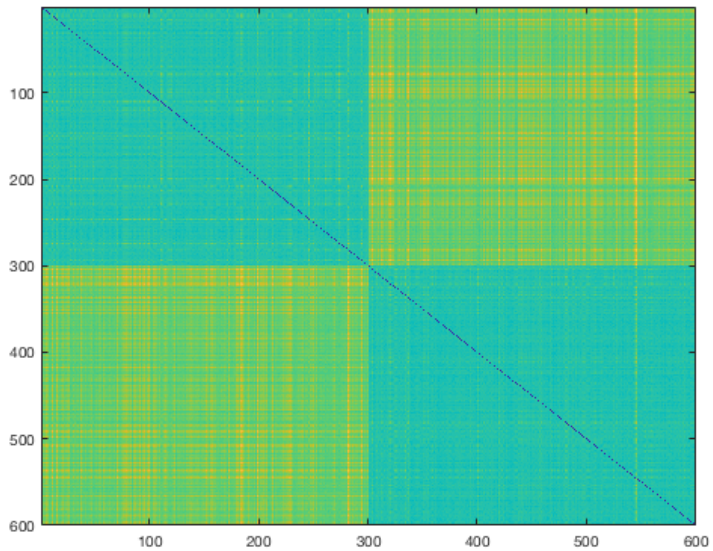
Johnson-Lindenstrauss lemma

- Take $u = \frac{X_i - X_j}{\|X_i - X_j\|}$, its norm will be preserved under a random projection, with high prob.
- We are saying, with high probability,

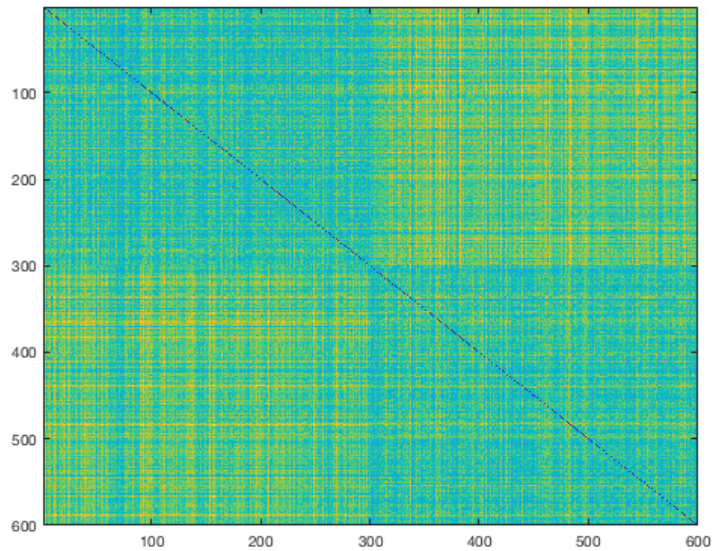
$$(1-\epsilon)\|X_i - X_j\|_2^2 \leq \|X_i^T R - X_j^T R\|_2^2 \leq (1+\epsilon)\|X_i - X_j\|_2^2$$

- If you pick $m = \log n/\epsilon^2$, then this will be satisfied for all pairs, with high probability.

- Draw 1000 dimensional Gaussians, 300 from origin and S as cov, 300 from all ones mean vector.

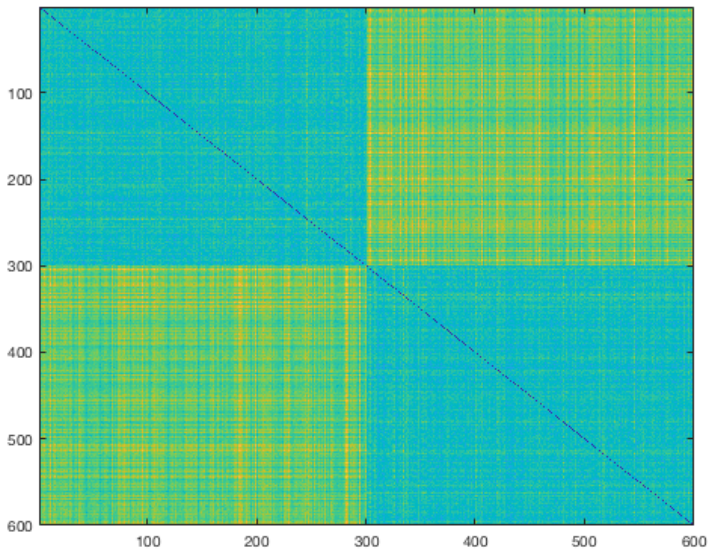


Pairwise distances

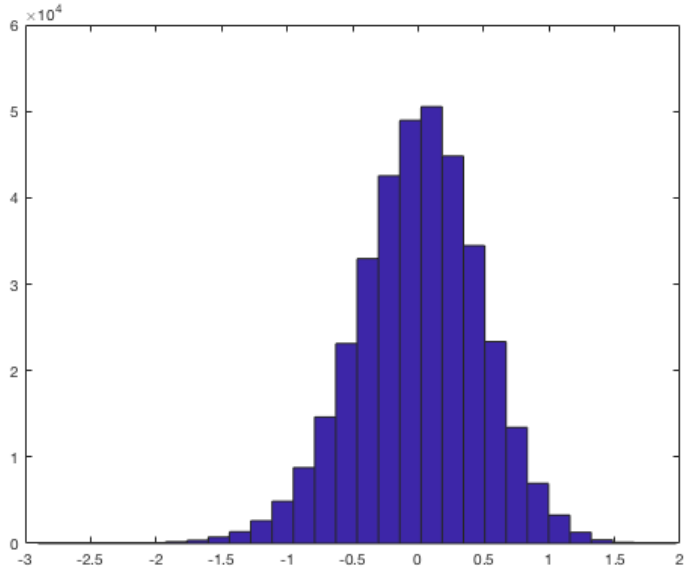


Pairwise distances with $k = 20$

- Draw 1000 dimensional Gaussians, 300 from origin and S as cov, 300 from all ones mean vector.



Pairwise distances with $k = 70$



Histogram of error

Acknowledgment

- Some pictures are borrowed from Brett Bernstein's notes from NYU.