

CPSC 340: Machine Learning and Data Mining

Principal Component Analysis (PCA)

Admin

- Ugrad events
- Looking for summer TAs for CPSC 340

Last Time: MAP Estimation

- MAP estimation maximizes posterior:

$$\underset{\text{"posterior"}}{p(w | X, y)} \propto \underset{\text{"likelihood"}}{p(y | X, w)} \underset{\text{"prior"}}{p(w)}$$

- 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 prior, 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.

Multi-Class Classification

- For **binary classification** with linear models we use:

$$y_i = \text{sign}(w^T x_i)$$

- For **multi-class classification** with linear models we use:

$$y_i = \underset{c}{\operatorname{argmax}} \{ w_c^T x_i \}$$

– Where we have a vector w_c for each class 'c'.

$$W = \begin{bmatrix} | & | & \dots & | \\ w_1 & w_2 & \dots & w_K \\ | & | & \dots & | \end{bmatrix}$$

- To jointly estimate the w_c , we can use **softmax likelihood**:

$$p(y_i = c | x_i, W) = \frac{\exp(w_c^T x_i)}{\sum_{c'=1}^K \exp(w_{c'}^T x_i)}$$

Multi-Class Classification

- For **multi-class classification** with linear models we use:

$$y_i = \operatorname{argmax}_c \{ w_c^T x_i \}$$

- To jointly estimate the w_c , we can use **softmax likelihood**:

$$p(y_i | x_i, w) = \frac{\exp(w_{y_i}^T x_i)}{\sum_{c=1}^K \exp(w_c^T x_i)}$$

- By taking the negative log and adding a **regularizer**, we get:

$$f(w) = \sum_{i=1}^n \left[-w_{y_i}^T x_i + \log \left(\sum_{c=1}^K \exp(w_c^T x_i) \right) \right] + \frac{\lambda}{2} \sum_{c=1}^K \sum_{j=1}^d w_{cj}^2$$

Tries to
make $w_c^T x_i$ big for
the correct label

Approximates $\max_c \{ w_c^T x_i \}$
so tries to make $w_c^T x_i$ small
for incorrect labels

Usual L_2 -regularizer
on elements of 'w'

Digression: Frobenius Matrix Norm

We can write $\sum_{i=1}^n \sum_{j=1}^d w_{ij}^2$ in matrix notation as $\|W\|_F^2$

The notation $\|W\|_F$ is the "Frobenius" norm of matrix W :

$$\|W\|_F = \sqrt{\sum_{i=1}^n \sum_{j=1}^d w_{ij}^2}$$

(L_2 -norm if we "stack" columns of 'W' into a big vector)

End of Part 3: Key Concepts

- **Linear models** base predictions on linear combinations 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.

- Popular 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 methods are **hypothesis testing** and **search and score**.
 - **L1-regularization** simultaneously regularizes and selects features.

The Story So Far...

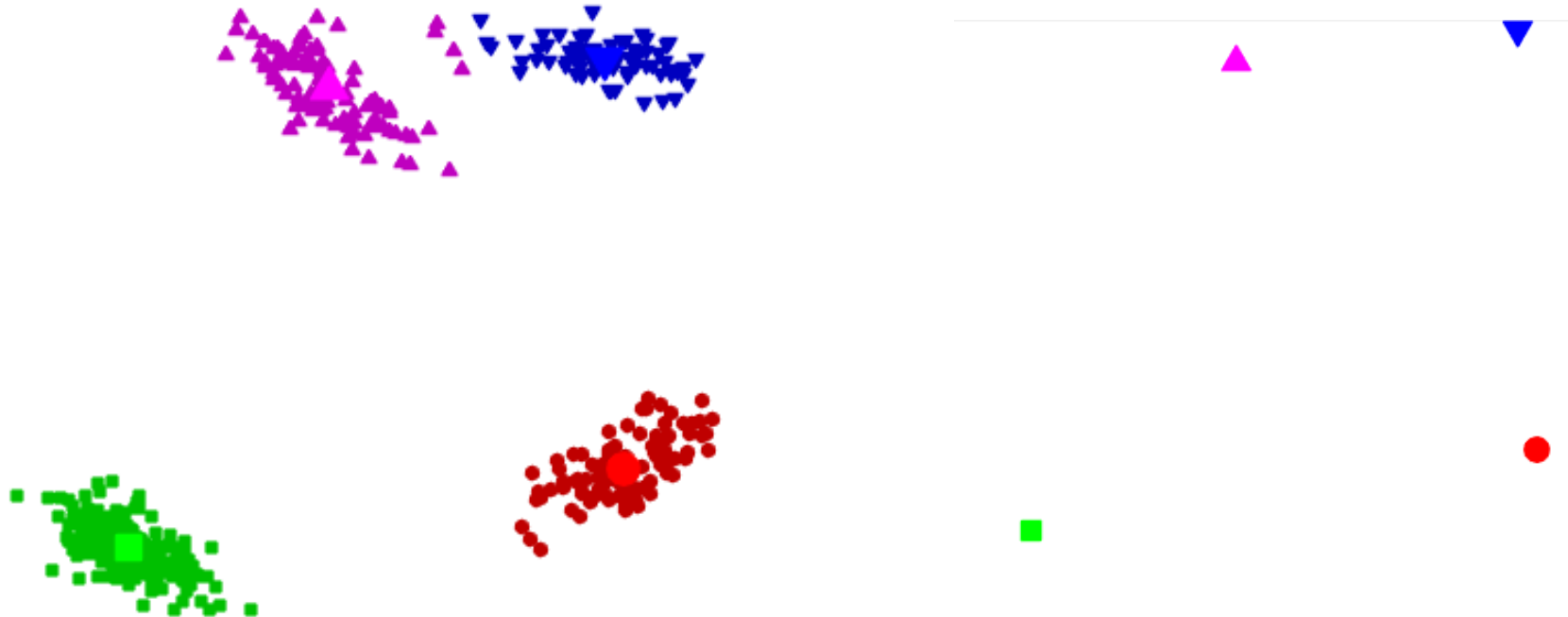
- Supervised Learning Part 1:
 - Methods based on counting and distances.
- Unsupervised Learning Part 1:
 - Methods based on counting and distances.
- Supervised Learning Part 2 (just finished):
 - Methods based on linear models and gradient descent.
- Unsupervised Learning Part 2 (starting today):
 - Methods based on linear models and gradient descent.

Unsupervised Learning Part 2

- Unsupervised learning:
 - We **only have x_i values**, but no explicit target labels.
 - You want to do ‘something’ with them.
- Some unsupervised learning tasks:
 - Clustering: What types of x_i are there?
 - Outlier detection: Is this a ‘normal’ x_i ?
 - Association rules: Which x_{ij} occur together?
 - Latent-factors: What ‘parts’ are the x_i made from?
 - Data visualization: What does the high-dimensional X look like?
 - Ranking: Which are the most important x_i ?

Motivation: Vector Quantization

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



Motivation: Vector Quantization

- We can write **vector quantization as a linear model**:
 - Define ' z_i ' as a **binary vector** that is zero except in position c_i .

If $k=4$ and $c_i=3$ then $z_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$

- Our weird notation for **mean matrix 'W'**:

$$W = \begin{matrix} k \times d \\ \left[\begin{array}{c} \text{---} w_{c_i} \text{---} \\ \text{---} w_{c_i} \text{---} \\ \vdots \\ \text{---} w_{c_i} \text{---} \end{array} \right] = \left[\begin{array}{c|c|c|c} w_1 & w_2 & \dots & w_d \\ \hline | & | & & | \end{array} \right] \end{matrix}$$

Each row is a mean. Each column is feature ' j ' for each mean.

Weird notation alert:

- w_{c_i} is row c_i of W
- w_j is column j of W

So $w_{c_i} = \begin{bmatrix} w_1^T z_i \\ w_2^T z_i \\ \vdots \\ w_d^T z_i \end{bmatrix} = W^T z_i$ So vector quantization uses $x_{ij} \approx w_j^T z_i$ and $x_i \approx W^T z_i$

Regression View of K-Means

- Recall that we said **k-means minimizes the objective**:

$$f(W, c) = \sum_{i=1}^n \sum_{j=1}^d (w_{c_{ij}} - x_{ij})^2$$

- In our new notation, we can write k-means as minimizing:

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

where $Z = \begin{bmatrix} - & z_1^T & - \\ - & z_2^T & - \\ & \vdots & \\ - & z_n^T & - \end{bmatrix}$

Each row has
1 non-zero

- 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 trying to **learn the basis z_i** .
 - Here **the outputs are the inputs** – so they are d-dimensional not 1-dimensional
 - Hence the extra sum as compared to the regular least squares loss
- This is an important slide – let's take our time here.

Principal Component Analysis (PCA)

- Principal component analysis (PCA) minimizes the same objective:

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

- But instead of “1 of k” binary z_i we allow a continuous basis z_i .
- Called a latent-factor model:
 - Instead of means, w_c called “factors” or “principal components”.
 - The z_i are called “factor loadings” or “low-dimensional basis”.
 - The z_i say how to mix the means/factors to approximate example ‘i’.
 - We don’t just approximate x by one of the means
 - We approximate it as a linear combination of all means/factors
 - This is like clustering with soft assignments to the cluster means

Principal Component Analysis (PCA)

- Principal component analysis (PCA) in matrix notation:

$$\begin{aligned} f(W, Z) &= \sum_{i=1}^n \sum_{j=1}^d (w_j^T z_i - x_{ij})^2 \\ &= \sum_{i=1}^n \sum_{j=1}^d (w_{j1} z_{i1} + w_{j2} z_{i2} + \dots + w_{jd} z_{id} - x_{ij})^2 \\ &= \sum_{i=1}^n \|W^T z_i - x_i\|^2 \\ &= \|ZW - X\|_F^2 \end{aligned}$$

- Also called a **matrix factorization** model: $\overset{n \times d}{X} \approx \overset{n \times k}{Z} \overset{k \times d}{W}$

PCA Applications

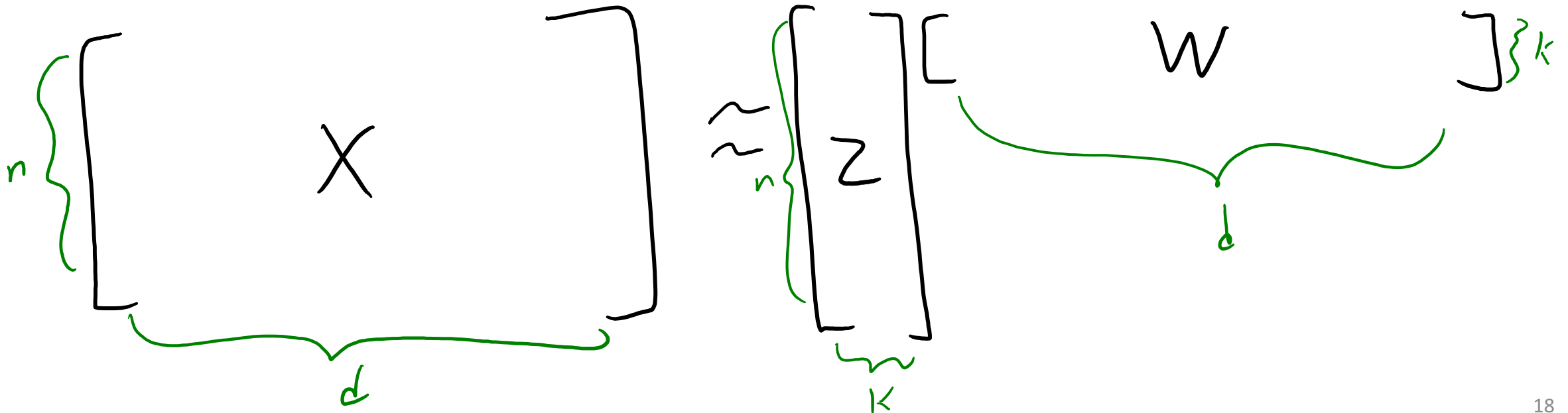
- PCA has been reinvented many times:

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), [quasi-harmonic 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.

PCA Applications

- Applications of PCA:
 - **Dimensionality reduction**: replace 'X' with lower-dimensional 'Z'.
 - If $k \ll d$, then compresses data.
 - Much 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.
 - Much better approximation than vector quantization.
 - **Outlier detection**: if PCA gives poor approximation of x_i , could be 'outlier'.
 - Though due to squared error PCA is sensitive to outliers.
 - **Partial least squares**: uses **PCA features as basis** for linear model.

Compute approximation $X \approx ZW$

Now Z as features in a linear model:

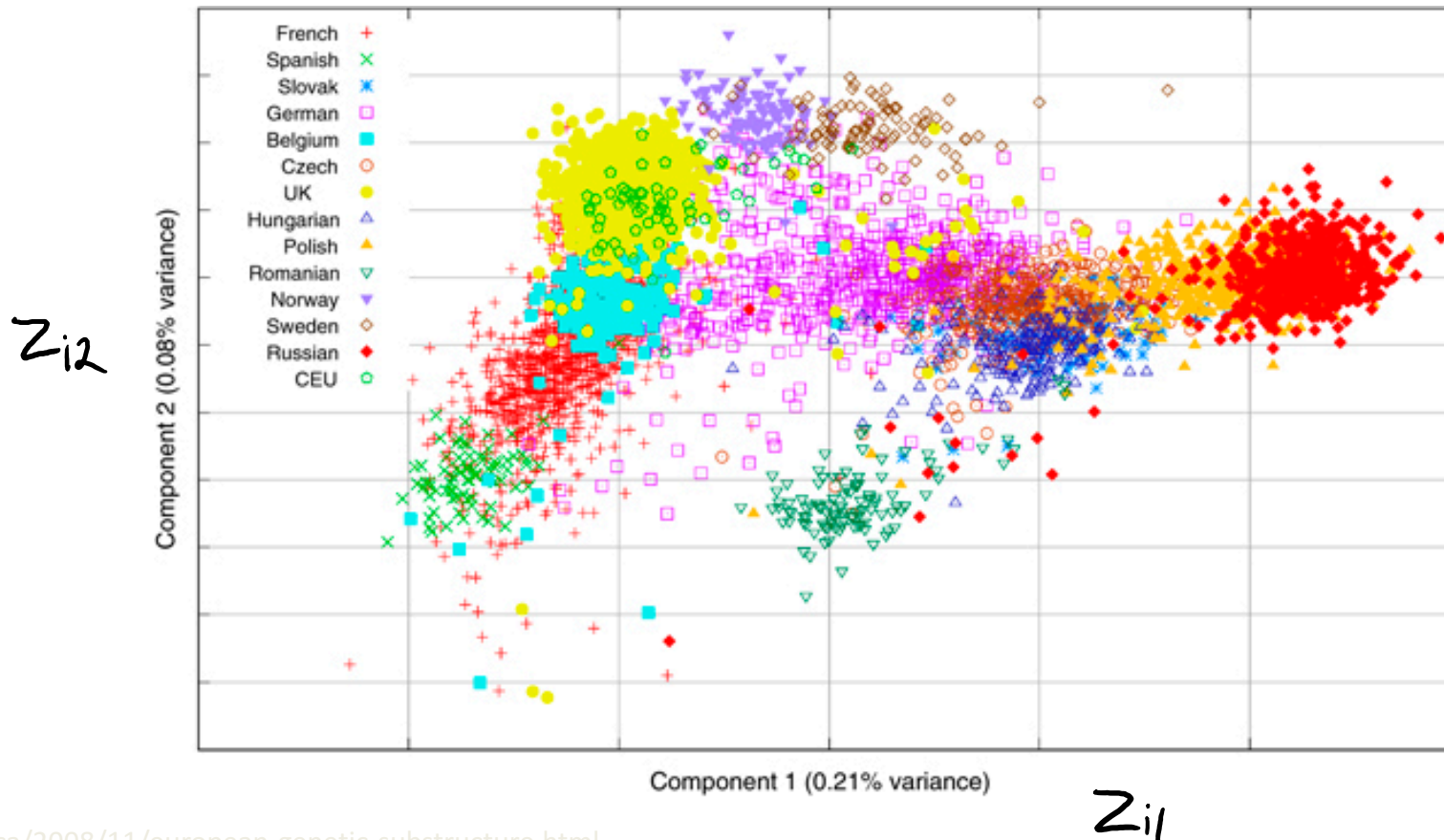
$$y_i = w^T z_i$$

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.

PCA with $d=1$

- Consider the case of PCA when $d=1$:

$$X = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix}_{n \times 1}$$

$$Z = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix}_{n \times 1}$$

$$W = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}_{1 \times 1}$$

PCA objective:

$$f(Z, W) = \sum_{i=1}^n (W Z_i - x_i)^2$$

- There is an obvious solution: $w = 1$ and $Z = X$.
 - PCA is only interesting when $k < d$, since otherwise we can set $Z = X$.
- PCA is not unique: $w = 1/\alpha$ and $z_i = \alpha x_i$ for any $\alpha \neq 0$ is a solution.
 - $(1/\alpha) * (\alpha x_i) = x_i$, so this achieves an error of 0 for non-zero α .
 - We can enforce $|w| = 1$ to avoid this problem.

Summary

- Latent-factor models:
 - Compress data as linear combination of ‘factors’.
 - Useful for dimensionality reduction, visualization, factor discovery.
- Principal component analysis:
 - Most common variant based on squared reconstruction error.
- Next time: face detection in images.