CS 383/613 – Machine Learning

Curse of Dimensionality

Principal Component Analysis

Slides adapted from material created by E. Alpaydin
Prof. Mordohai, Prof. Greenstadt, Pattern Classification (2nd Ed.),
Pattern Recognition and Machine Learning

Objectives

- Curse of Dimensionality
- Principal Component Analysis (PCA)

Data Dimensionality

And the “curse”

Data Dimensionality

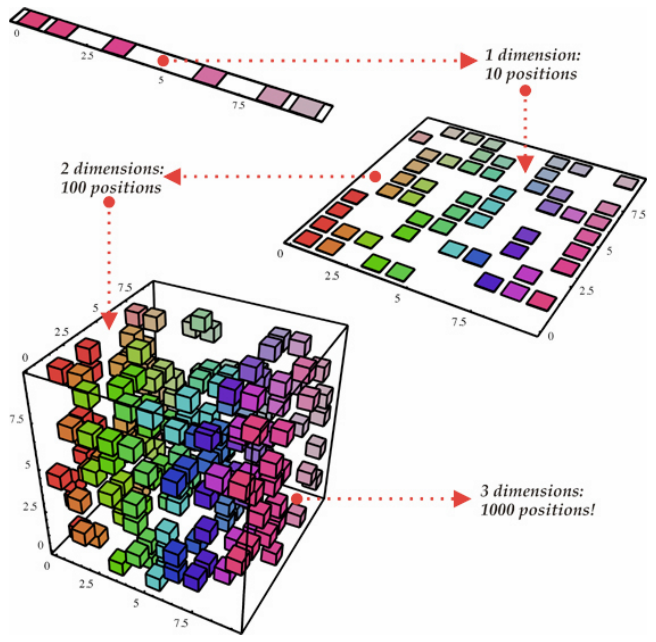
- Typically, our data has a lot of information in it
  - An image has millions of pixels
  - A textbook has thousands of words
- We call the amount of information we have for a given data sample, it's *dimension*
- Therefore, if data sample X_i has D values associated with it then we can write this as $X_i = \left(X_{i,j}\right)_{j=1}^{D}$ and call D the *dimensionality* of X_i

Data Dimensionality

- Can there be drawbacks of too much information (too high of dimensionality)?
- Computation cost
  - Both time and space efficiency
- Statistical Cost
  - Too specific? Doesn't generalize enough?
- Visualization
  - How can we look at the data to understand its structure?

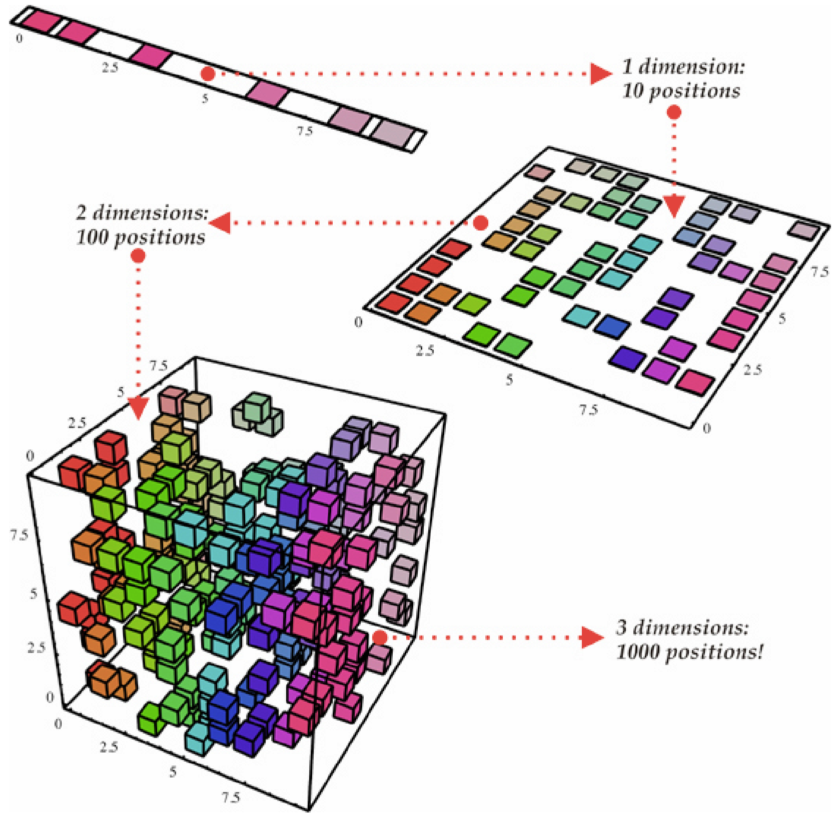
Curse of Dimensionality

- Imagine we only have one dimension
- Ideally, we'd have a sample for every single possible location in this space.
- If we have discrete space, and the range is [0, 9], then we'd just need 10 samples to have complete coverage
    - Great. No big deal!


1 dimension: 10 positions
2 dimensions: 100 positions
3 dimensions: 1000 positions!
0
2.5
5
7.5

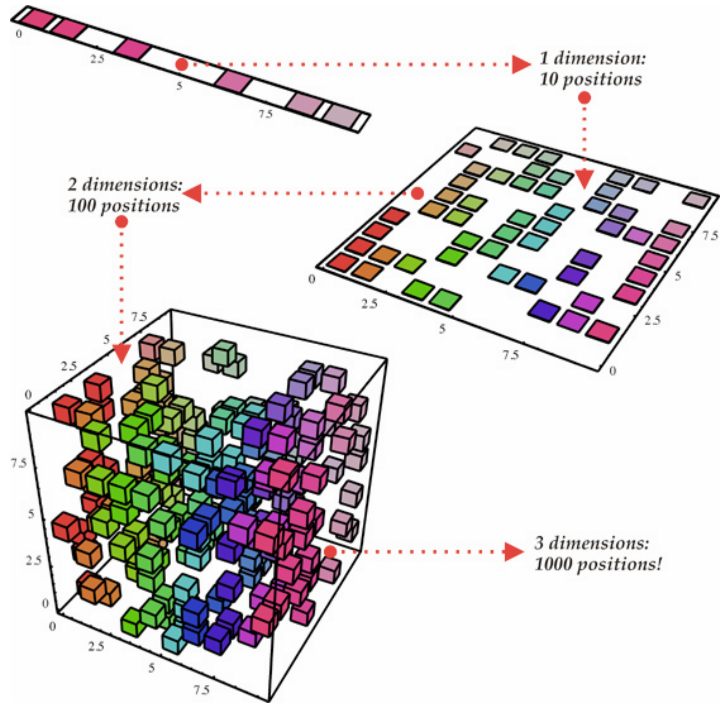
Curse of Dimensionality

- How about if we had two features ($D = 2$)?
- To have the same coverage we'd need 10^2=100 samples



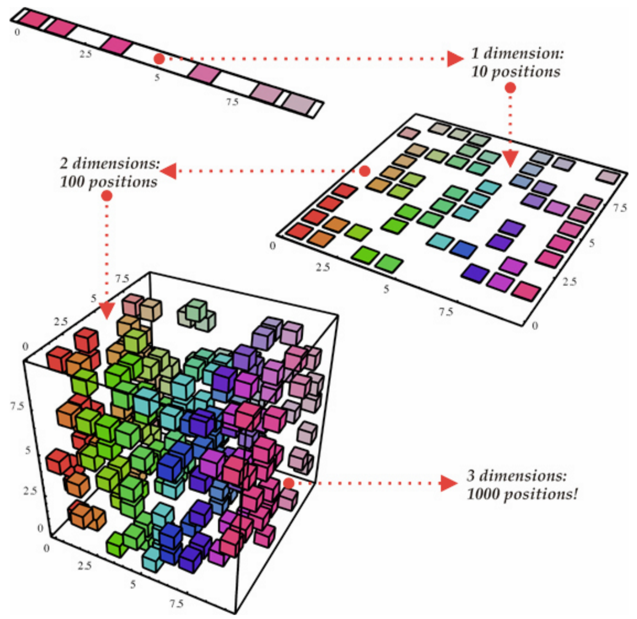
Curse of Dimensionality

- Ok. But in the real world our data ends up having D be really large
  - If we're trying to classify an image, each pixel is a feature, and thus $D = millions$
  - And each pixel might have 256 values
  - Therefore, to have complete coverage we'd need 256^{millions} samples
  - Impossible!


1 dimension: 10 positions
2 dimensions: 100 positions
3 dimensions: 1000 positions!

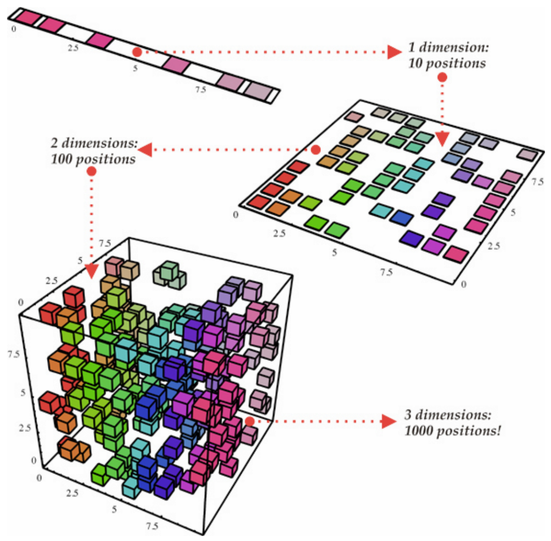
Curse of Dimensionality

- This is the *curse of dimensionality*
  - In higher dimension space we need exponentially more samples for equivalent coverage.
- Therefore, if our data is in high dimensional space, it will likely sparsely cover that space
  - And instances will be far apart from one another



Curse of Dimensionality

- This might be one motivation for *dimensionality reduction*
    - Going from higher dimension data to lower
- However, we want to do this "intelligently"
    - We don't want to lose much important information by doing this!

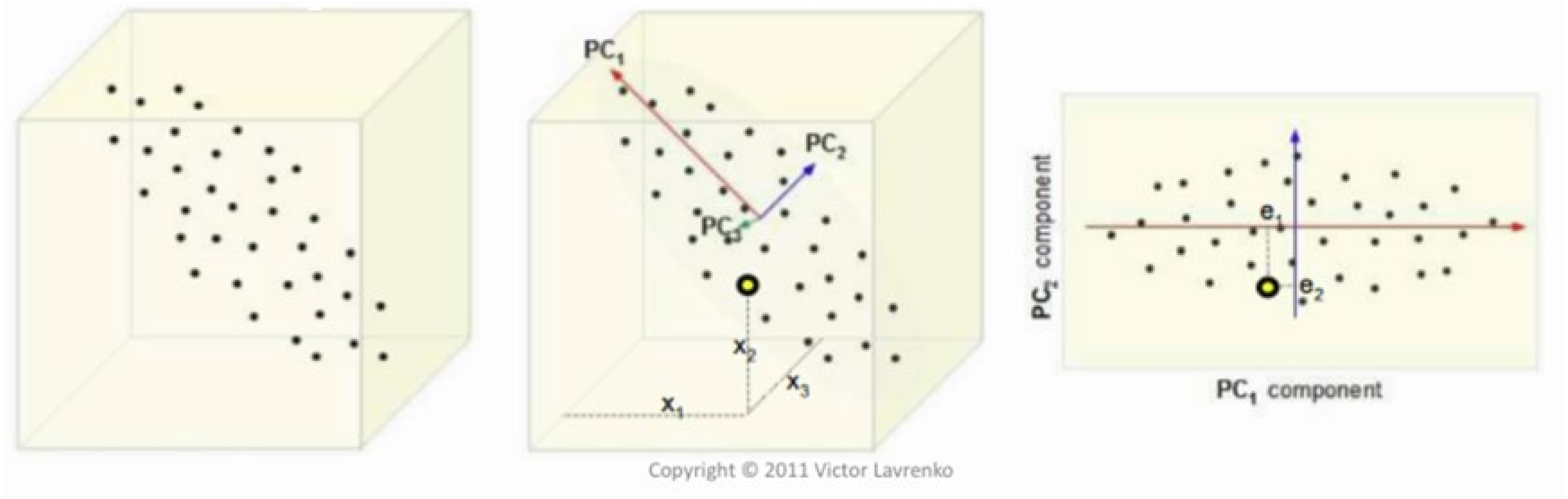


Dimensionality Reduction

- Goal: Represent instances with fewer variables
  - Try to preserve as much structure in the data as possible
  - If there is class information
    - Discriminative: increase class separability
- Benefits:
  - Need less data to cover the feature space
  - Easier learning – fewer parameters to learn
  - Easier visualization – hard to visualize more than 3D or 4D
- The technique we'll look at is *principal component analysis (PCA)*
  - And projection

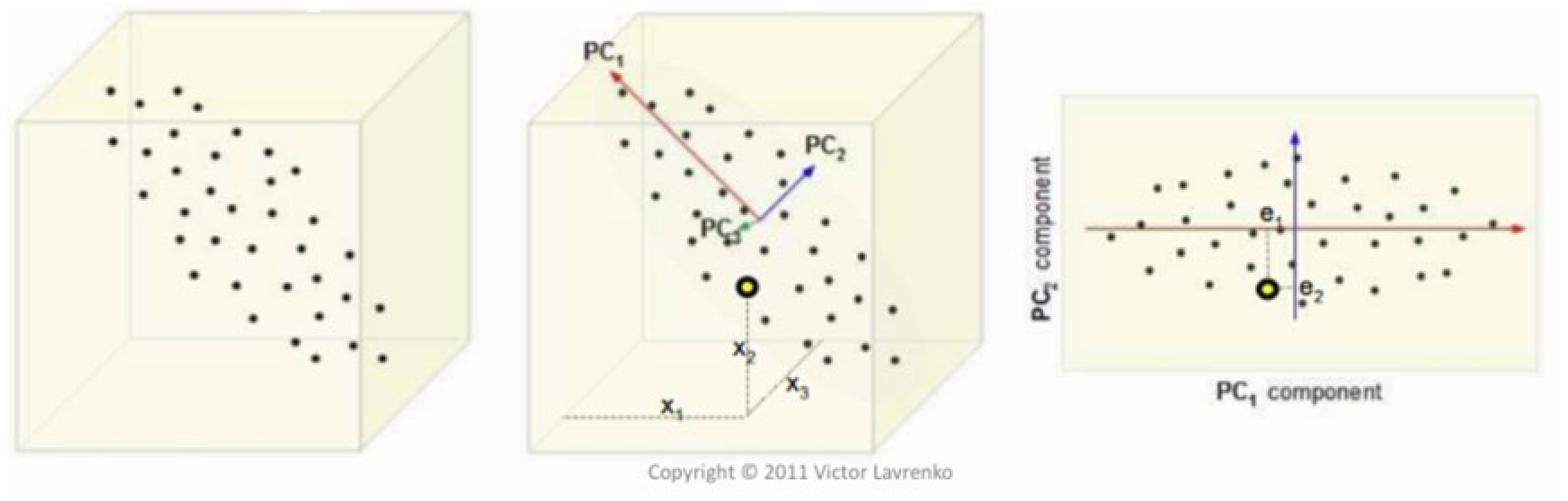
Principal Component Analysis (PCA)

- Principal component analysis (PCA) defines a set of principal components (basis)
    - 1st: direction of the greatest variability in the data
    - 2nd: perpendicular to 1st, greatest variability of what's left
    - Etc… until D, the original dimensionality



Principal Component Analysis (PCA)

- We can then choose the number of dimensions we want, $k < D$ and project the original data onto the principal components
- Each projection will result in a point on that axis, resulting in a new k-dimensional feature vector



PCA Derivation

- We want to find new points $Z = X\mathbf{w}$
- Given w as a $D \times 1$ column vector, the projection onto this axis is:

$$Z = X\mathbf{w}$$

- We want to maximize the variance of Z.
- In machine learning, it is common to set up a function, called an ***objective function* ,** that we are attempting to either minimize or maximize.
- For PCA, this objective function could be:

$$J = Var(Z) = Var(X\boldsymbol{w})$$

- Assuming our data's columns are zero mean (which they should be if we zscored our data)

$$J = \mathrm{Var}(X\mathbf{w}) = \frac{(X\mathbf{w})^T(X\mathbf{w})}{N-1} = \frac{\mathbf{w}^T X^T X\mathbf{w}}{N-1} = \mathbf{w}^T\Sigma\mathbf{w}$$

- Where Σ is the covariance matrix of X

PCA Derivation

$$J = \mathrm{Var}(X\mathbf{w}) = \frac{(X\mathbf{w})^T(X\mathbf{w})}{N-1} = \frac{\mathbf{w}^T X^T X\mathbf{w}}{N-1} = \mathbf{w}^T \Sigma \mathbf{w}$$

- So how do we find the value of $\boldsymbol{w}$ to maximize this?
- Calculus!?
  - Take the derivative with respect to $\boldsymbol{w}$, set it equal to zero and solve for $\boldsymbol{w}$
- With a little help from our math primers, we should arrive at

$$\frac{dJ}{d\boldsymbol{w}} = 2\Sigma\boldsymbol{w}$$

- Let $zeros$ be a column vector that is the same size as w
- Setting $\frac{dJ}{d\boldsymbol{w}} = zeros$ to solve for w we get:

$$\mathbf{w} = \Sigma^{-1} zeros$$

- Hmmm…. ☹

PCA Derivation

$$J = \mathbf{w}^T\Sigma\mathbf{w}, \qquad \frac{dJ}{d\boldsymbol{w}} = 2\Sigma\boldsymbol{w}, \qquad \mathbf{w} = \Sigma^{-1}zeros$$

- To obtain a non-trivial solution, we'll need to add in a constraint.
- Since $\boldsymbol{w}$ is mean to be an *axis* of our new coordinate system, let's want that the length of w, be one!
- Recall that the length of vector $\boldsymbol{w}$ is written as $|\boldsymbol{w}|$ and computed as:

$$|\boldsymbol{w}| = \sqrt{\boldsymbol{w}^T\boldsymbol{w}}$$

- We can now augment our objective function to penalize the squared length of $\boldsymbol{w}$ for being greater than one:

$$J = \boldsymbol{w}^T\Sigma\boldsymbol{w} - \lambda\left(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{w} - 1\right)$$

PCA Derivation

$$J = \boldsymbol{w}^T \Sigma \boldsymbol{w} - \lambda\left(\boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{w} - 1\right)$$

- Taking the derivate of this we get:

$$\frac{dJ}{d\boldsymbol{w}} = 2\boldsymbol{\Sigma}\boldsymbol{w} - 2\lambda\boldsymbol{w}$$

- Setting this equal to $zeros$ we get

$$2\Sigma\boldsymbol{w} - 2\lambda\boldsymbol{w} = zeros$$

- How can we solve for w (and λ)?

$$(\Sigma - \lambda I)w = zeros$$

- Dead end again:

$$w = (\Sigma - \lambda I)^{-1} zeros$$

Matrix Decomposition

- In linear algebra, we often attempt to find a triplet $(\boldsymbol{u}, \boldsymbol{v}, \lambda)$, that solves the equation $A\boldsymbol{u} = \lambda\boldsymbol{v}$, where A is a known matrix.
- Our problem can fall into this category:

$$2\Sigma\boldsymbol{w} - 2\lambda\boldsymbol{w} = zeros$$
$$\Sigma\boldsymbol{w} = \lambda\boldsymbol{w}$$

- Finding solutions involves *decomposing* the matrix such that

$$\mathrm{A} = USV^T$$

- Where:
  - U is a matrix of the *left eigenvectors* (assembled as columns, each of which is a u)
  - V are the right eigenvectors (assembled as columns, each of which is a v)
  - S is a matrix with the *eigenvalues* on its diagonal (each of which is λ).
- There are several algorithms that can solve for this, one of which is *singular value decomposition (SVD)*

PCA via Decomposition

$$\Sigma \boldsymbol{w} = \lambda \boldsymbol{w}$$

- So, we can just use SVD on Σ to get our solutions for $\boldsymbol{w}$ (and λ)!
- It's also worth noting that in the above formulation, $\boldsymbol{u} = \boldsymbol{v} = \boldsymbol{w}$.
- In this case, we can solve via an algorithms called *eigen-decomposition.*

Eig vs SVD

Eigen Decomposition

- Finds solution to equation in the form
$$A\boldsymbol{w} = \lambda \boldsymbol{w}$$
- Eigenvalues can be positive or negative.

Singular Value Decomposition

- Finds solutions to equation in the form of
$$A\boldsymbol{u} = \lambda \boldsymbol{v}$$
- Decomposes A into matrices such that $A = USV^T$
- Eigenvalues are non-negative.
- Typically, faster.

Matrix Decomposition

- In this course, we'll just use a linear algebra package to decompose a matrix into eigenvectors and eigenvalues.
- In MATLAB (numpy is similar), decomposing our covariance matrix via eigen-decomposition is done as:

$$[W, \lambda] = eig(\Sigma)$$

  - Where:
    - W is a matrix such that its columns are the eigenvectors.
    - λ is a **diagonal** matrix (all zeros except on the diagonal), such that the corresponding eigenvalues are on the diagonal.
- Doing this via singular value decomposition in MATLAB (numpy is similar) is

$$[\mathrm{U}, \lambda, V^T] = s\mathrm{vd}(\Sigma)$$

Choosing Eigenvectors

$$J = \boldsymbol{w}^T \Sigma \boldsymbol{w} - \lambda\left(\boldsymbol{w}^T \boldsymbol{w} - 1\right)$$

- So which combination of eigen-value/vector is the “best”?
- If our objective function is something we’re looking to *maximize* the best eigenvector is the one associated with the *largest* eigenvalue.
- Conversely, if our objective is something we’re looking to minimize, the best eigenvector is the one associated with the *smallest* eigenvalue.

Choosing Eigenvectors

- Do we want just one eigenvector?
- Maybe…
    - This gives us one axis/dimension.
- What if we want more?
- The grab the k most useful eigenvectors.
- How do we choose k?
- Perhaps the user/problem can determine this.

Choosing Eigenvectors

- Or maybe we just want to make sure to include up to some percent of the total eigenvalues?
  - Find k such that

$$\frac{\sum_{i=1}^{k}|\lambda_i|}{\sum_{i=1}^{D}|\lambda_i|} \geq \alpha$$

  - Typical values for the threshold α are 0.9 or 0.95
- We can think of this ratio as being the total amount of variance in the data explained by just k eigenvectors.

Using PCA for Dimensionality Reduction

- Now we have a set of k principal components (eigenvectors) $\boldsymbol{e_1}, \dots, \boldsymbol{e_k}$
  - Orthogonal, unit length
- Concatenated, they form an $D \times k$ *projection matrix* $W = [\boldsymbol{e_1}, \dots, \boldsymbol{e_k}]$
- Now can *project* our D-dimensional data into k-dimensions
  - $Z = XW$

Example

- Assume data

$$X = \begin{bmatrix} 7 & 1 \\ 2 & 4 \\ 2 & 3 \\ 3 & 6 \\ 4 & 4 \\ 9 & 4 \\ 6 & 8 \\ 9 & 5 \\ 8 & 7 \\ 10 & 8 \end{bmatrix}$$

- What is the first principal component?
- What are the observations' values projected onto that component?

Example

1. First let's zero-mean our data so we can compute the covariance matrix as $\Sigma = \frac{X^T X}{N-1}$

$$X = \begin{bmatrix} 7 & 1 \\ 2 & 4 \\ 2 & 3 \\ 3 & 6 \\ 4 & 4 \\ 9 & 4 \\ 6 & 8 \\ 9 & 5 \\ 8 & 7 \\ 10 & 8 \end{bmatrix}$$

$$X \Rightarrow \begin{bmatrix} 1 & -4 \\ -4 & -1 \\ -4 & -2 \\ -3 & 1 \\ -2 & -1 \\ 3 & -1 \\ 0 & 3 \\ 3 & 0 \\ 2 & 2 \\ 4 & 3 \end{bmatrix}$$

Example

2. Compute covariance matrix
   - This is quite easy since our data is already zero-centered!
     - $\Sigma = \frac{X^T X}{N-1} = \begin{bmatrix} 9.33 & 2.22 \\ 2.22 & 5.11 \end{bmatrix}$

$$X = \begin{bmatrix} 1 & -4 \\ -4 & -1 \\ -4 & -2 \\ -3 & 1 \\ -2 & -1 \\ 3 & -1 \\ 0 & 3 \\ 3 & 0 \\ 2 & 2 \\ 4 & 3 \end{bmatrix}$$

Example

$$\Sigma = \frac{X^T X}{N-1} = \begin{bmatrix} 9.33 & 2.22 \\ 2.22 & 5.11 \end{bmatrix}$$

3. Get the Eigenvalues/vectors of the covariance matrix
   - Eigenvalues = [4.16, 10.29]
   - Eigenvectors

$$\begin{bmatrix} 0.39 \\ -0.92 \end{bmatrix}, \begin{bmatrix} -0.92 \\ -0.39 \end{bmatrix}$$

   - Try drawing these!

Example

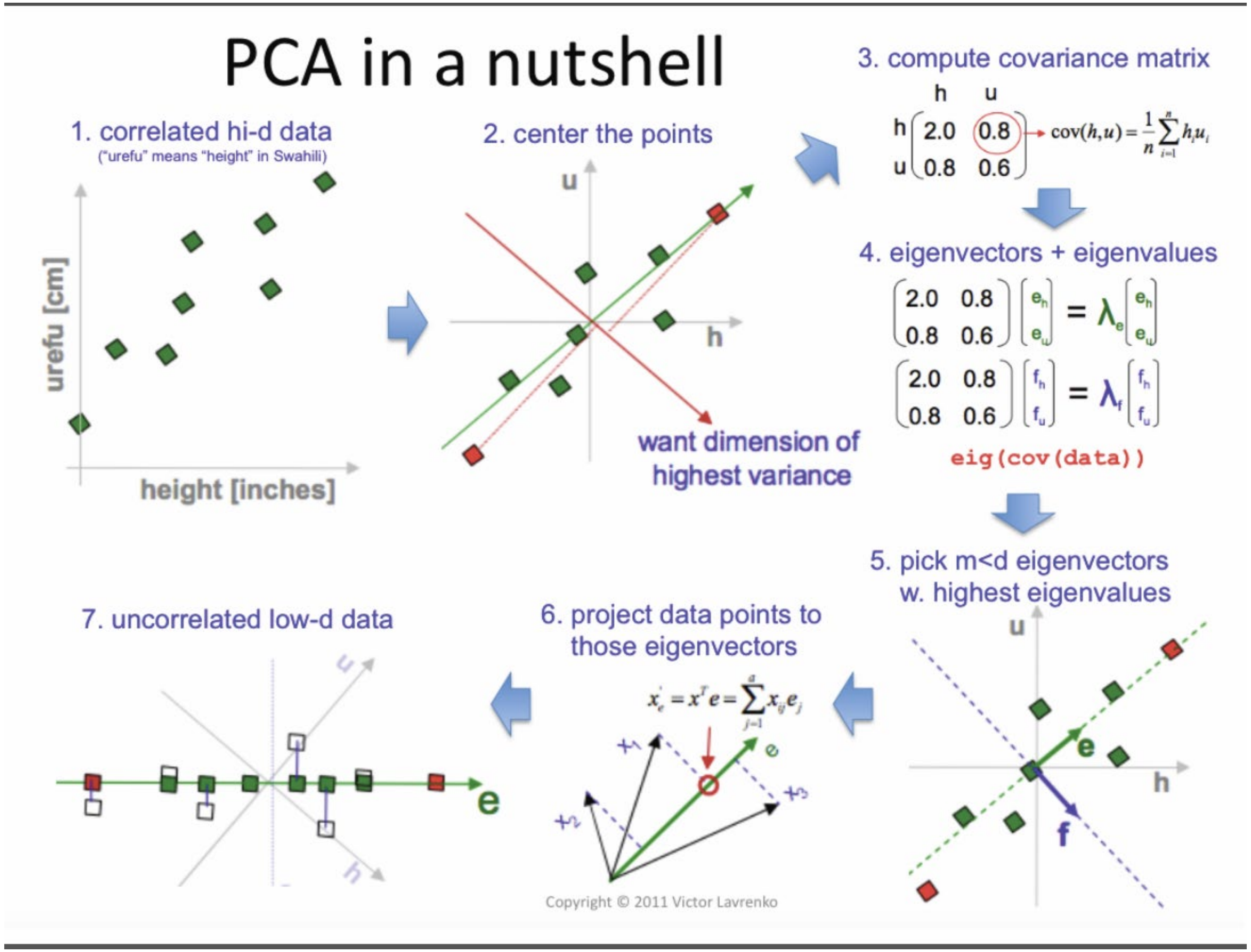
$$X = \begin{bmatrix} 1 & -4 \\ -4 & -1 \\ -4 & -2 \\ -3 & 1 \\ -2 & -1 \\ 3 & -1 \\ 0 & 3 \\ 3 & 0 \\ 2 & 2 \\ 4 & 3 \end{bmatrix}$$

- Eigenvalues = [4.16, 10.29]
- Eigenvectors

$$\begin{bmatrix} 0.39 \\ -0.92 \end{bmatrix}, \begin{bmatrix} -0.92 \\ -0.39 \end{bmatrix}$$

4. Finally let's project the points onto the single best vector (i.e., the one with the highest eigenvalue). Note we'll do this on the zero-centered data
   - $Z_{1,1} = X_1 W$ = [1,-4]$\begin{bmatrix} -0.92 \\ -0.39 \end{bmatrix}$= 0.66
   - Etc…
   - But actually, just do it all at once as $Z = XW$

PCA in a nutshell


1. correlated hi-d data
("urefu" means "height" in Swahili)
urefu [cm]
height [inches]
2. center the points
u
h
want dimension of highest variance
3. compute covariance matrix
$\begin{array}{c|cc} & h & u \\ \hline h & 2.0 & 0.8 \\ u & 0.8 & 0.6 \end{array}$
$\text{cov}(h,u) = \frac{1}{n}\sum_{i=1}^{n} h_i u_i$
4. eigenvectors + eigenvalues
$\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix}\begin{pmatrix} e_h \\ e_u \end{pmatrix} = \lambda_e \begin{pmatrix} e_h \\ e_u \end{pmatrix}$
$\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix}\begin{pmatrix} f_h \\ f_u \end{pmatrix} = \lambda_f \begin{pmatrix} f_h \\ f_u \end{pmatrix}$
eig(cov(data))
5. pick m<d eigenvectors w. highest eigenvalues
u
h
e
f
6. project data points to those eigenvectors
$x'_e = x^T e = \sum_{j=1}^{d} x_{ij} e_j$
e
7. uncorrelated low-d data
e
u
h
Copyright © 2011 Victor Lavrenko

Reconstruction

- PCA for feature reduction can be thought of as a *compression* or *encoding* process.
- So, can we somehow decompress/decode?
- Yes!
  - Kind of
- If we used **all** the eigenvectors, then we can perfectly reconstruct.
- Otherwise, we basically did lossy-compression, and therefore can only reconstruct to a point.
- So how do we do it?

Reconstruction

- Let's start off with the lossless case.
- It helps to think of eigenvectors as axes of coordinate systems.
- If we took the first (zero-meaned) observation, $\boldsymbol{x} = [1 \quad -4]$ and projected it using all the eigenvectors, then its location in the new coordinate system (defined by the eigenvectors) would be:

$$\boldsymbol{z} = [0.66 \quad 4.07]$$

- How can we get back to the original coordinate system?

Reconstruction

$$W = \begin{bmatrix} -0.92 & 0.39 \\ -0.39 & -0.92 \end{bmatrix}$$

$$\boldsymbol{x} = [1 \quad -4] \rightarrow [0.66 \quad 4.07] = \boldsymbol{z}$$

- How can we get back to the original coordinate system?
- We can move 0.66 units down the first (most important) eigenvector
$$\hat{\boldsymbol{x}} = 0.66[-0.92 \quad -0.39] = [-0.61 \quad -0.26]$$
- Then 4.07 down the second one
$$\hat{\boldsymbol{x}} \mathrel{+}= 4.07[0.39 \quad -0.92] = [1 \quad -4]$$
- Thus:
$$\hat{\boldsymbol{x}} = 0.66[-0.92 \quad -0.39] + 4.07[0.39 \quad -0.92] = [1 \quad -4]$$

Reconstruction

$$W = \begin{bmatrix} -0.92 & 0.39 \\ -0.39 & -0.92 \end{bmatrix}$$

$$\hat{\boldsymbol{x}} = 0.66[-0.92 \quad -0.39] + 4.07[0.39 \quad -0.92] = [1 \quad -4]$$

- How can I write this via linear algebra?

$$\hat{\boldsymbol{x}} = \boldsymbol{z}W^T$$

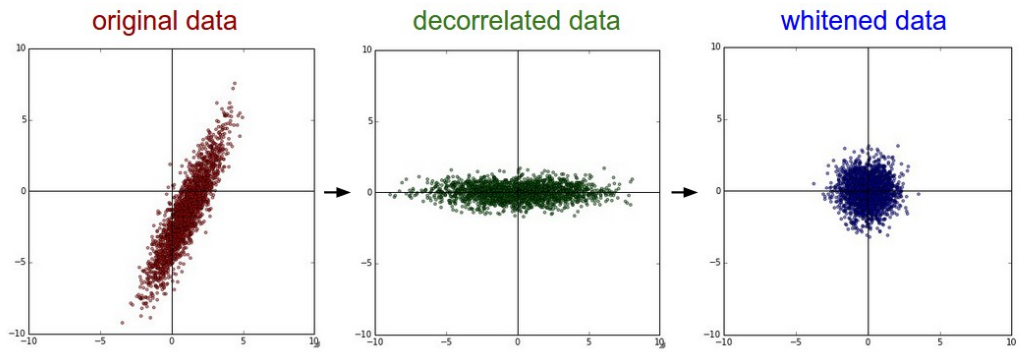
- Or for all observations Z:

$$\hat{X} = ZW^T$$

- If we used less eigenvectors, then we will have lossy reconstruction
  - But the good news is that most the information is encoded in the first few most relevant eigenvectors!

PCA

- Recall that the goal of PCA is to project data onto the direction of maximum variance.
  - Then onto the next direction of maximum variance, perpendicular to the first.
  - Etc..
- This can result in larger variance in the new first feature and decreasing variance thereafter.
- Therefore, if we want to z-score our data, we'll likely need to do it *after* PCA projection.



References

- Springer Text: 6.3-6.4, 10.2