CPSC 340:
Machine Learning and Data Mining

PCA: the model ("predict")

Original version of these slides by Mark Schmidt, with modifications by Mike Gelbart.

Admin

- Assignment 4:
  - Solutions posted
- Assignment 5:
  - Coming soon (tomorrow?)
  - Remember to request partners
- 3rd Informal lunch:
  - Tomorrow, 12-1pm, Agora Café (basement of Macmillan building)

Last Time: MAP Estimation

- MAP estimation maximizes posterior:

$$\underbrace{p(w \mid X, y)}_{\text{"posterior"}} \propto \underbrace{p(y \mid X, w)}_{\text{"likelihood"}} \underbrace{p(w)}_{\text{"prior"}}$$

- 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:

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

- 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:

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

- We model non-linear effects using a change of basis:
  - Replace d-dimensional x_i with k-dimensional z_i and use $v^T z_i$.
  - 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

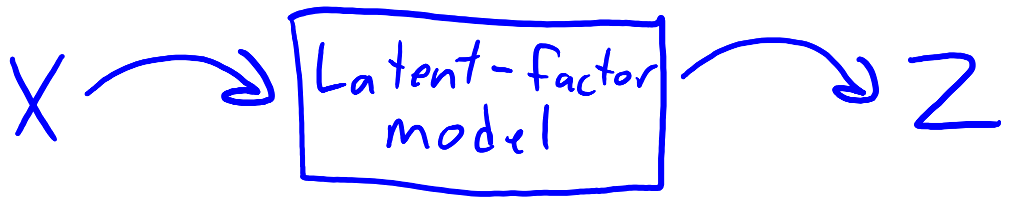
- Gradient descent finds local minimum of smooth objectives.
  - Converges to a global optimum for convex functions.
  - Can use smooth approximations (Huber, log-sum-exp)
- Stochastic gradient methods allow huge/infinite 'n'.
  - Though very sensitive to the step-size.
- Kernels let us use similarity 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 (just finished).
  - Methods based on linear models and gradient descent.
- Part 4: Unsupervised Learning (starting today).
  - Methods based on linear models and gradient descent.

Part 4: Latent-Factor Models

- In high dimensions, it can be hard to find a good basis.
- Part 4 is about learning the basis from the data.



- Main idea: let's "distill" the information from X down into Z
  - We do this by learning a transformation
  - It will be a linear transformation (for now)
  - The mapping will be stored in a matrix called W
  - The mapped values will be stored in a matrix called Z

Jupyter notebook demo

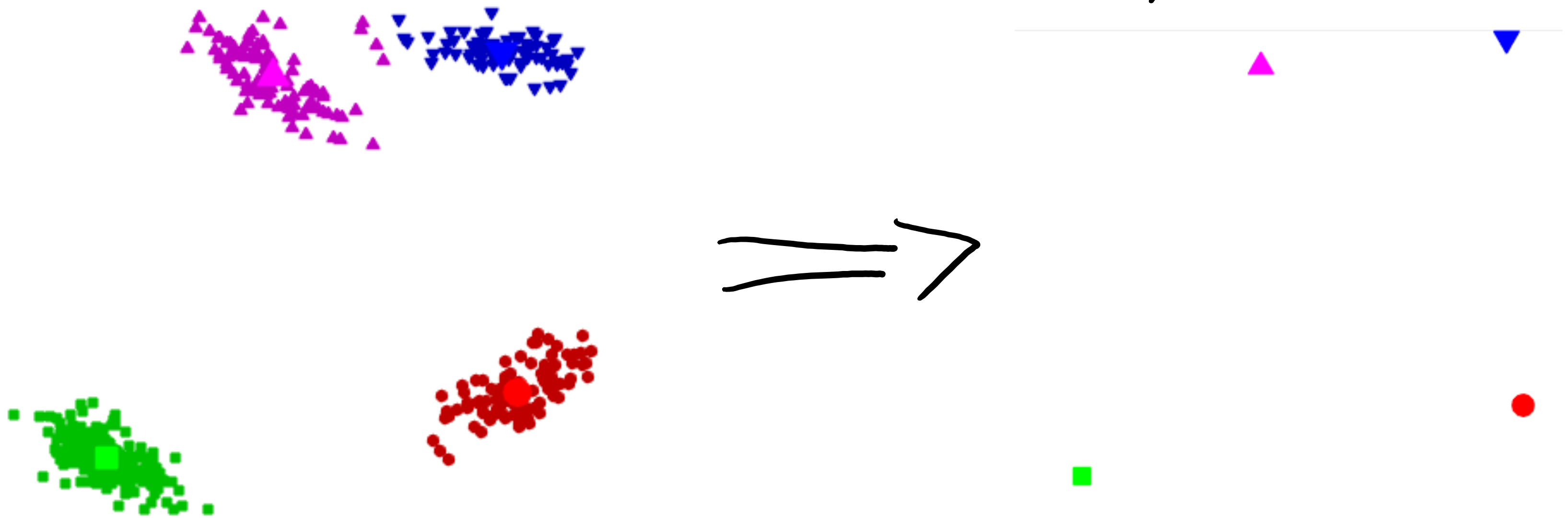
The Plan

- Rest of today's class:
  - What are W and Z exactly… and what does it all mean?

- Next class:
  - How to get W and Z given X (loss/training)?

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 $\hat{y}_i$ for each object 'i'.
  - Replace features x_i by mean of cluster: $\hat{x}_i \approx w_{\hat{y}_i}$



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

Vector Quantization (VQ) as Latent-Factor Model

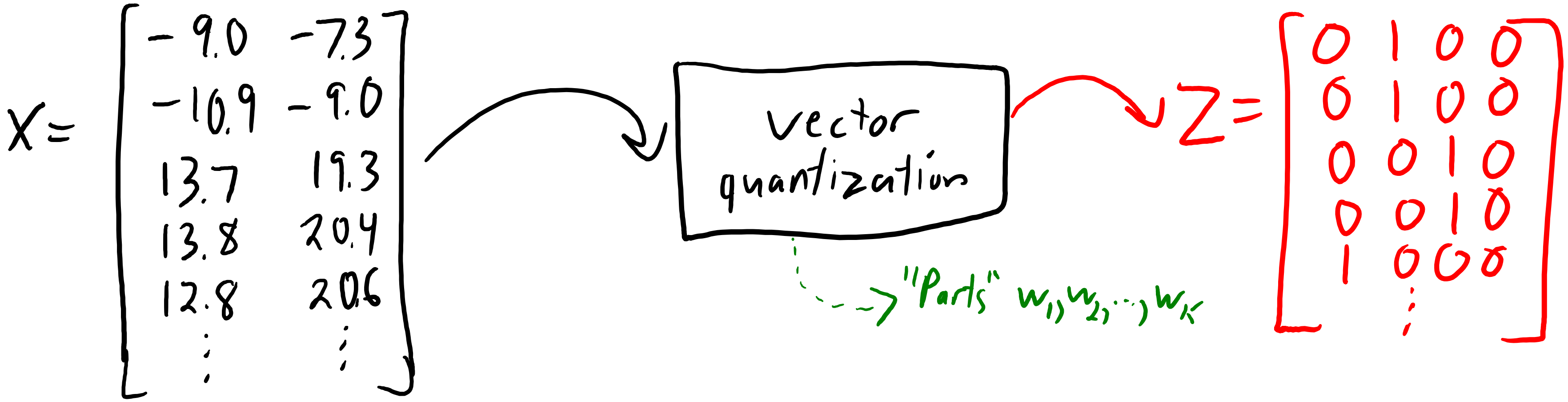
- If x_i is in cluster 2, VQ approximates x_i by mean w_2 of cluster 2:

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

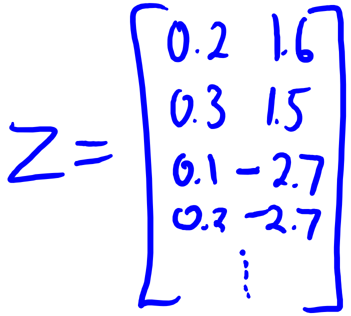
- So in this example we would have z_i = [0 1 0 … 0].
  – VQ only uses one factor (the particular cluster mean).

Vector Quantization vs. PCA

- So vector quantization is a latent-factor model:



- But it only uses 1 factor, it's just memorizing 'k' points in d-space.
  - What we want is combinations of factors.
- 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.



Principal Component Analysis Notation

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

$$Z = \begin{bmatrix} — z_1^T — \\ — z_2^T — \\ \vdots \\ — z_n^T — \end{bmatrix} \Big\} n \quad (n \times k) \qquad W = \begin{bmatrix} — w_1^T — \\ — w_2^T — \\ \vdots \\ — w_k^T — \end{bmatrix} \Big\} k \quad (k \times d) = \begin{bmatrix} | & | & & | \\ w^1 & w^2 & \cdots & w^d \\ | & | & & | \end{bmatrix} \Big\} k$$

- 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:

$$Z = \underbrace{\begin{bmatrix} - & z_1^T & - \\ - & z_2^T & - \\ & \vdots & \\ - & z_n^T & - \end{bmatrix}}_{k} \Big\} n \qquad W = \underbrace{\begin{bmatrix} - & w_1^T & - \\ - & w_2^T & - \\ & \vdots & \\ - & w_k^T & - \end{bmatrix}}_{d} \Big\} k = \underbrace{\begin{bmatrix} | & | & & | \\ w_1 & w_2 & \cdots & w_d \\ | & | & & | \end{bmatrix}}_{d} \Big\} k$$

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

$$\hat{x}_{ij} = z_{i1} w_{1j} + z_{i2} w_{2j} + \cdots + z_{ik} w_{kj} = \sum_{c=1}^{k} z_{ic} w_{cj} = (w^j)^T z_i$$

  - K-means: take index ‘j’ of closest mean.
  - PCA: use z_i to weight index ‘j’ of all “means” (factors)
- We can write approximation of the vector x_i as:

$$\underset{d \times 1}{\hat{x}_i} = \begin{bmatrix} (w^1)^T z_i \\ (w^2)^T z_i \\ \vdots \\ (w^d)^T z_i \end{bmatrix} = \underset{d \times k}{W^T} \underset{k \times 1}{z_i}$$

Important Stuff

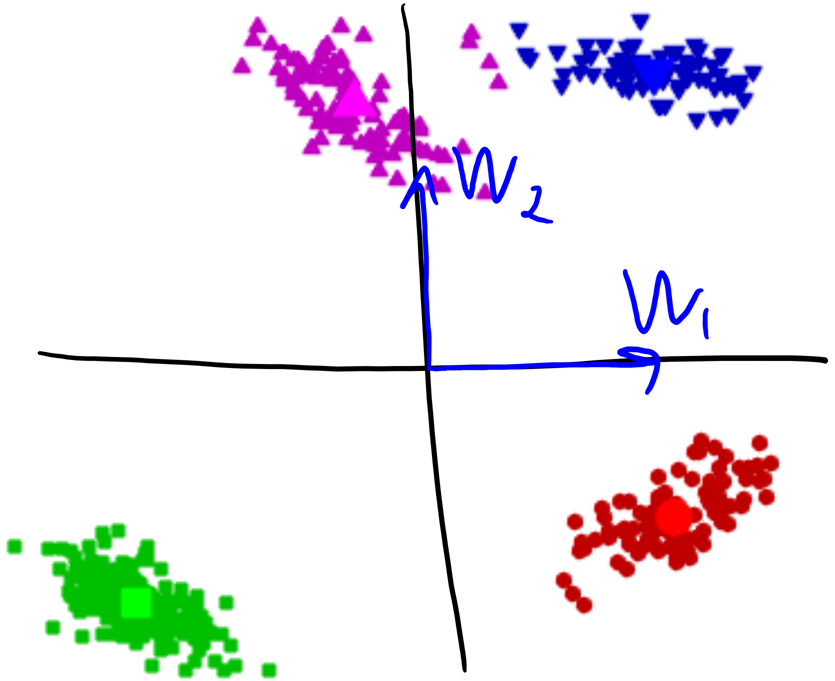
- PCA is also called a "matrix factorization" model:

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

- Punch line: PCA learns a k-dimensional subspace of the original d-dimensional space
  - The subspace is represented by k basis vectors
  - The basis vectors are the rows of W
  - The representations in the new basis are the rows of Z

Digression: PCA only makes sense for k < d

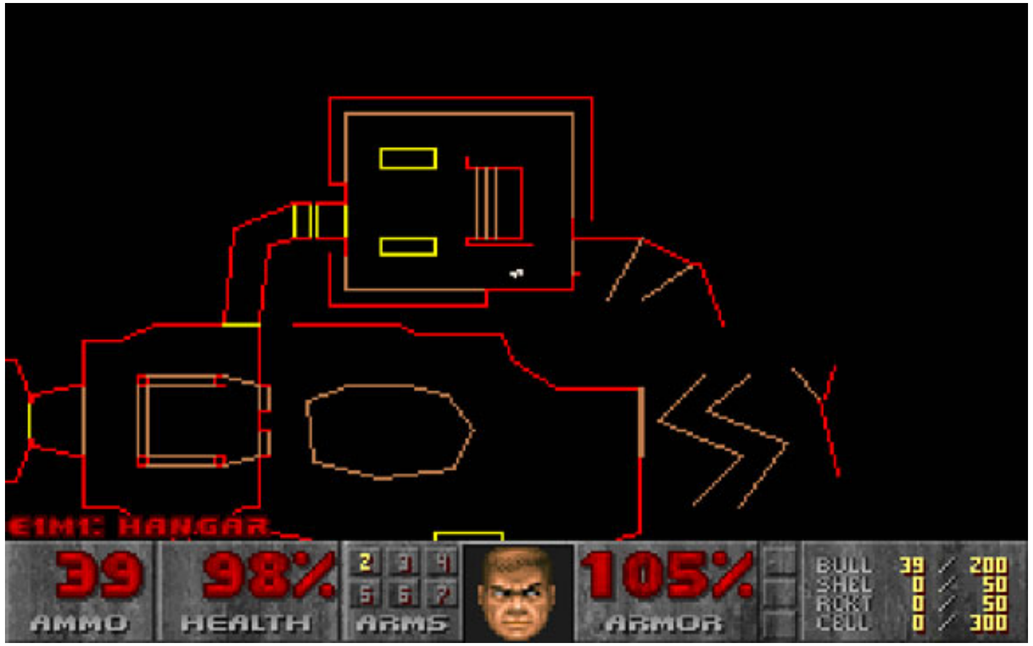
- Remember our clustering dataset with 4 clusters:



- It doesn’t make sense to use PCA with k=4 on this dataset.
  - We only need two vectors [1 0] and [0 1] to exactly represent all 2d points.

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.

https://en.wikipedia.org/wiki/Doom_(1993_video_game)
https://forum.minetest.net/viewtopic.php?f=5&t=9666

Overhead Map and Latent-Factor Models

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

$$x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \end{bmatrix} \begin{matrix} \leftarrow \text{"x" coordinate} \\ \leftarrow \text{"y" coordinate} \\ \leftarrow \text{"z" coordinate} \end{matrix}$$

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

$$z_i = \begin{bmatrix} z_{i1} \\ z_{i2} \end{bmatrix} \begin{matrix} \leftarrow \text{"x" coordinate} \\ \leftarrow \text{"y" coordinate} \end{matrix}$$

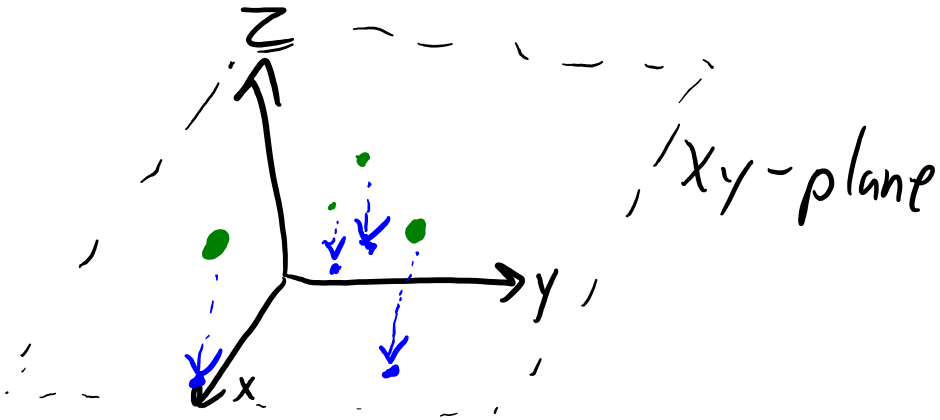
- Our k=2 latent factors (basis vectors) are the following:

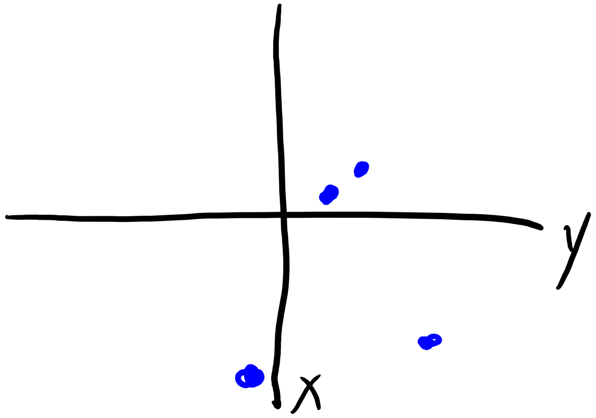
$$W = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

- So our approximation of x_i is: $\hat{x}_i = z_{i1}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + z_{i2}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

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.

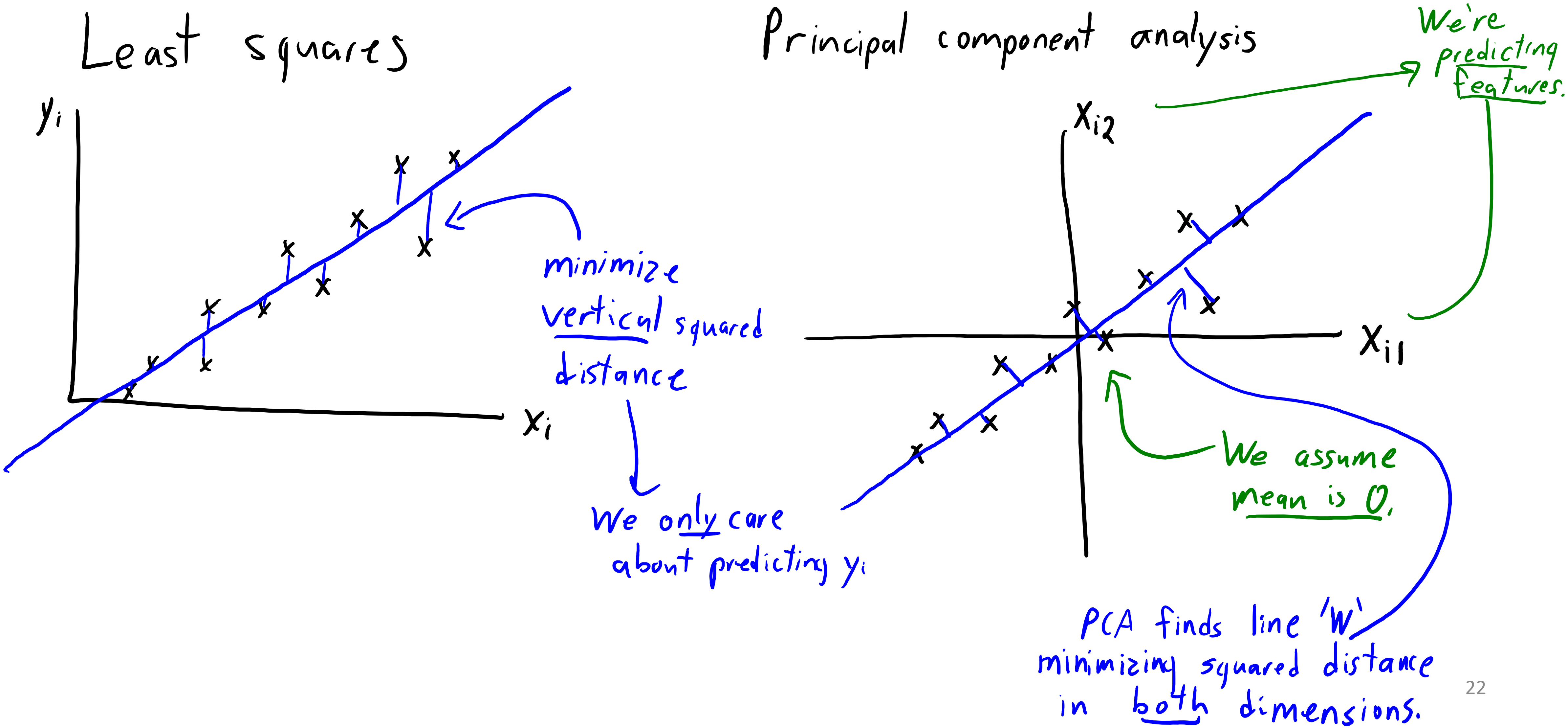
Overhead Map and Latent-Factor Models

- 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.

www.momtastic.com/webecoist/2010/11/07/some-fine-dam-climbing-goats-scaling-steep-vertical-wall

Least squares vs. PCA



Least squares vs. PCA

- Least squares learns a d-dimensional hyperplane
  - We think of this as living inside a d+1 dimensional space
    - The d features, plus the target
  - The goal is to input d values and output 1 value
  - This is supervised learning
- PCA learns a k-dimensional hyperplane for any integer $0<k<d$
  - When d=2 then it must be that k=1
  - The goal is to input d values and output k values
  - This is unsupervised learning

PCA applications

- Supervised learning: we could use ‘Z’ as our inputs.
- Outlier detection: it might be an outlier if isn’t a combination of new features.
- Dimension reduction: compress data into limited number dimensions.
- Visualization: if we have only 2 dimensions, we can view data as a scatterplot.
- Interpretation: we can try and figure out what the new features represent.

Summary

- Latent-factor models:
  - Try to learn factors W from training examples X.
  - Usually, the z_i are coefficients for factors w_c.
  - Useful for dimensionality reduction, visualization, factor discovery, etc.
- Principal component analysis:
  - We can view ‘W’ as best lower-dimensional hyper-plane.
  - We can view ‘Z’ as the coordinates in the lower-dimensional hyper-plane.
  - We haven’t completely specified PCA yet – will finish next class.

Motivation: Human vs. Machine Perception

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

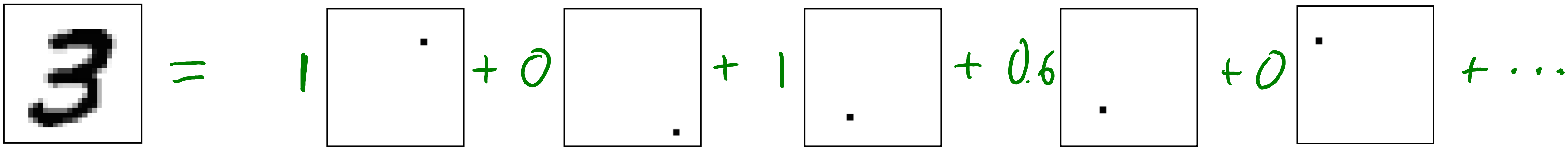
What we see:

What the computer "sees":

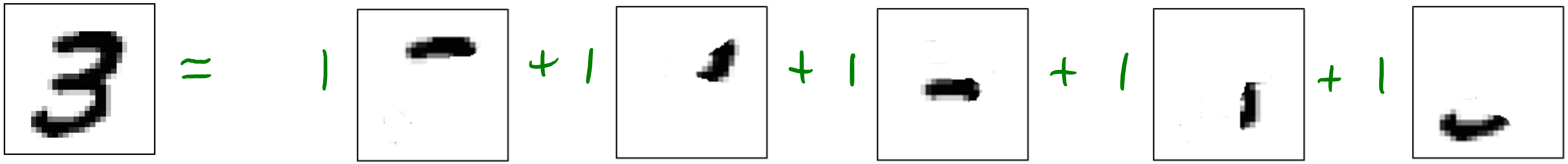
- But maybe images shouldn't be written as combinations of pixels.

Motivation: Pixels vs. Parts

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



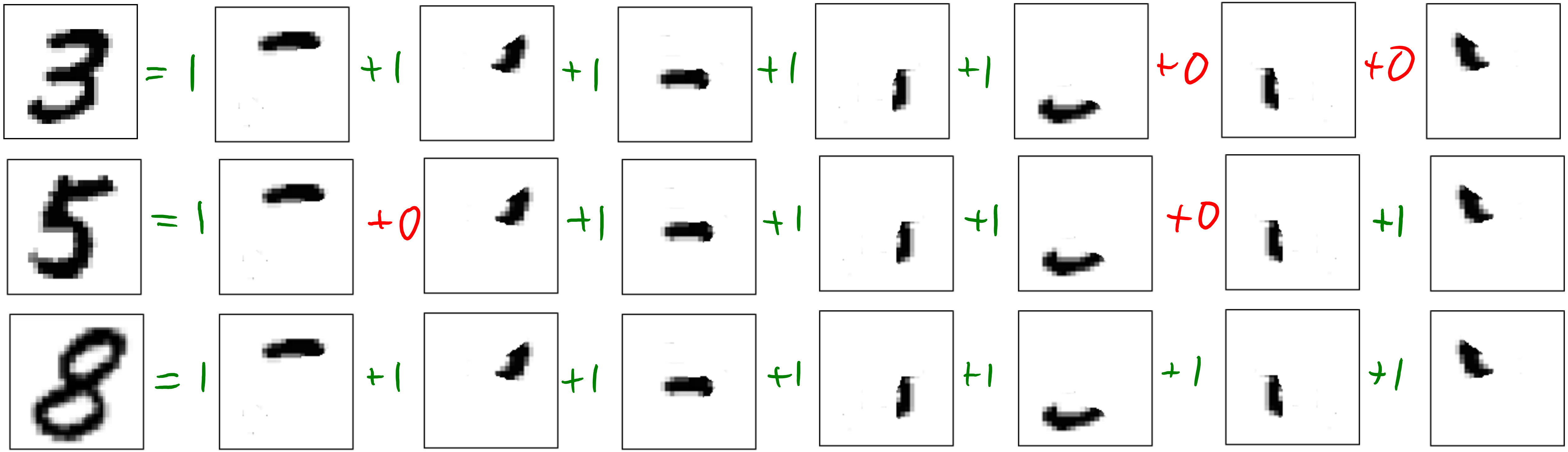
  - We have one image for each pixel.
  - The weights specify “how much of this pixel is in the image”.
    - A weight of zero means that pixel is white, a weight of 1 means it’s black.
- This is non-intuitive, isn’t a “3” made of small number of “parts”?



  - 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.

Principal Component Analysis (PCA) Applications

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

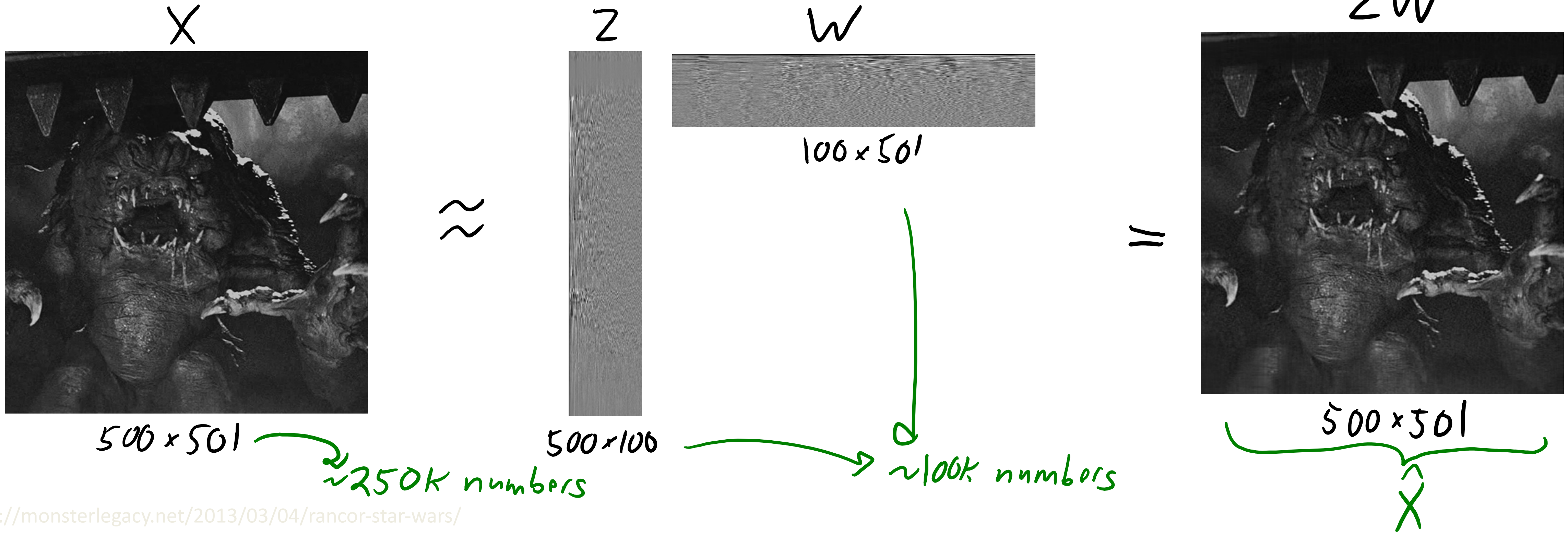
PCA was invented in 1901 by Karl Pearson,[1] as an analogue of the principal axis theorem in mechanics; it was later independently developed (and named) by Harold Hotelling in the 1930s.[2] Depending on the field of application, it is also named the discrete Kosambi-Karhunen–Loève transform (KLT) in signal processing, the Hotelling transform in multivariate quality control, proper orthogonal decomposition (POD) in mechanical engineering, singular value decomposition (SVD) of **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 th orthogonal direction. The vectors shown are the eigenvectors of the covariance matrix scaled by the squa root of the corresponding eigenvalue, and shifted so their tails are at the mean.

https://en.wikipedia.org/wiki/Principal_component_analysis

PCA Applications

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



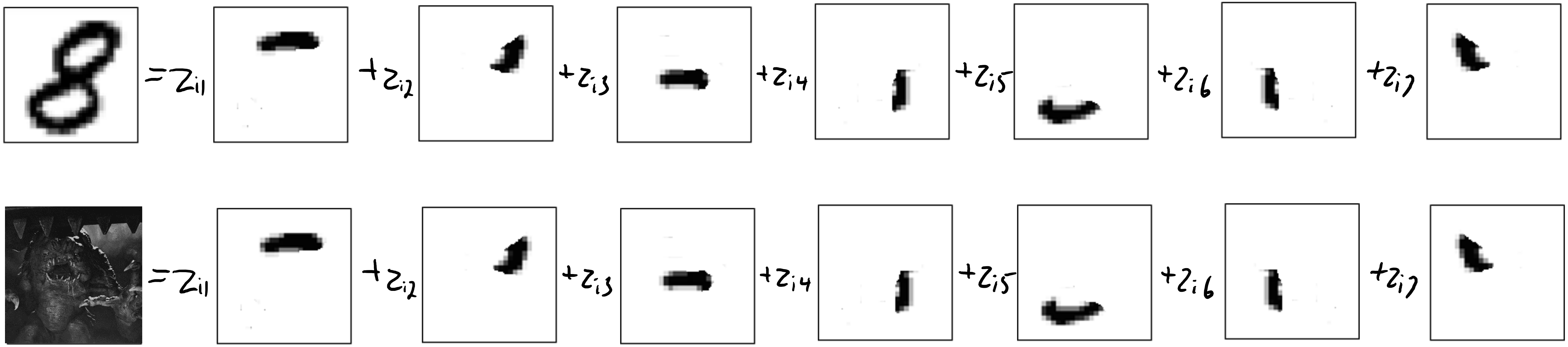
https://monsterlegacy.net/2013/03/04/rancor-star-wars/

PCA Applications

- Applications of PCA:
  - Dimensionality reduction: replace ‘X’ with lower-dimensional ‘Z’.
    - If k << 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.



Example Application: Supervised Learning

- 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:

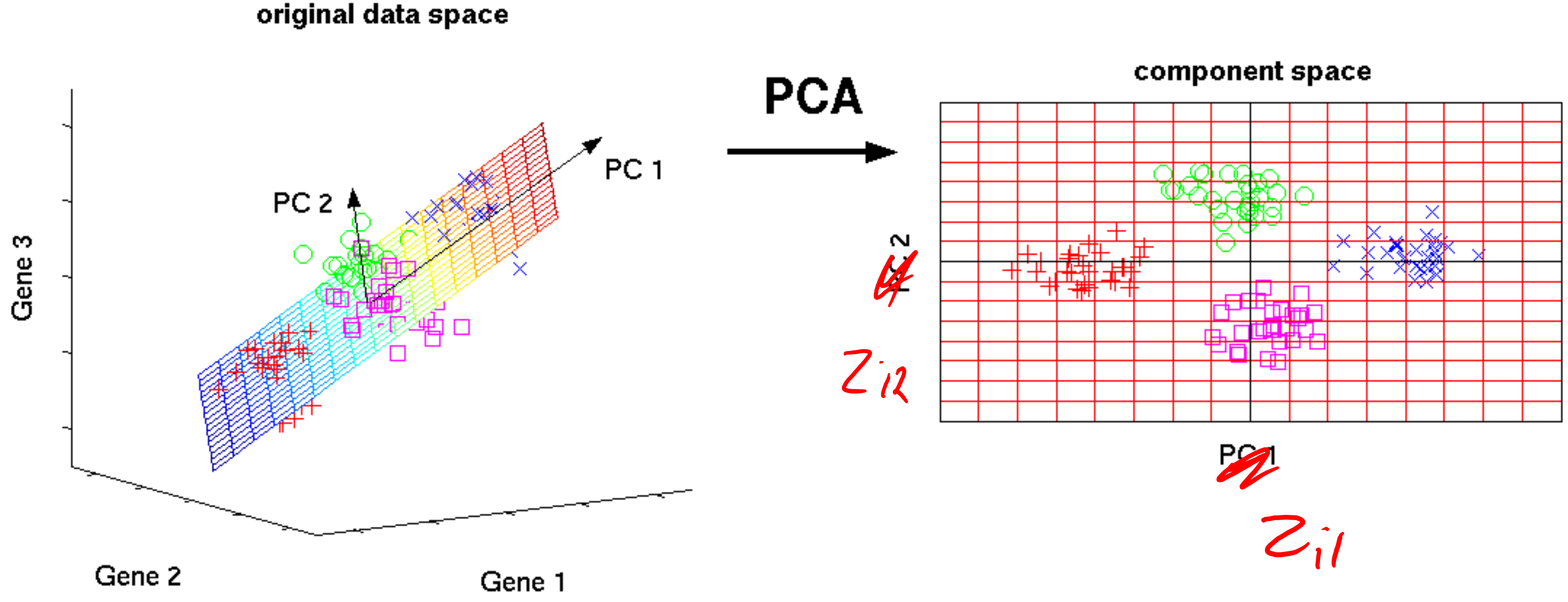
$$y_i = v^T z_i$$

linear regression weights 'v' trained under this change of basis.

lower-dimensional than original features so less overfitting

PCA with d=3 and k=2.

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

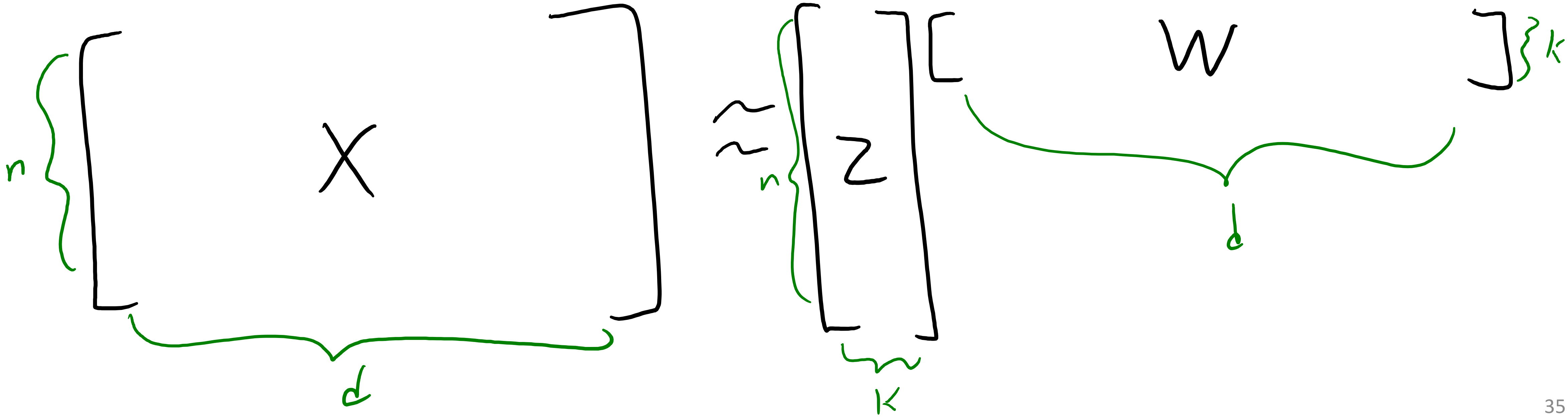


- With d=3, PCA (k=1) finds line minimizing squared distance to x_i.

http://www.nlpca.org/fig_pca_principal_component_analysis.png

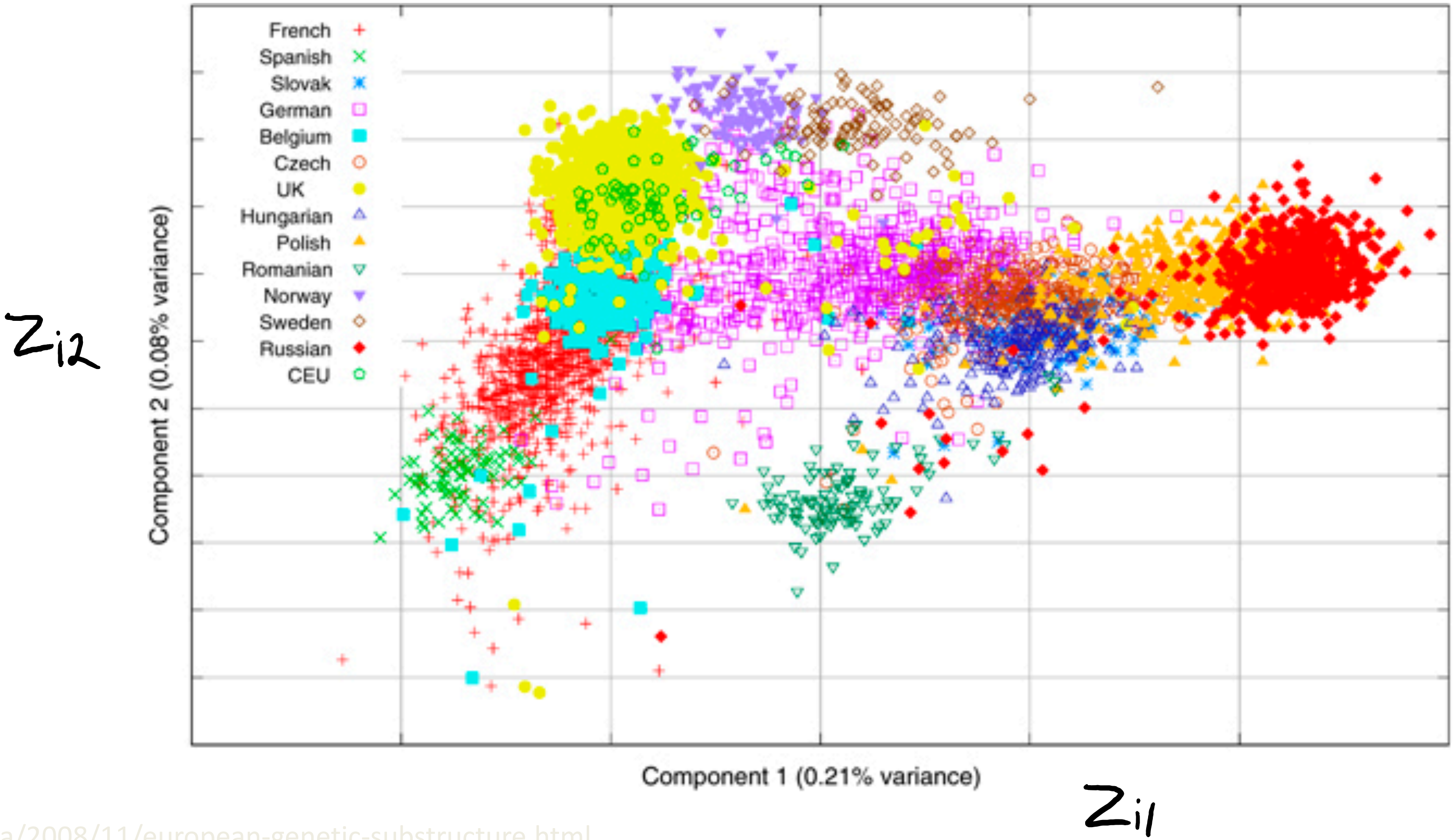
PCA Applications

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



PCA Applications

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



http://infoproc.blogspot.ca/2008/11/european-genetic-substructure.html

PCA Applications

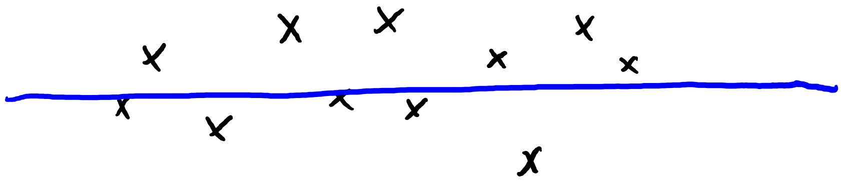
- Applications of PCA:
  - Data interpretation: we can try to assign meaning to latent factors w_c.
    - Hidden "factors" that influence all the variables.

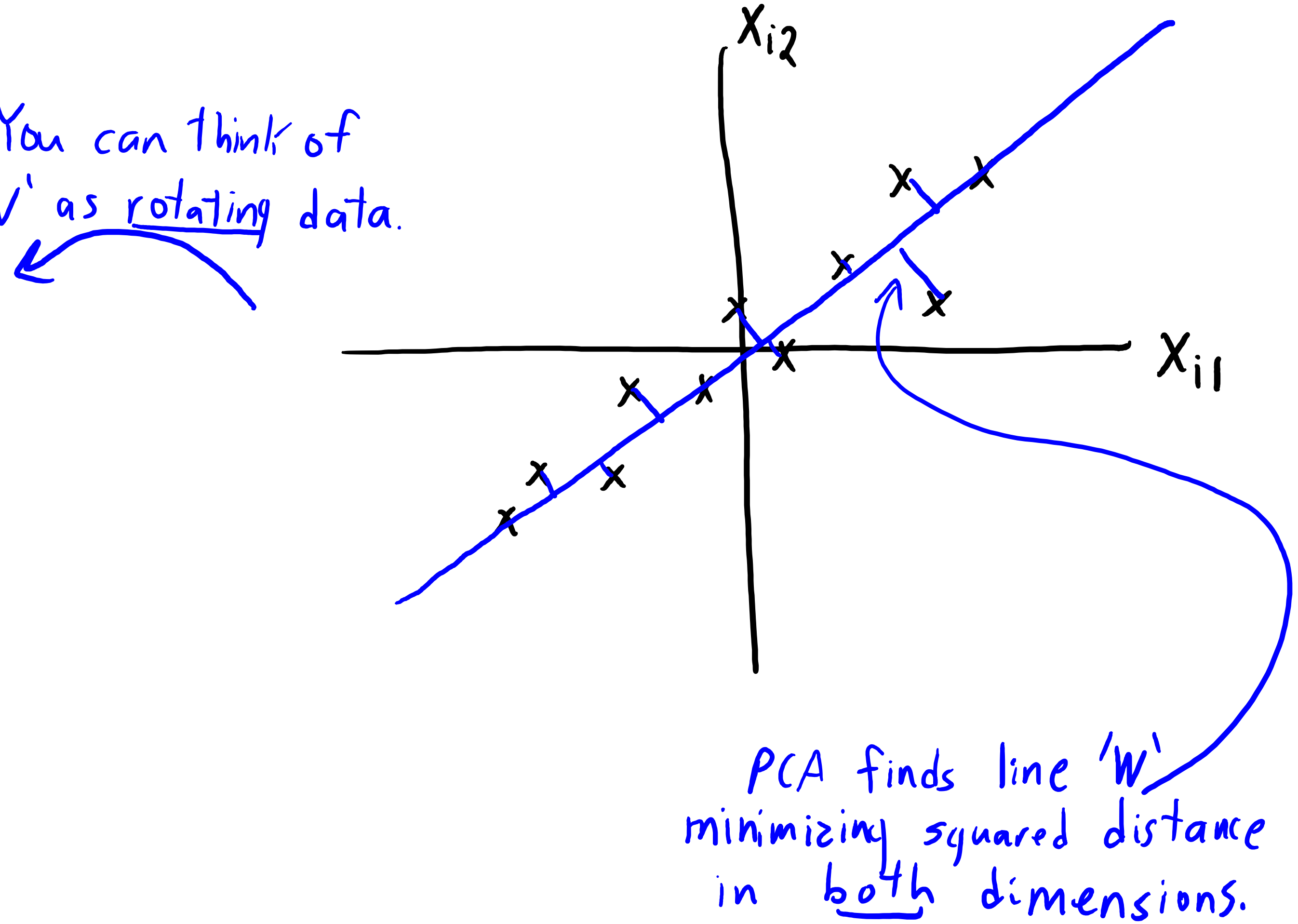
Trait	Description
Openness	Being curious, original, intellectual, creative, and open to new ideas.
Conscientiousness	Being organized, systematic, punctual, achievement-oriented, and dependable.
Extraversion	Being outgoing, talkative, sociable, and enjoying social situations.
Agreeableness	Being affable, tolerant, sensitive, trusting, kind, and warm.
Neuroticism	Being anxious, irritable, temperamental, and moody.

https://new.edu/resources/big-5-personality-traits

PCA with d=2 and k =1

Principal component analysis

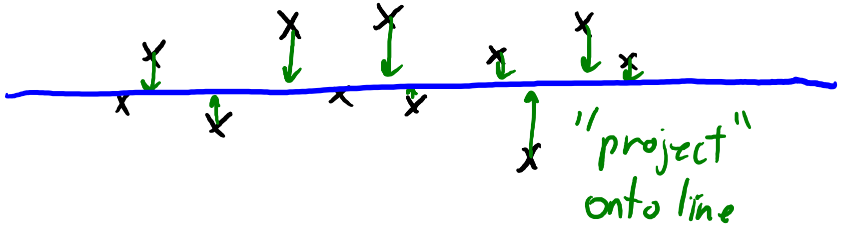


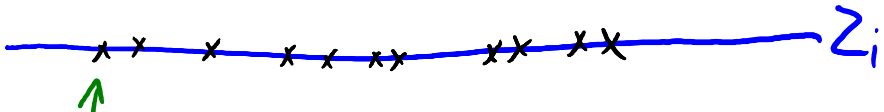


PCA with d=2 and k =1

Principal component analysis

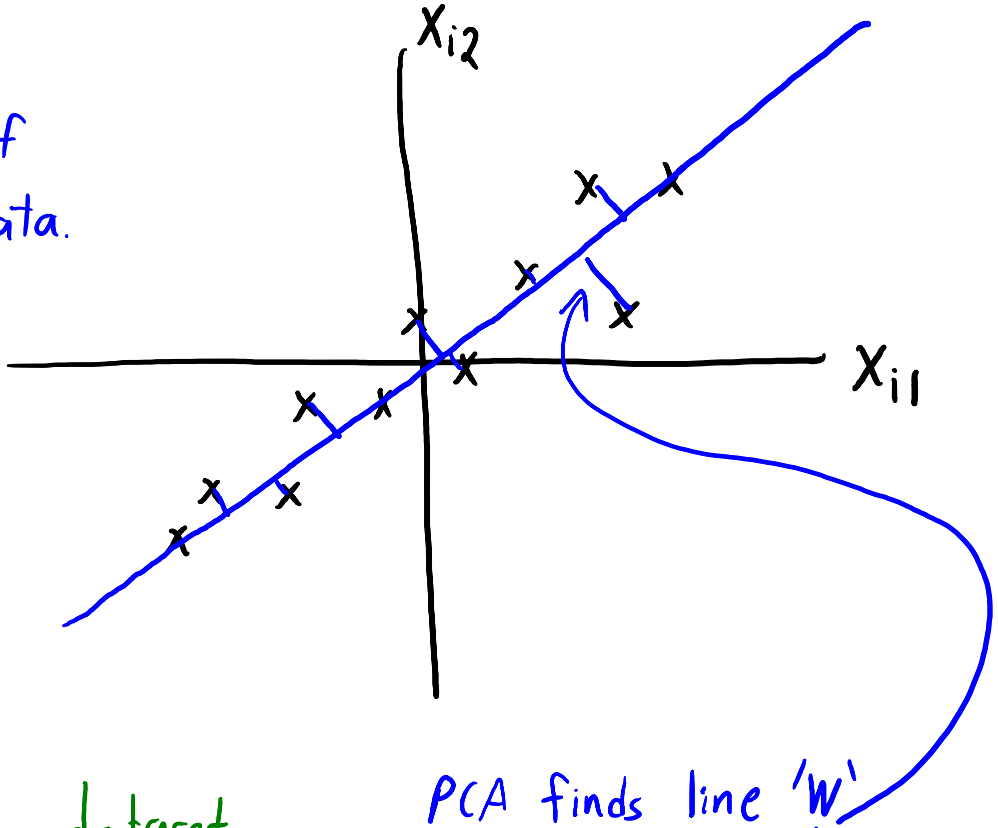
You can think of 'W' as rotating data.





z_i can be interpreted as position along the line.

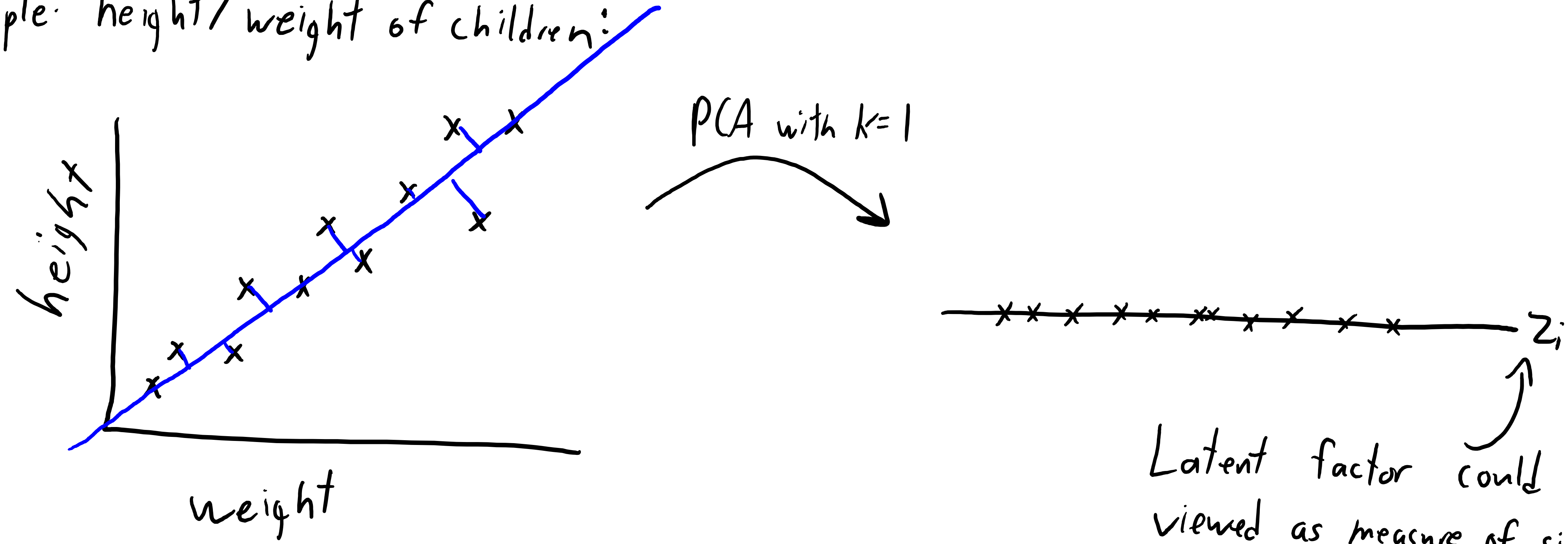
(turned a 2d dataset into a 1d dataset)



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

PCA with d=2 and k =1

Example: height/weight of children:



Latent factor could be viewed as measure of size.