

CPSC 340: Machine Learning and Data Mining

Principal Component Analysis

Fall 2017

Admin

- Assignment 3:
 - 2 late days to hand in tonight.
- Assignment 4:
 - Due Friday of next week.

Last Time: MAP Estimation

- MAP estimation maximizes posterior:

$$p(w | X, y) \propto p(y | X, w) p(w)$$

"posterior" "likelihood" "prior"

- Likelihood measures probability of labels 'y' given parameters 'w'.
- Prior measures probability of parameters 'w' before we see data.
- For IID training data and independent priors, equivalent to using:

$$f(w) = -\sum_{i=1}^n \log(p(y_i | x_i, w)) - \sum_{j=1}^d \log(p(w_j))$$

- So log-likelihood is an error function, and log-prior is a regularizer.
 - Squared error comes from Gaussian likelihood.
 - L2-regularization comes from Gaussian prior.

End of Part 3: Key Concepts

- Linear models predict based on linear combination(s) of features:

$$w^T x_i = w_1 x_{i1} + w_2 x_{i2} + \dots + w_d x_{id}$$

- We model non-linear effects using a change of basis:

- Replace x_i with z_i and use $w^T z_i$.
 - Examples include polynomial basis and (non-parametric) RBFs.

- Regression is supervised learning with continuous labels.

- Logical error measure for regression is squared error:

$$f(w) = \frac{1}{2} \|Xw - y\|^2$$

- Can be solved as a system of linear equations.

End of Part 3: Key Concepts

- We can reduce over-fitting by using regularization:

$$f(w) = \frac{1}{2} \|Xw - y\|^2 + \frac{\lambda}{2} \|w\|^2$$

- Squared error is not always right measure:
 - Absolute error is less sensitive to outliers.
 - Logistic loss and hinge loss are better for binary y_i .
 - Softmax loss is better for multi-class y_i .
- MLE/MAP perspective:
 - We can view loss as log-likelihood and regularizer as log-prior.
 - Allows us to define losses based on probabilities.

End of Part 3: Key Concepts

- Gradient descent finds local minimum of smooth objectives.
 - Converges to a global optimum for convex functions.
 - Can use smooth approximations (Huber, log-sum-exp)
- Stochastic gradient methods allow huge/infinite ‘n’.
 - Though very sensitive to the step-size.
- Kernels let us use similarity between examples, instead of features.
 - Let us use some exponential- or infinite-dimensional features.
- Feature selection is a messy topic.
 - Classic method is forward selection based on L0-norm.
 - L1-regularization simultaneously regularizes and selects features.

The Story So Far...

- Part 1: Supervised Learning.
 - Methods based on **counting and distances**.
- Part 2: Unsupervised Learning.
 - Methods based on **counting and distances**.
- Part 3: Supervised Learning.
 - Methods based on **linear models and gradient descent**.
- Part 4: Unsupervised Learning.
 - Methods based on **linear models and gradient descent**.

Motivation: Human vs. Machine Perception

- Huge difference between what we see and what computer sees:

What we see:



What the computer “sees”:



- But maybe images shouldn't be written as combinations of pixels.

Motivation: Pixels vs. Parts

- Can view 28x28 image as **weighted sum** of “single pixel on” images:

$$\begin{matrix} 3 \end{matrix} = 1 \begin{matrix} \cdot \\ \end{matrix} + 0 \begin{matrix} & \\ \cdot \end{matrix} + 1 \begin{matrix} & \\ \cdot \end{matrix} + 0.6 \begin{matrix} & \\ \cdot \end{matrix} + 0 \begin{matrix} \cdot \\ \end{matrix} + \dots$$

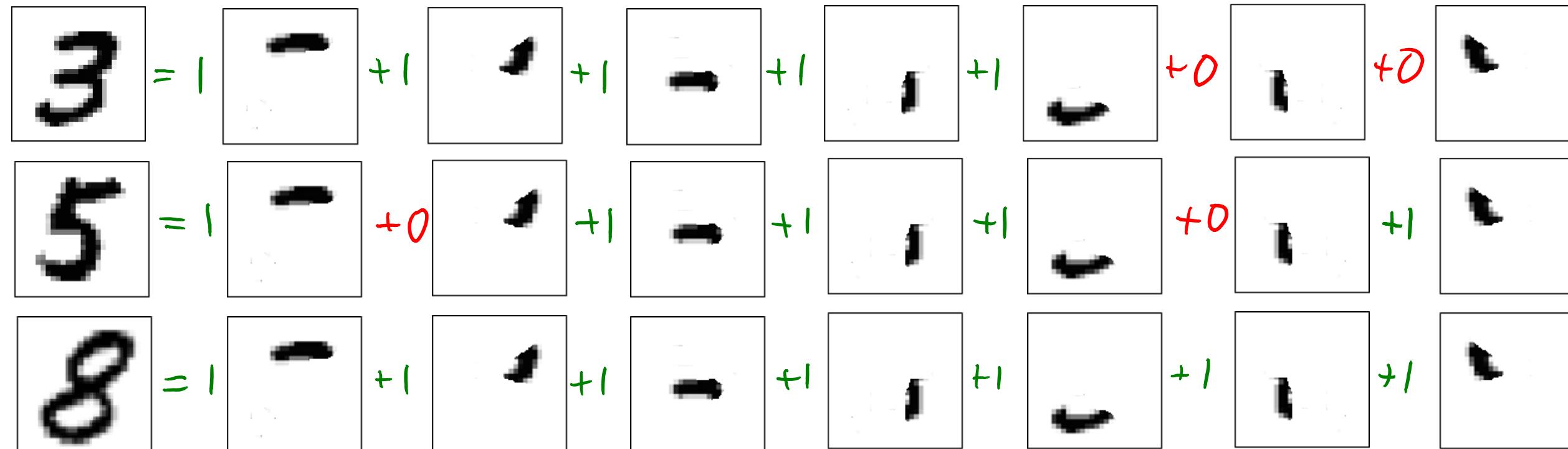
- We have one image for each pixel.
- The **weights** specify “how much of this pixel is in the image”.
 - A weight of zero means that pixel is white, a weight of 1 means it’s black.
- This is **non-intuitive**, isn’t a “3” made of **small number of “parts”**?

$$\begin{matrix} 3 \end{matrix} = 1 \begin{matrix} \text{---} \\ \end{matrix} + 1 \begin{matrix} \text{---} \\ 1 \end{matrix} + 1 \begin{matrix} \text{---} \\ \text{---} \end{matrix} + 1 \begin{matrix} & \\ \text{---} \end{matrix} + 1 \begin{matrix} & \\ \text{---} \end{matrix}$$

- Now the weights are “**how much of this part is in the image**”.

Motivation: Pixels vs. Parts

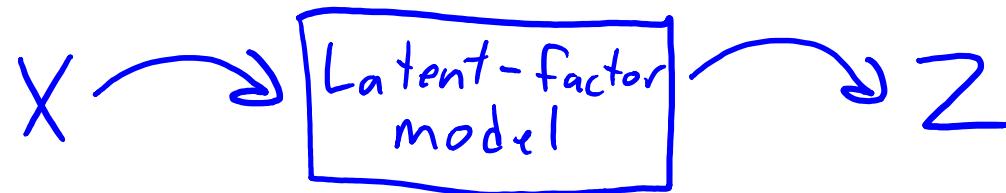
- We could represent other digits as different combinations of “parts”:



- Consider replacing images x_i by the weights z_i of the different parts:
 - The 784-dimensional x_i for the “5” image is replaced by 7 numbers: $z_i = [1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1]$.
 - Features like this could make learning much easier.

Part 4: Latent-Factor Models

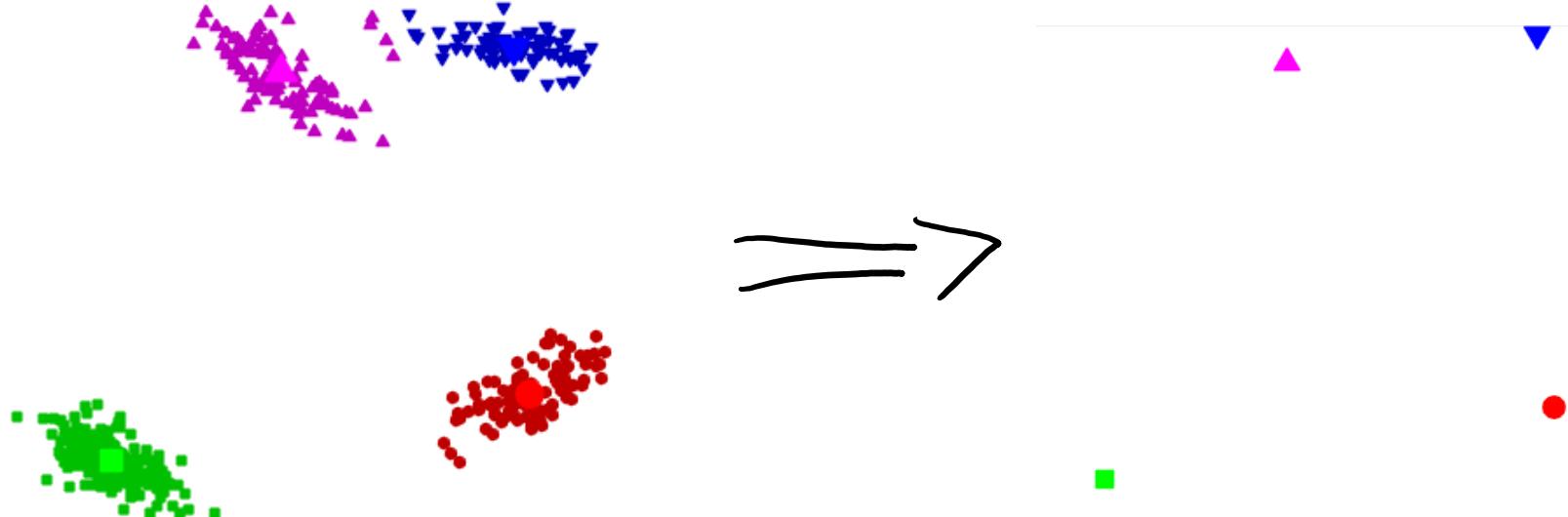
- The “part weights” are a change of basis from x_i to some z_i .
 - But in high dimensions, it can be hard to find a good basis.
- Part 4 is about learning the basis from the data.



- Why?
 - Supervised learning: we could use “part weights” as our features.
 - Outlier detection: it might be an outlier if isn’t a combination of usual parts.
 - Dimension reduction: compress data into limited number of “part weights”.
 - Visualization: if we have only 2 “part weights”, we can view data as a scatterplot.
 - Interpretation: we can try and figure out what the “parts” represent.

Previously: Vector Quantization

- Recall using **k-means** for vector quantization:
 - Run k-means to find a set of “means” w_c .
 - This gives a cluster y_i for each object ‘i’.
 - Replace features x_i by mean of cluster: $x_i \approx w_{y_i}$



- This can be viewed as a (really bad) latent-factor model.

Vector Quantization (VQ) as Latent-Factor Model

- When $d=3$, we could write x_i exactly as:

$$x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \end{bmatrix} = x_{i1} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + x_{i2} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + x_{i3} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (\text{this is like "one pixel on" representation of images})$$

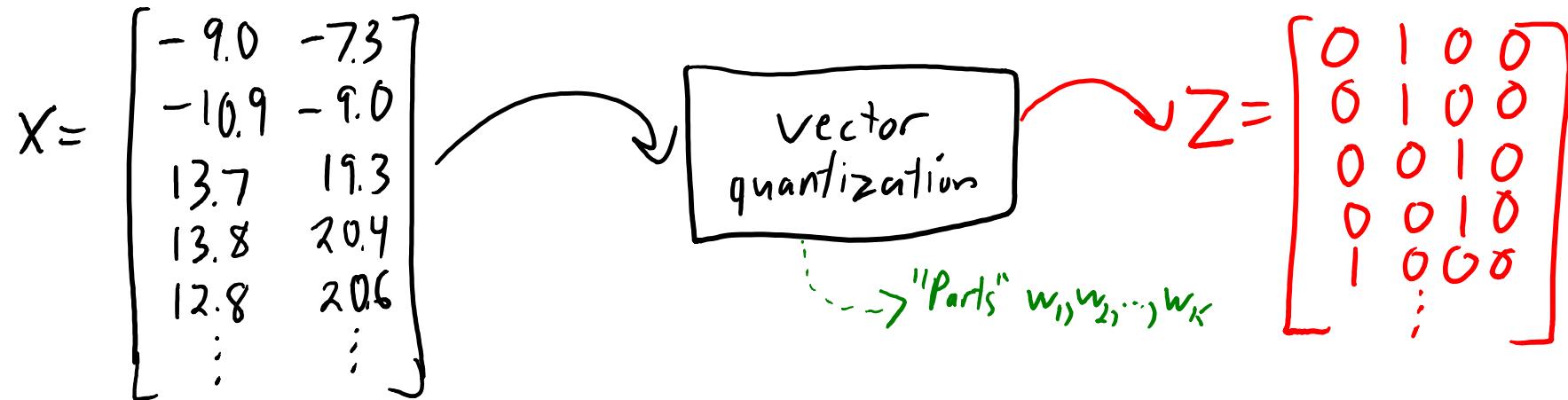
- If x_i is in cluster 2, VQ approximates x_i by mean w_2 of cluster 2:

$$x_i \approx w_2 = 0w_1 + 1w_2 + 0w_3 + \dots + 0w_k$$

- So in this example we would have $z_i = [0 \ 1 \ 0 \ \dots \ 0]$.
 - The “parts” are the means from k-means.
 - VQ only uses one part (the “part” from the cluster).

Vector Quantization vs. PCA

- So vector quantization is a **latent-factor model**:



- But it **only uses 1 part**, it's just memorizing 'k' points in x_i space.
 - What we want is **combinations of parts**.
- **PCA is a generalization that allows continuous ' z_i ':**
 - It can have more than 1 non-zero.
 - It can use fractional weights and negative weights.

$$Z = \begin{bmatrix} 0.2 & 1.6 \\ 0.3 & 1.5 \\ 0.1 & -2.7 \\ 0.3 & -2.7 \\ \vdots & \vdots \end{bmatrix}$$

Principal Component Analysis (PCA) Applications

- Principal component analysis (PCA) has been invented many times:

PCA was invented in 1901 by [Karl Pearson](#),^[1] as an analogue of the [principal axis theorem](#) in mechanics; it was later independently developed (and named) by [Harold Hotelling](#) in the 1930s.^[2] Depending on the field of application, it is also named the discrete [Kosambi–Karhunen–Loève transform](#) (KLT) in signal processing, the [Hotelling transform](#) in multivariate quality control, proper orthogonal decomposition (POD) in mechanical engineering, [singular value decomposition](#) (SVD) of \mathbf{X} (Golub and Van Loan, 1983), [eigenvalue decomposition](#) (EVD) of $\mathbf{X}^T \mathbf{X}$ in linear algebra, [factor analysis](#) (for a discussion of the differences between PCA and factor analysis see Ch. 7 of ^[3]), [Eckart–Young theorem](#) (Harman, 1960), or [Schmidt–Mirsky theorem](#) in psychometrics, [empirical orthogonal functions](#) (EOF) in meteorological science, [empirical eigenfunction decomposition](#) (Sirovich, 1987), [empirical component analysis](#) (Lorenz, 1956), [quasiharmonic modes](#) (Brooks et al., 1988), [spectral decomposition](#) in noise and vibration, and [empirical modal analysis](#) in structural dynamics.

standard deviation of 3 in roughly the (0.878, 0.478) direction and of 1 in the orthogonal direction. The vectors shown are the eigenvectors of the [covariance matrix](#) scaled by the square root of the corresponding eigenvalue, and shifted so their tails are at the mean.

Principal Component Analysis Notation

- PCA takes in a matrix 'X' and an input 'k', and outputs two matrices:

$$Z = \begin{bmatrix} z_1^T \\ z_2^T \\ \vdots \\ z_n^T \end{bmatrix} \Big\}_n \quad W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix} \Big\}_k = \begin{bmatrix} | & | & \cdots & | \\ w_1 & w_2 & \cdots & w_c \\ | & | & \cdots & | \end{bmatrix} \Big\}_c$$

- For row 'c' of W, we use the notation W_c .
 - Each W_c is a “part” (also called a “factor” or “principal component”).
- For row 'i' of Z, we use the notation z_i .
 - Each z_i is a set of “part weights” (or “factor loadings” or “features”).
- For column 'j' of W, we use the notation w_j .
 - Index 'j' of all the 'k' “parts” (value of pixel 'j' in all the different parts).

Principal Component Analysis Notation

- PCA takes in a matrix 'X' and an input 'k', and outputs two matrices:

$$Z = \begin{bmatrix} -z_1^T - \\ -z_2^T - \\ \vdots \\ -z_n^T - \end{bmatrix} \} n \quad W = \begin{bmatrix} -w_1^T - \\ -w_2^T - \\ \vdots \\ -w_k^T - \end{bmatrix} \} k = \begin{bmatrix} | & | & \cdots & | \\ w_1 & w_2 & \cdots & w_k \\ | & | & \cdots & | \end{bmatrix} \} k$$

- With this notation, we can write our approximation of one x_{ij} as:

$$x_{ij} \approx z_{i1} w_{1j} + z_{i2} w_{2j} + \cdots + z_{ik} w_{kj} = \sum_{c=1}^k z_{ic} w_{cj} = w_j^T z_i$$

- K-means: “take index ‘j’ of closest mean”.
- PCA: “use z_i to weight index ‘j’ of all means”.

- We can write approximation of the vector x_i as: $x_i \approx \begin{bmatrix} w_1^T z_i \\ w_2^T z_i \\ \vdots \\ w_k^T z_i \end{bmatrix} = W^T z_i$

PCA Objective Function

- K-means and PCA both use the same objective function:

$$f(W, z) = \sum_{i=1}^n \sum_{j=1}^d (w_j^\top z_i - x_{ij})^2$$

- In k-means, z_i has a single ‘1’ value and all other entries are zero.
- In PCA, z_i can be any real number.
- We can view this as solving ‘d’ regression problems:
 - Each w_j is trying to predict column ‘j’ of ‘X’ from the basis z_i .
 - But we’re also learning the features!
 - Each z_i is trying to predict row ‘i’ of ‘X’ from the “sub-parts” w_j .

Principal Component Analysis (PCA)

- Different ways to write the PCA objective function:

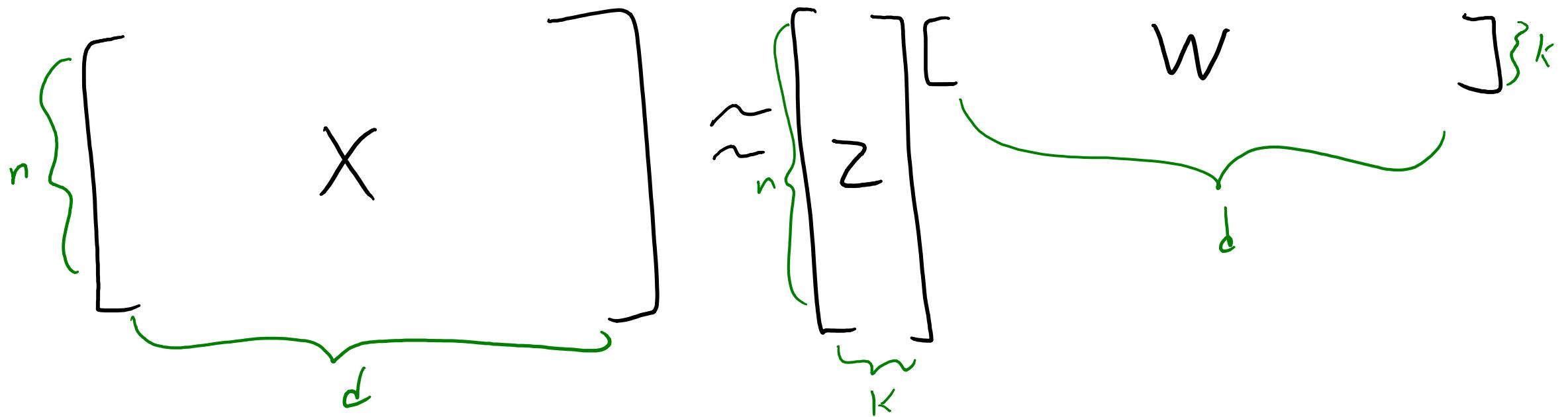
$$\begin{aligned} f(W, Z) &= \sum_{i=1}^n \sum_{j=1}^d (w_j^T z_i - x_{ij})^2 && \text{(approximating } x_{ij} \text{ by } w_j^T z_i \text{)} \\ &= \sum_{i=1}^n \|W^T z_i - x_i\|^2 && \text{(approximating } x_i \text{ by } W^T z_i \text{)} \\ &= \|Z^T W - X\|_F^2 && \text{(approximating } X \text{ by } Z^T W \text{)} \end{aligned}$$

- PCA is also called a “matrix factorization” model:

$$X \overset{n \times d}{\approx} Z \overset{n \times k}{\approx} W \overset{k \times d}{\approx}$$

PCA Applications

- Applications of PCA:
 - Dimensionality reduction: replace ‘X’ with lower-dimensional ‘Z’.
 - If $k \ll d$, then compresses data.
 - Often better approximation than vector quantization.



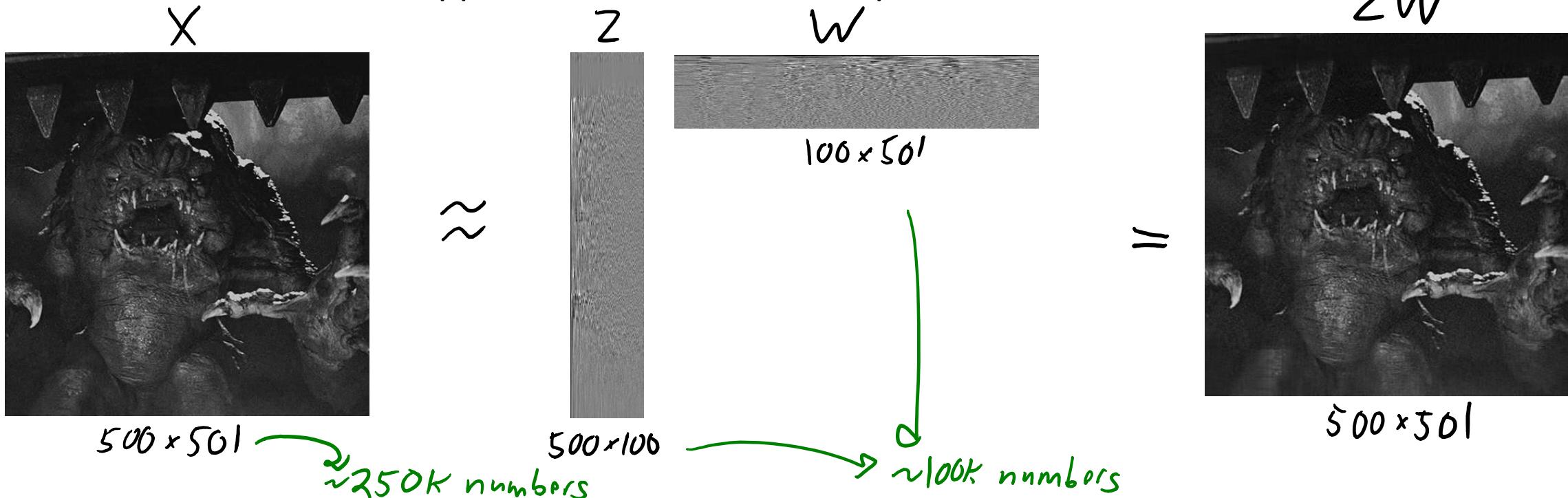
PCA Applications

- Applications of PCA:

- Dimensionality reduction: replace ‘X’ with lower-dimensional ‘Z’.

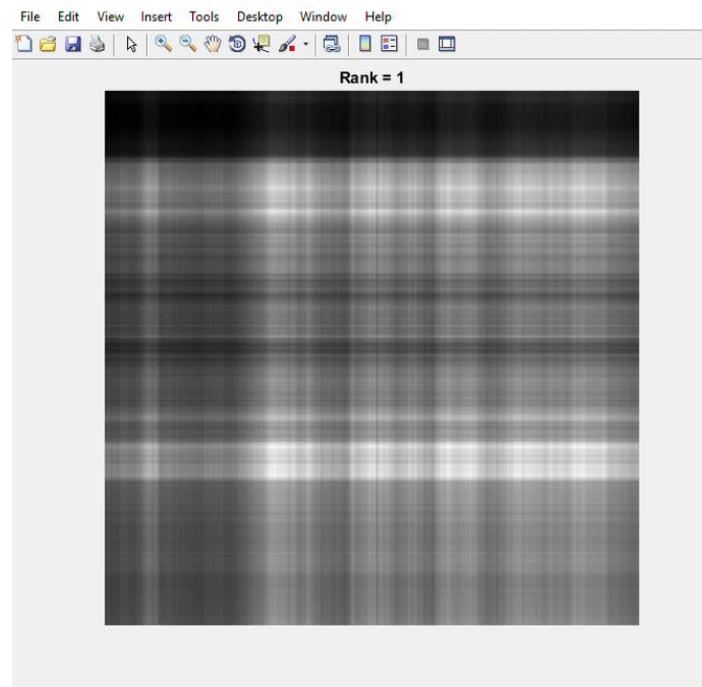
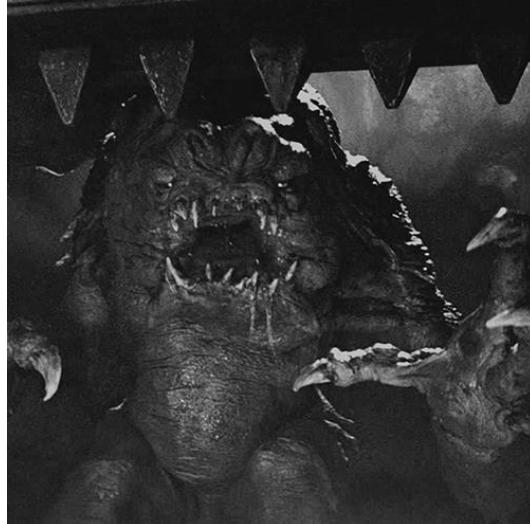
- If $k \ll d$, then compresses data.

- Often better approximation than vector quantization.



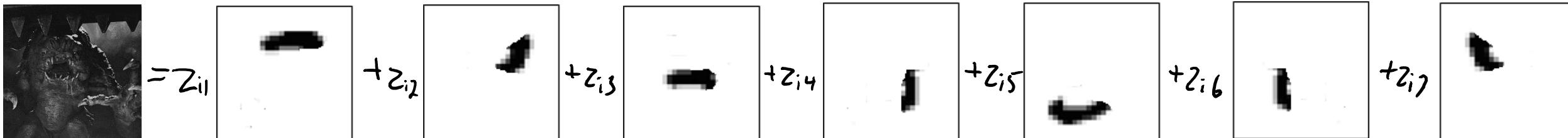
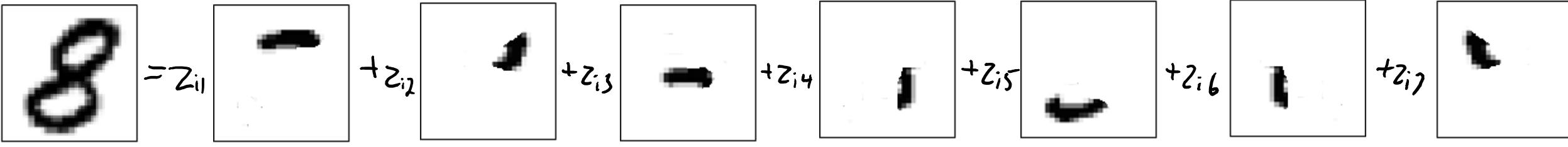
PCA Applications

- Applications of PCA:
 - Dimensionality reduction: replace ‘X’ with lower-dimensional ‘Z’.
 - If $k \ll d$, then compresses data.
 - Often better approximation than vector quantization.



PCA Applications

- Applications of PCA:
 - **Outlier detection:** if PCA gives poor approximation of x_i , could be ‘outlier’.
 - Though due to squared error **PCA is sensitive to outliers.**



PCA Applications

- Applications of PCA:
 - Partial least squares: uses PCA features as basis for linear model.

Compute approximation $X \approx ZW$

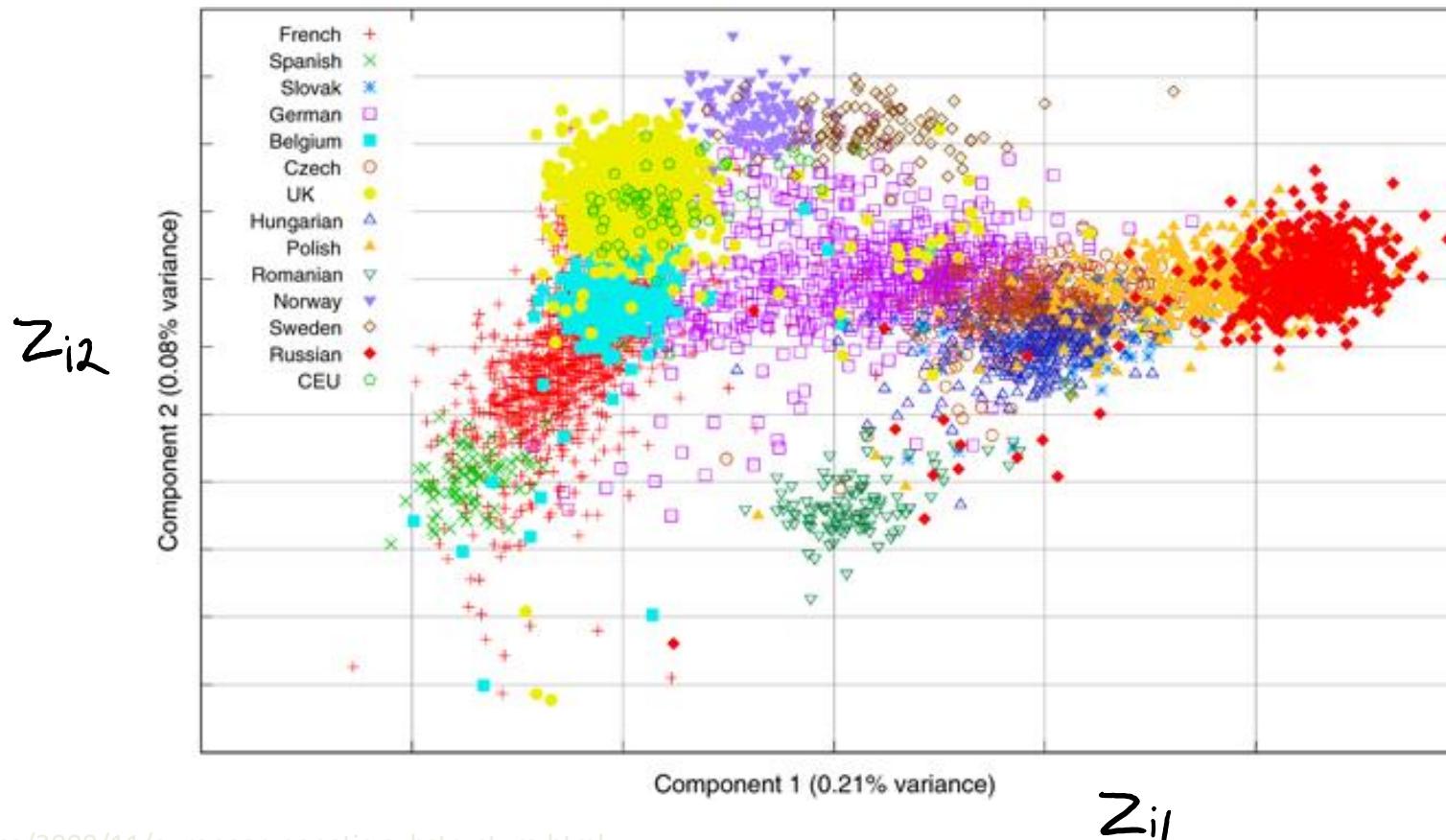
Now use Z as features in a linear model:

$$y_i = w^T z_i$$

a separate 'w'
trained for regression ↑ lower-dimensional than original features so less overfitting

PCA Applications

- Applications of PCA:
 - Data visualization: plot z_i with $k = 2$ to visualize high-dimensional objects.



PCA Applications

- Applications of PCA:
 - Data interpretation: we can try to assign meaning to latent factors W_c .
 - Hidden “factors” that influence all the variables.

Trait	Description
O penness	Being curious, original, intellectual, creative, and open to new ideas.
C onscientiousness	Being organized, systematic, punctual, achievement-oriented, and dependable.
E xtraversion	Being outgoing, talkative, sociable, and enjoying social situations.
A greeableness	Being affable, tolerant, sensitive, trusting, kind, and warm.
N euroticism	Being anxious, irritable, temperamental, and moody.

What is PCA actually doing?

When should PCA work well?

Today I just want to show geometry,
we'll talk about implementation on Friday.

Doom Overhead Map and Latent-Factor Models

- Original “Doom” video game included an “overhead map” feature:



- This map can be viewed as latent-factor model of player location.

Overhead Map and Latent-Factor Models

- Actual player location at time 'i' can be described by 3 coordinates:

$$x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \end{bmatrix}$$

← "x" coordinate
← "y" coordinate
← "z" coordinate

- The overhead map approximates these 3 coordinates with only 2:

$$z_i = \begin{bmatrix} z_{i1} \\ z_{i2} \end{bmatrix}$$

← "x" coordinate
← "y" coordinate

- Our k=2 latent factors are the following:

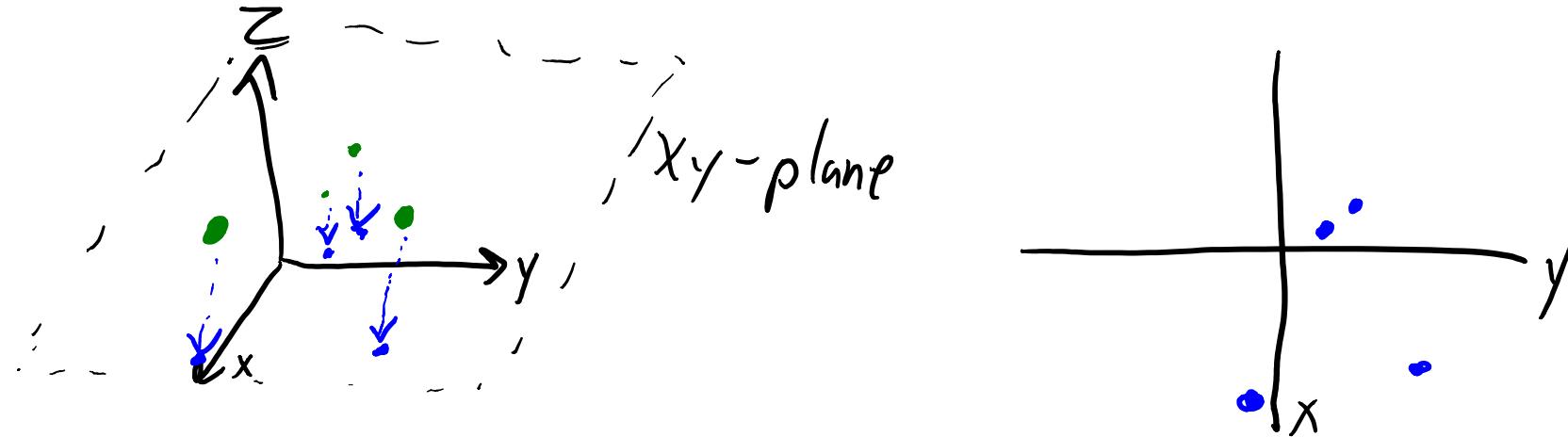
$$W = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

- So our approximation of x_i is:

$$x_i \approx z_{i1} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + z_{i2} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Overhead Map and Latent-Factor Models

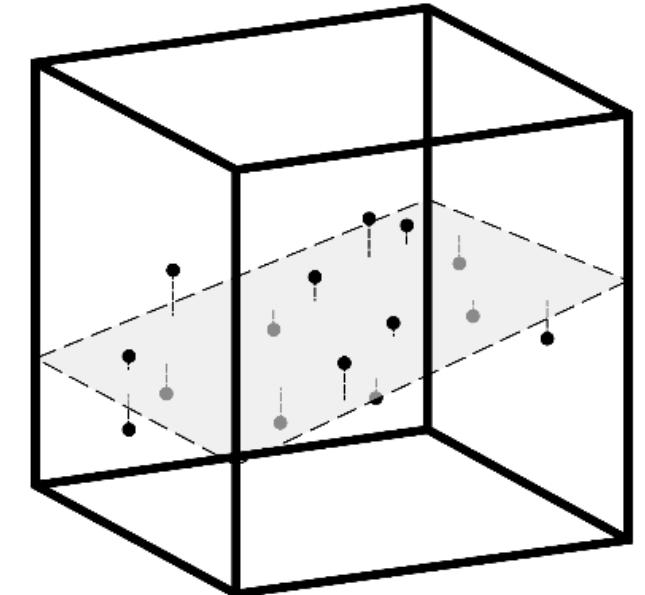
- The “overhead map” approximation just **ignores the “height”**.



- This is a **good approximation if the world is flat**.
 - Even if the character jumps, the first two features will approximate location.
- But it's a **poor approximation if heights are different**.

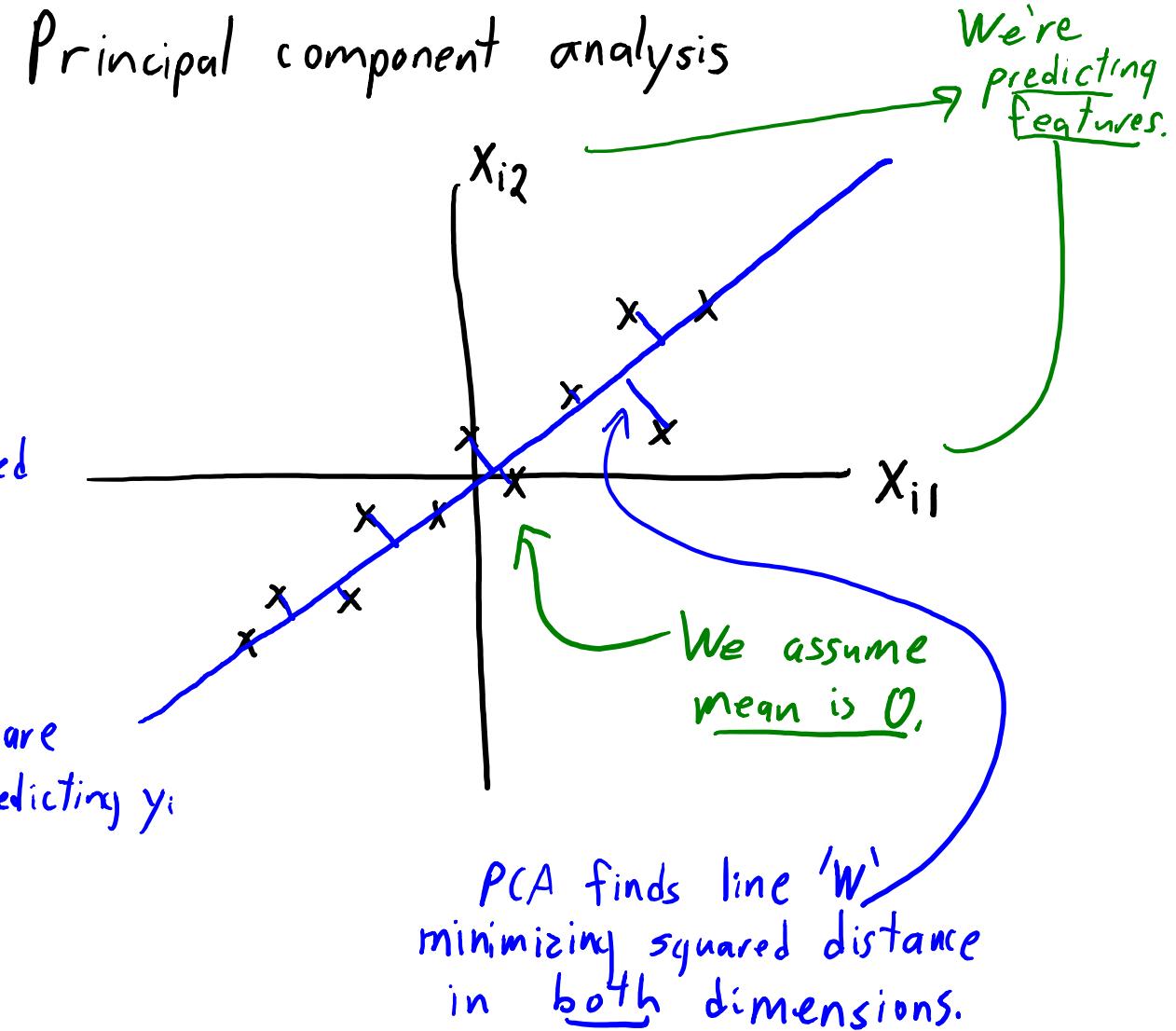
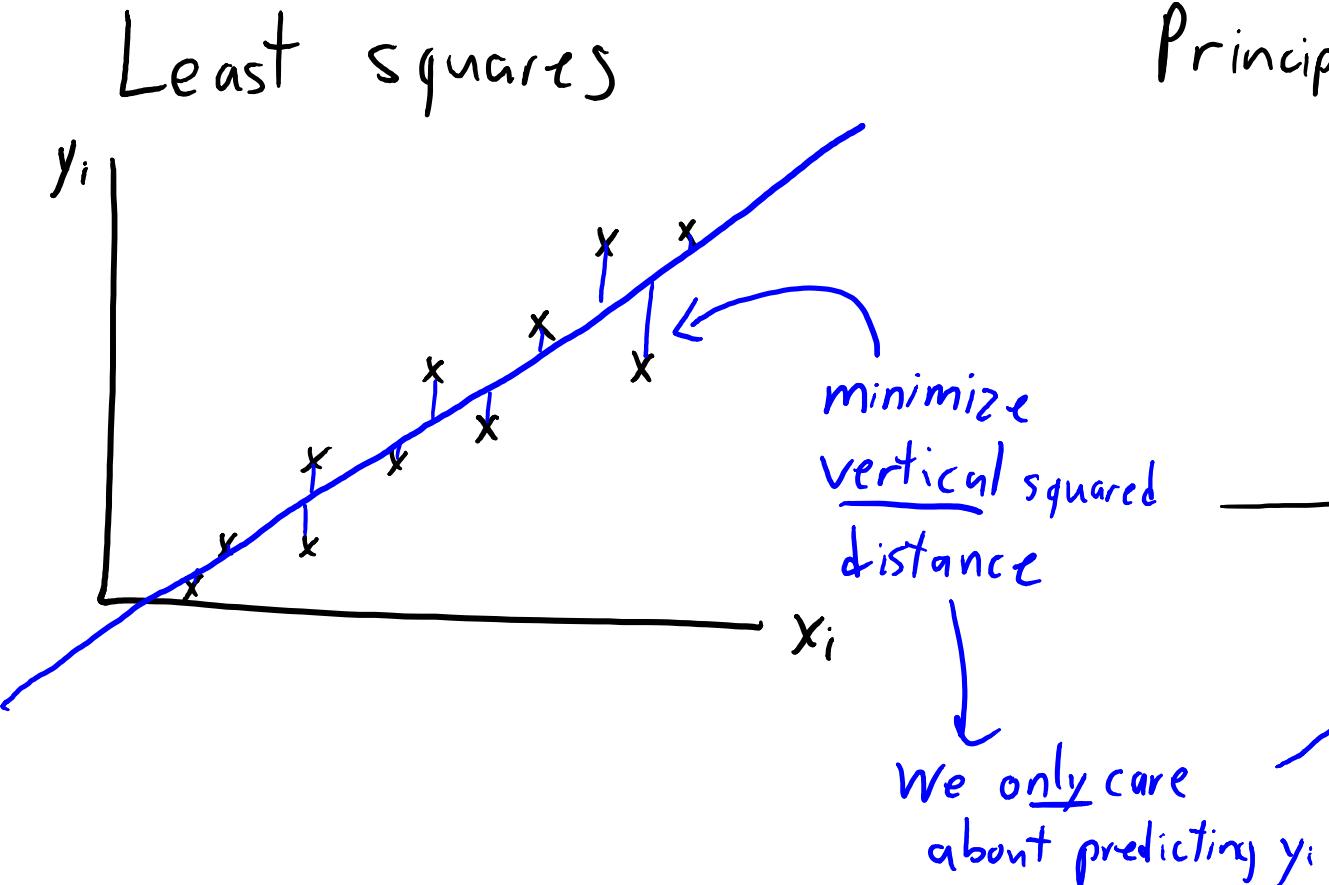
Overhead Map and Latent-Factor Models

- Consider these crazy goats trying to get some salt:
 - Ignoring height gives poor approximation of goat location.



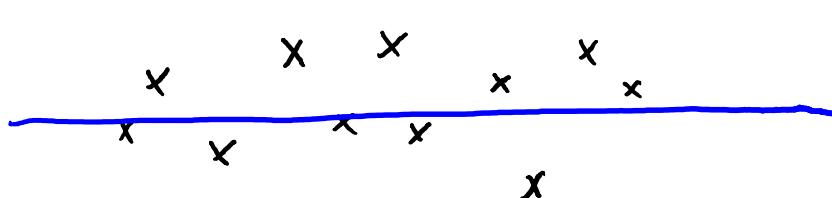
- But the “goat space” is basically a **two-dimensional plane**.
 - Better k=2 approximation: define ‘W’ so that combinations give the plane.

PCA with $d=2$ and $k=1$

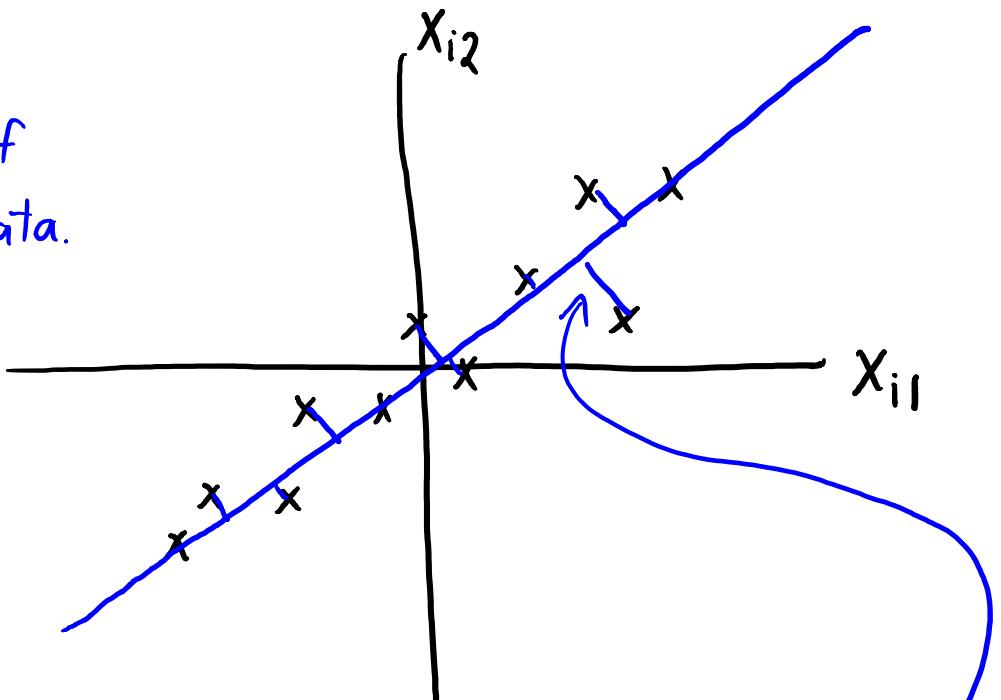


PCA with d=2 and k =1

Principal component analysis



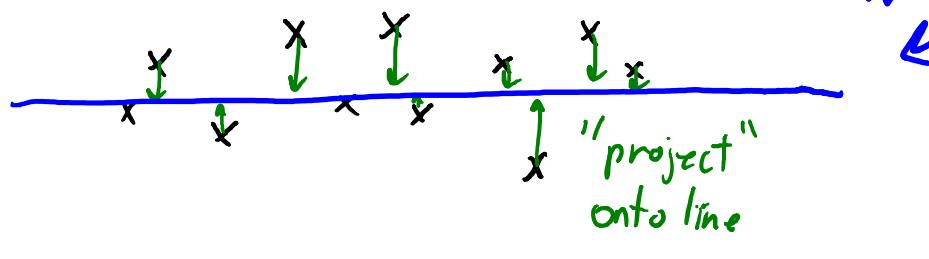
You can think of
'W' as rotating data.



PCA finds line 'W'
minimizing squared distance
in both dimensions.

PCA with $d=2$ and $k=1$

Principal component analysis

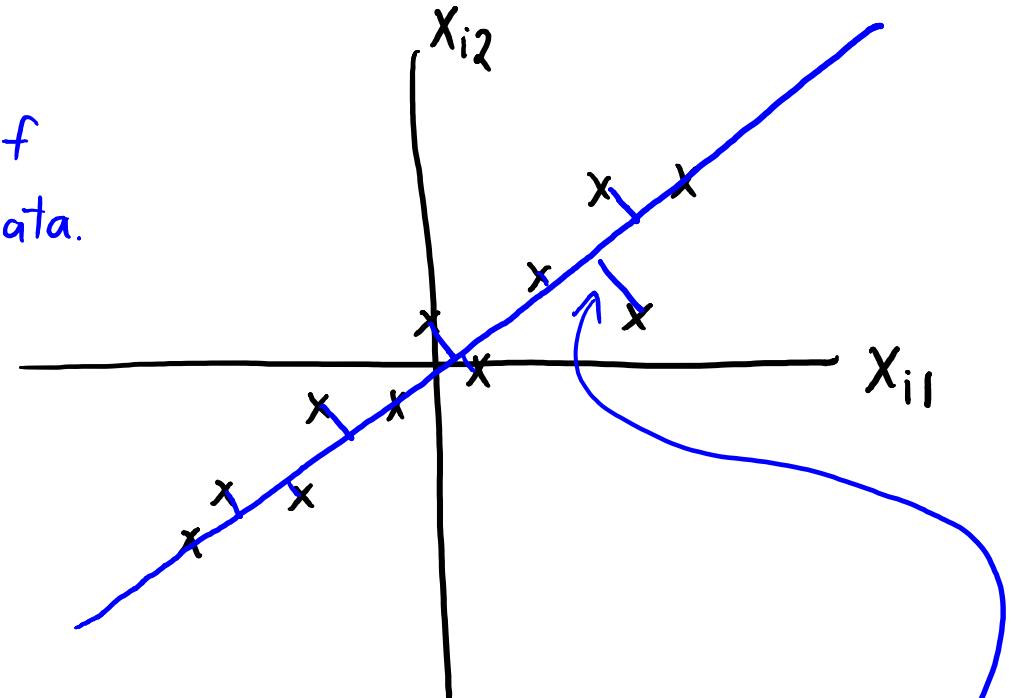


You can think of
'W' as rotating data.



Z_i can be interpreted as position along the line.

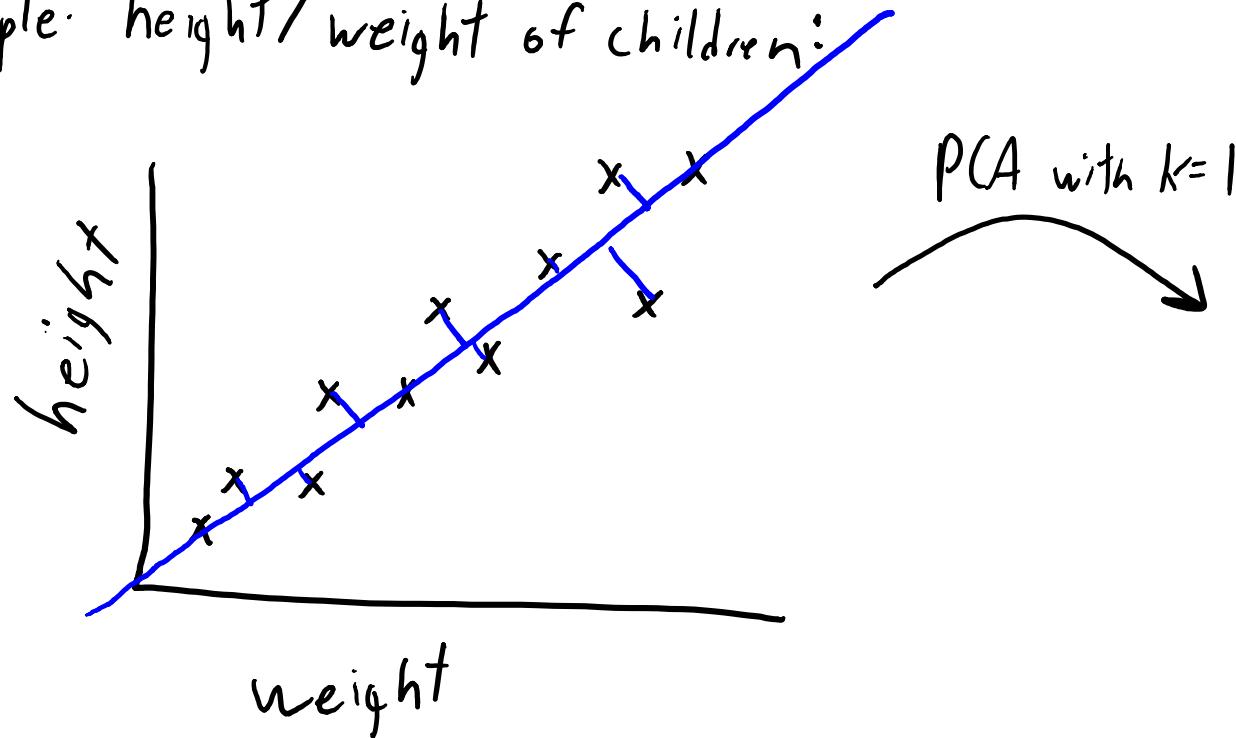
(turned a 2d dataset into a 1d dataset)



PCA finds line 'W'
minimizing squared distance
in both dimensions.

PCA with $d=2$ and $k=1$

Example: height/weight of children:



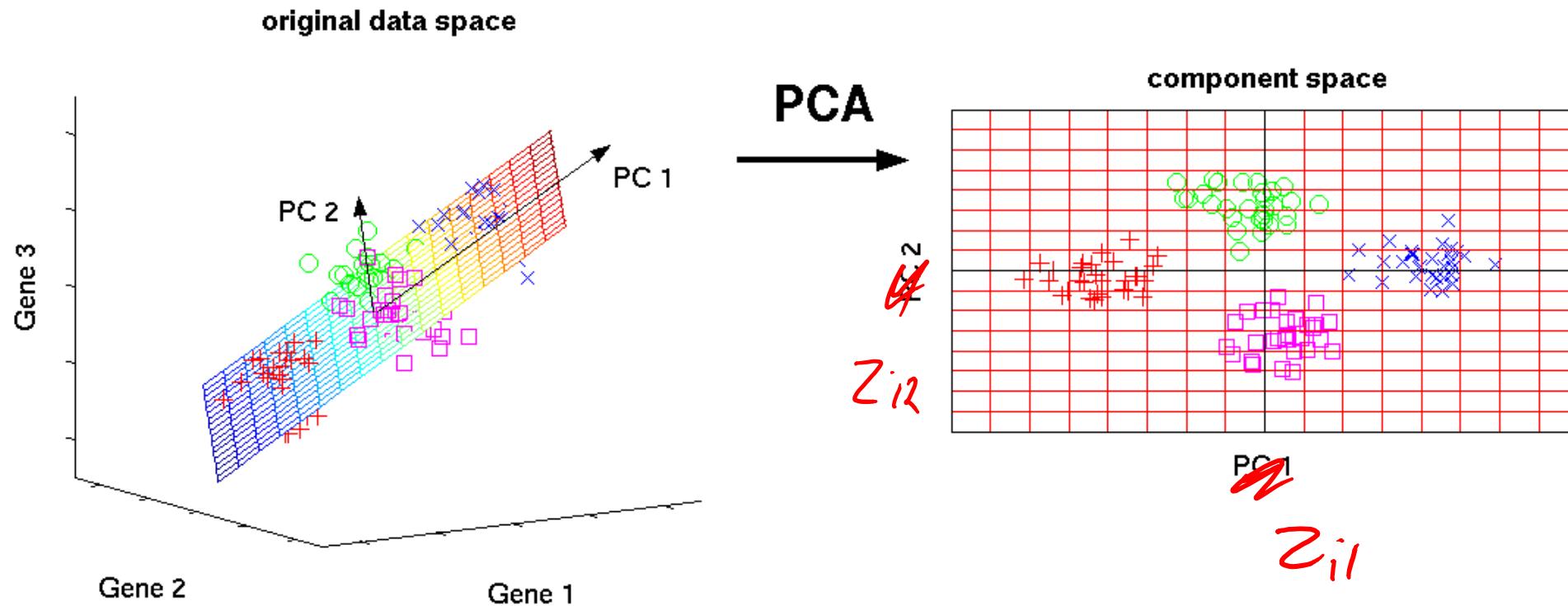
PCA with $k=1$



Latent factor could be viewed as measure of size.

PCA with $d=3$ and $k=2$.

- With $d=3$, PCA ($k=2$) finds plane minimizing squared distance to x_i .



- With $d=3$, PCA ($k=1$) finds line minimizing squared distance to x_i .

Summary

- Latent-factor models:
 - Try to learn basis Z from training examples X .
 - Usually, the z_i are “part weights” for “parts” W_c .
 - Useful for dimensionality reduction, visualization, factor discovery, etc.
- Principal component analysis:
 - Most common latent-factor model based on squared reconstruction error.
 - We can view ‘ W ’ as best lower-dimensional hyper-plane.
 - We can view ‘ Z ’ as the coordinates in the lower-dimensional hyper-plane.
- Next time: basis for faces (and annoying Facebook chat effects).