Data Mining: W4240
Dimensionality Reduction I

Lecture 23

Prof. Lauren Hannah
Columbia University
Department of Statistics

November 26, 2013

Outline

1. High dimensional data
2. Principal components analysis (PCA) overview
3. (Review: eigenvalues and eigenvectors)
4. PCA computation

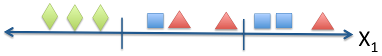
Next time: more PCA, doing PCA with R

The Curse of Dimensionality

The Curse of Dimensionality: "There are multiple phenomena referred to by this name in domains such as numerical analysis, sampling, combinatorics, machine learning, data mining and databases. The common theme of these problems is that when the dimensionality increases, the volume of the space increases so fast that the available data becomes sparse. This sparsity is problematic for any method that requires statistical significance. In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows exponentially with the dimensionality." (Wikipedia)

The Curse of Dimensionality

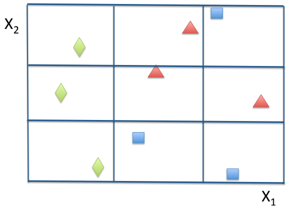
Example: predict the class given covariates

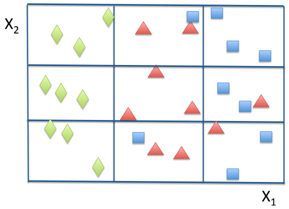


Pretty easy when data is in 1 dimension (9 samples gives us a good enough predictor)

The Curse of Dimensionality

Example: predict the class given covariates; same data in 2 dimensions

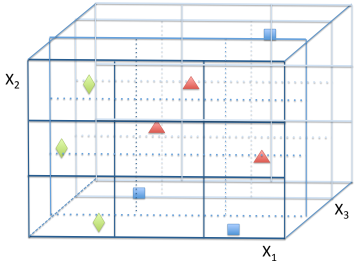




In 2 dimensions, we are much less sure about our predictions (9 for 9 regions...) or we need more data ($3 \times 3^2 = 27$ observations for same sureity)

The Curse of Dimensionality

Example: predict the class given covariates; original data in 3 dimensions



In 3 dimensions, we only have data in 1/3 of our regions (9 in 27 regions...). For same predictive power, we need $3 \times 3^3 = 81$

The Curse of Dimensionality

Implications:

- Data required for prediction grows like n^d (prediction methods fail)
- Describing and storing the data becomes increasingly difficult (summarizing data hard)

Goals:

- Summarize data compactly
- Reduce dimensions as preprocessing for other methods

How do we beat the curse?

Higher Dimensional Data

Problem: want to describe D&D characters

Attributes:

- Strength
- Dexterity
- Constitution
- Intelligence
- Wisdom
- Charisma

Each can take values in $\{1, 2, \ldots, 20\}$. Do I really need 64 million (20^6) different values to describe characters well?

Higher Dimensional Data

Possible character A:

- Strength = 18
- Dexterity = 14
- Constitution =16
- Intelligence = 10
- Wisdom = 11
- Charisma = 9

Possible character B:

- Strength = 18
- Dexterity = 18
- Constitution = 6
- Intelligence = 4
- Wisdom = 19
- Charisma = 9

Would you expect to see both of these characters?

Higher Dimensional Data

Can describe set of possible characters pretty well:

- mage
- fighter
- rogue
- cleric
- specialized subtypes: rangers, monks, bards, etc

64 million possible builds probably closer to 10,000... and can even summarize more efficiently with a small set of classes (clusters)

Clustering can be used to efficiently summarize large data sets, but it does not help for preprocessing data for statistical methods

Higher Dimensional Data

Problem: want to describe student performance and compare students

Data:

- scores on five homeworks
- midterm score
- final score

How can I summarize data for a student in a way that will be useful for comparisons?

Higher Dimensional Data

If data *truly* high dimensional:

- scores for each of 7 items have low correlation
- need all items to summarize student (good on HWs 1,2,4, middling on HW 3 and midterm, abysmal on HW 5, low on final)
- no good way to compare students

But we would not expect to see a student like the one above. Why?

Any suggestions for summarizing student performance in a way that is useful for comparing students? Why would this be reasonable?

Higher Dimensional Data

I have some (randomized) data. Let's look at it in R.

Does it agree with your prediction about summary?

This approach is called **dimensionality reduction**. We will be studying this for the next two classes.

Dimensionality Reduction

Data with d dimensions:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ \vdots \\ x_d \end{bmatrix}$$

x_j has many names: *covariate*, *feature*, *input*, or *independent variable*.

Assumptions for today:

- all features are continuous-valued ($\mathbf{x} \in \mathbb{R}^d$)
- d is large (10 or 100 or more)

Dimensionality Reduction

Data with n observations and d dimensions:

$$\mathbf{X} = \begin{bmatrix} x_{11} & \dots & x_{1d} \\ x_{21} & \dots & x_{2d} \\ \vdots & & \vdots \\ x_{n1} & \dots & x_{nd} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix}$$

Here $\mathbf{x}^T$ means the transpose of $\mathbf{x}$.

Today, **bold** means matrix or vector, and *plain* means scalar.

Dimensionality Reduction

Two approaches:

- **Feature Selection:** choose a subset of features for prediction task

$$[x_1, x_2, \ldots, x_d] \to [x_{i1}, x_{i2}, \ldots, x_{iK}]$$

  But you need to know your prediction task before you do feature selectionn.

- **Feature Extraction:** create new features by combining existing ones

$$\begin{aligned}[x_1, x_2, \ldots, x_d] \to [f_1(x_{11}, x_{12}, \ldots, x_{1d}), \ldots, f_K(x_{K1}, x_{K2}, \ldots, x_{Kd})]\\ = [y_1, \ldots, y_K]\end{aligned}$$

Feature extraction is often a data preprocessing step since you do not need to know which prediction methods you will use.

Dimensionality Reduction

Today, we will look at *linear feature extraction*

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ \vdots \\ x_d \end{bmatrix} \rightarrow \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \dots & \dots & \dots & w_{1d} \\ w_{21} & w_{22} & \dots & \dots & \dots & w_{2d} \\ \vdots & & & & & \vdots \\ w_{K1} & w_{K2} & \dots & \dots & \dots & w_{Kd} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ \vdots \\ x_d \end{bmatrix}$$

$$\mathbf{Y} = \mathbf{X}\mathbf{W}^T$$

Terminology: $y_{i1}, \dots, y_{iK}$ are the *scores*, $\mathbf{w}_1, \dots, \mathbf{w}_K$ are the *loadings*.

Linear Feature Extraction

Student score averages are a form of linear feature extraction:

$$y = [0.067,\ 0.067,\ 0.133,\ 0.067,\ 0.067,\ 0.25,\ 0.35] \begin{bmatrix} HW1 \\ HW2 \\ HW3 \\ HW4 \\ HW5 \\ Midterm \\ Final \end{bmatrix}$$

The student score average is the *score* and the assignment weights are the *loadings*

Linear Feature Extraction

How can we find a good set of loadings and scores for a general data set?

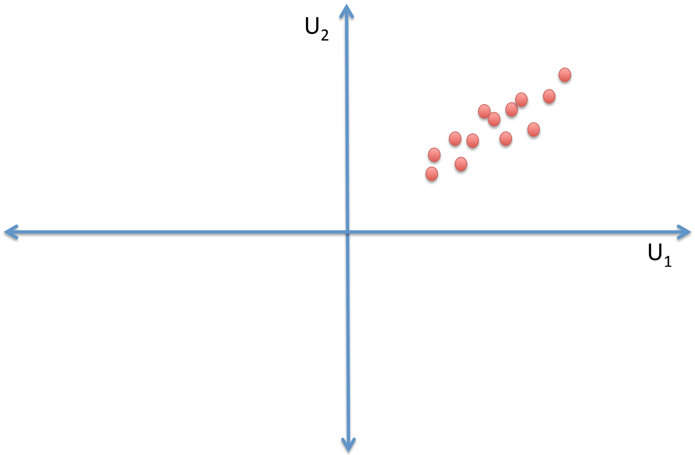
Principal components analysis (PCA): a covariance matrix singular value decomposition method for unsupervised data

Principal components: a set of linearly uncorrelated variables—these will be the loadings

Multiply loadings with original data to get scores!

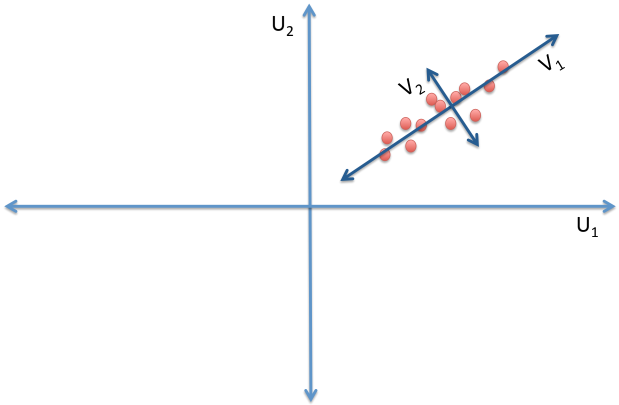
Principal Components Analysis

Basic idea: have a dataset



Principal Components Analysis

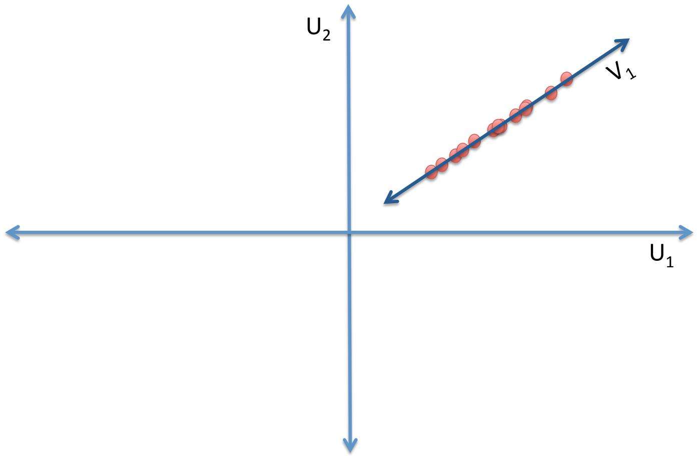
Basic idea: given dataset, want to find rotation (linear transformation) that best describes data



$y_1 = w_{11}x_1 + w_{12}x_2, \; y_2 = w_{21}x_1 + w_{22}x_2$

Principal Components Analysis

Basic idea: given linear transformation, we can throw out the less descriptive dimensions and still have a decent representation

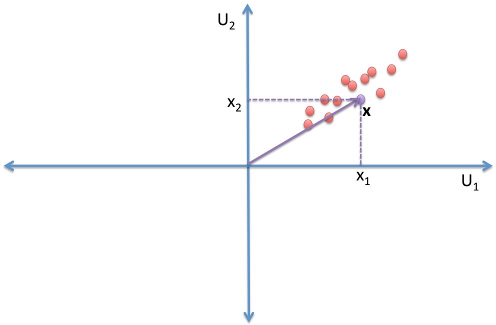


Principal Components Analysis

Mathematically, how do we do this?

Simple case: $\mathbf{x} \in \mathbb{R}^2$, and we want a good projection $y \in \mathbb{R}$

- make x_1, x_2 scalars
- make the axes vectors, $\mathbf{u}_1, \mathbf{u}_2$
- $\mathbf{x} = x_1\mathbf{u}_1 + x_2\mathbf{u}_2$



Principal Components Analysis

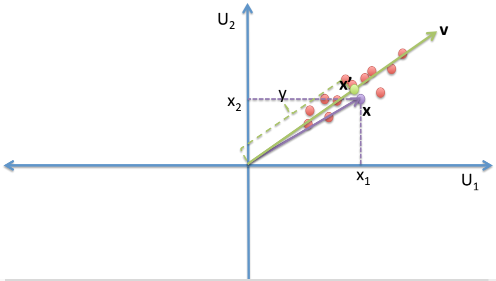
Mathematically, how do we do this?

Simple case: $\mathbf{x} \in \mathbb{R}^2$, and we want a good projection $y \in \mathbb{R}$

- now suppose we make a new direction vector $\mathbf{v}$
- let $\mathbf{w}_1$ be the linear transformation of $\mathbf{u}_1, \ \mathbf{u}_2$ into $\mathbf{v}$:

$$\mathbf{v} = w_{11}\mathbf{u}_1 + w_{12}\mathbf{u}_2$$

- we can project $\mathbf{x}$ onto $\mathbf{v}$ to make $\mathbf{x}' = y\mathbf{w}_1$
- two coordinates x_1, x_2 get turned into one, y



Principal Components Analysis

To find best $\mathbf{w}_1$:

- "closeness" is based on squared error between original points and new points
- usual notion of distance is Euclidean:

$$d(\mathbf{x}_i, \mathbf{x}_i') = \sqrt{\sum_{j=1}^{d} (x_{ij} - x_{ij})^2}$$

- we will measure distance as Euclidean distance *squared* $\rightarrow$ big deviations heavily penalized, smaller deviations less so
- objective is to minimize squared errors

$$\hat{\mathbf{w}}_1 = \arg\min_{\mathbf{w}_1} \left\{ \sum_{i=1}^{n} ||\mathbf{x}_i - \mathbf{x}_i'||_2^2 = \sum_{i=1}^{n} \sum_{j=1}^{d} \left(\mathbf{x}_{ij} - \mathbf{x}_{ij}'\right)^2 \right\}$$

Principal Components Analysis

To find best $\mathbf{w}_1$:

- also have constraint:

$$||\mathbf{w}_1||_2 = \sqrt{\sum_{j=1}^{d} w_{ij}^2} = 1$$

this means one step in old coordinates $\mathbf{u}$ is equal to one step in new coordinates $\mathbf{v}$ (this is called the L2 norm... we will use this and the L1 norm $||\mathbf{x}|| = \sum_{j=1}^{d} |x_j|$ a lot in this class)

Principal Components Analysis

To find the best single feature $\mathbf{w}_1$:

1. center the data (subtract mean)
2. find best single feature, $\mathbf{w}_1$ by

$$\begin{aligned}
\hat{\mathbf{w}}_1 &= \arg \min_{\mathbf{w}_1 : ||\mathbf{w}_1||_2 = 1} \sum_{i=1}^{n} \sum_{j=1}^{d} \left(x_{ij} - x'_{ij}\right)_2^2 \\
&= \arg \min_{\mathbf{w}_1 : ||\mathbf{w}_1||_2 = 1} \sum_{i=1}^{n} \sum_{j=1}^{d} \left(x_{ij} - y_i w_{1j}\right)^2 \\
&= \arg \min_{\mathbf{w}_1 : ||\mathbf{w}_1||_2 = 1} \sum_{i=1}^{n} \sum_{j=1}^{d} \left(x_{ij} - w_{1j} \mathbf{x}_i^T \mathbf{w}_1\right)^2 \\
&= \ldots \qquad \text{(matrix algebra)} \\
&= \arg \max_{\mathbf{w}_1 : ||\mathbf{w}_1||_2 = 1} \left\{ \mathbf{w}_1^T \mathbf{X}^T \mathbf{X} \mathbf{w}_1 = \frac{\mathbf{w}_1^T \left(\frac{1}{n} \mathbf{X}^T \mathbf{X}\right) \mathbf{w}_1}{\mathbf{w}_1^T \mathbf{w}_1} \right\}
\end{aligned}$$

Principal Components Analysis

Fun fact: $\frac{1}{n}\mathbf{X}^T\mathbf{X} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^T$ is the maximum likelihood estimate of the covariance matrix for $[x_1, \ldots, x_d]^T$

Another fun fact:

$$\frac{\mathbf{w}_1^T\left(\frac{1}{n}\mathbf{X}^T\mathbf{X}\right)\mathbf{w}_1}{\mathbf{w}_1^T\mathbf{w}_1}$$

is called a *Rayleigh quotient*

Since $\frac{1}{n}\mathbf{X}^T\mathbf{X}$ is symmetric:

- the Rayleigh quotient has a value that is equal to the largest eigenvalue of $\frac{1}{n}\mathbf{X}^T\mathbf{X}$
- the value of $\mathbf{w}_1$ that maximizes the Rayleigh quotient is the is the eigenvector associated with the largest eigenvalue

Principal Components Analysis

To get the next best feature, $\mathbf{w}_2$:

1. re-center the data by subtracting off the projection $\mathbf{x}'_i$ from $\mathbf{x}_i$:

$$\tilde{\mathbf{X}}_1 = \mathbf{X} - \mathbf{X}\mathbf{w}_1\mathbf{w}_1^T$$

2. minimize squared error as we did previously:

$$\hat{\mathbf{w}}_2 = \arg \min_{\mathbf{w}_2 \, : \, ||\mathbf{w}_2||_2 = 1} \mathbf{w}_2^T \tilde{\mathbf{X}}_1^T \tilde{\mathbf{X}}_1 \mathbf{w}_2$$

This process can be repeated until we have constructed $\mathrm{rank}(\mathbf{X})$ features

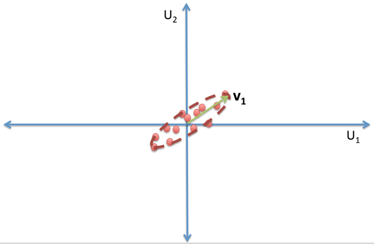
Principal Components Analysis

There is actually a nicer geometric interpretation.

- objective is to minimize squared errors

$$\hat{\mathbf{w}}_1 = \arg \min_{\mathbf{w}_1 : ||\mathbf{w}_1||_2 = 1} \sum_{i=1}^{n} \sum_{j=1}^{d} \left(\mathbf{x}_{ij} - \mathbf{x}'_{ij}\right)_2^2$$

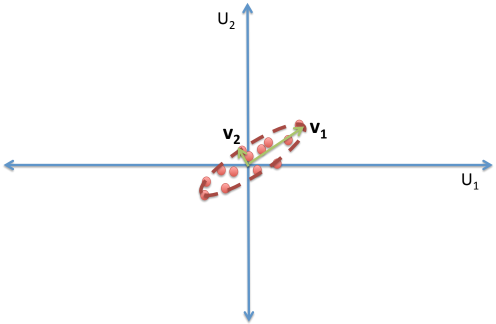
- center data
- fit a multivariate Gaussian distribution fit to the data
- Gaussian has mean 0, covariance Σ
- $\mathbf{w}_1$ is the direction of the covariance ellipse with maximum variance



Principal Components Analysis

This generalizes to multiple dimensions

- suppose we have d original dimensions and we would like K new ones
- the optimal vectors have the same direction as the K ellipse directions with maximum variance



Principal Components Analysis

How do we find the K most descriptive directions?

- center everything to have mean 0
- fit a multivariate Gaussian distribution to the data
- estimate the covariance matrix,

$$\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T$$

- find the *eigenvalues* and *eigenvectors* of the covariance matrix
- the K most descriptive directions are the eigenvectors associated with the K largest eigenvalues

Detour: Matrices, Eigenvalues, and Eigenvectors

Matrices can be used to describe transformations:

- $\mathbf{x} \to \mathbf{A}\mathbf{x}$ is a transformation of $\mathbf{x}$
- Examples (look at these graphically):
  - $\mathbf{x} =$ $(1,1)$, $(1,0)$, $(1,-1)$, $(0,1)$, $(-1,1)$, $(-1,0)\,(-1,1)$, $(0,-1)$
  - $\mathbf{A} = \left[\begin{array}{cc} 2 & 1 \\ 1 & 2 \end{array}\right]$
  - $\mathbf{A} = \left[\begin{array}{cc} 2 & 0 \\ 0 & 2 \end{array}\right]$

Detour: Matrices, Eigenvalues, and Eigenvectors

Matrices can be used to describe transformations:

- under transformations, some of the original directions may be preserved

$$\mathbf{Ax} = \lambda \mathbf{x}$$

- the direction $\mathbf{x}$ is an *eigenvector* of $\mathbf{A}$
- the scalar λ is an *eigenvalue* $\mathbf{A}$
- the eigenvalue describes the magnitude of the change in $\mathbf{x}$ under $\mathbf{A}$

Wikipedia actually has a nice .gif demonstrating this

Detour: Matrices, Eigenvalues, and Eigenvectors

To find the eigenvectors of **A**, find the roots of the polynomial

$$\det(\mathbf{A} - \lambda I) = 0$$

Let's do this with our examples:

- $\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$

- $\mathbf{A} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$

Detour: Matrices, Eigenvalues, and Eigenvectors

We can use the eigenvalues to find the eigenvectors

- start with eigenvalue λ_i
- solve the set of linear equations

$$\mathbf{Ax} = \lambda_i \mathbf{x}$$

- $\mathbf{x}$ is the eigenvector associated with λ_i

Let's do this with our examples:

- $\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$
- $\mathbf{A} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$

Principal Components Analysis

Back to PCA...

1. center the data
2. compute the covariance matrix $\frac{1}{n}\mathbf{X}^T\mathbf{X}$
3. compute the eigenvectors of $\frac{1}{n}\mathbf{X}^T\mathbf{X}$ along with their eigenvalues to get loadings
4. for the eigenvectors with the K largest eigenvalues, make factor scores by setting

$$\mathbf{y}_i = \begin{bmatrix} y_{i1} \\ y_{i2} \\ \vdots \\ y_{iK} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1d} \\ w_{21} & w_{22} & \dots & w_{2d} \\ \vdots & & & \vdots \\ w_{K1} & w_{K2} & \dots & w_{Kd} \end{bmatrix} \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \\ \vdots \\ \vdots \\ x_{id} \end{bmatrix}$$

Principal Components Analysis

5. do computations in the (smaller) **y** space, the space of factor scores
6. transform results back to original space (if needed)

$$\mathbf{y}_i = \begin{bmatrix} y_{i1} \\ y_{i2} \\ \vdots \\ y_{iK} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1d} \\ w_{21} & w_{22} & \dots & w_{2d} \\ \vdots & & & \vdots \\ w_{K1} & w_{K2} & \dots & w_{Kd} \end{bmatrix} \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \\ \vdots \\ \vdots \\ x_{id} \end{bmatrix}$$

Principal Components Analysis

Let's do an example:

$$\mathbf{X} = \begin{bmatrix} -4 & -4 \\ -1 & 1 \\ 1 & -1 \\ 4 & 4 \end{bmatrix}$$

$$\frac{1}{n}\mathbf{X}^T\mathbf{X} =$$

$$\det(\frac{1}{n}\mathbf{X}^T\mathbf{X} - \lambda I) =$$

$$[\lambda_1, \lambda_2] =$$

Principal Components Analysis

Let's do an example:

$$\mathbf{X} = \begin{bmatrix} -4 & -4 \\ -1 & 1 \\ 1 & -1 \\ 4 & 4 \end{bmatrix}$$

$[\mathbf{w}_1, \mathbf{w}_2] =$

$[\mathbf{y}_1, \mathbf{y}_2] =$

PCA: Computational Underpinnings

So how do we get these values? Matrix factorization.

- can compute eigenvectors/eigenvalues of $\mathbf{X}^T\mathbf{X}$ through spectral decomposition
- can use *singular value decomposition* of $\mathbf{X}$
  - more computationally stable than first method
  - better when $d > n$
  - we won't get into the details

Question of the Day

I have found that giving you some simple practice questions helps improve test performance. At the beginning of next class, I will have one of you work this out on the board (voluntary basis).

Find the principal components for the following dataset:

$$\mathbf{X} = \begin{bmatrix} -6 & -4 \\ -2 & 3 \\ 2 & -3 \\ 6 & 4 \end{bmatrix}$$