CPSC 340: Machine Learning and Data Mining

Recommender Systems

Fall 2016

As students, we all probably thought abut the following question: “how much money can I expect to make when I graduate?” To answer that question, Microsoft and DataSense challenge you to **build a machine learning algorithm** which "learns" patterns from real world Canadian Census data to predict how much money Canadians make. Compete for awesome prizes and be judged by Data Science professionals (including a Microsoft Recruiter!)

For more information, visit our facebook event page at **https://goo.gl/eRNfGB**

Admin

- Assignment 4:
  - 1 late date to hand in Wednesday, 2 for Friday, 3 for Monday.
- Office hours:
  - Tuesday 2pm office hours moved to ICICS 104.
- Assignment 5:
  - Out tonight/tomorrow (2-3 questions).
- Assignment 6:
  - Out by weekend (2 questions).
- Final:
  - December 12

Last 3 Lectures: Latent-Factor Models

- We've been discussing latent-factor models of the form:

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

- We get different models with under different conditions:
  - K-means: each z_i has one '1' and the rest are zero.
  - Least squares: we only have one variable (d=1) and the z_i are fixed.
  - PCA: the columns w_c have a norm of 1 and have an inner product of zero.
  - NMF: all elements of W and Z are non-negative.

Last Time: Variations on Latent-Factor Models

- We can use all our tricks for linear regression in this context:

$$f(W,Z) = \sum_{i=1}^{n}\sum_{j=1}^{d} |w_j^T z_i - x_{ij}| + \frac{\lambda_1}{2}\sum_{i=1}^{n}\sum_{c=1}^{k} z_{ic}^2 + \frac{\lambda_2}{2}\sum_{j=1}^{d}\sum_{c=1}^{k} |w_{cj}|$$

- Absolute loss gives robust PCA that is less sensitive to outliers.
- We can use L2-regularization.
  - Though only reduces overfitting if we regularize both 'W' and 'Z'.
- We can use L1-regularization to give sparse latent factors/features.
- We can use logistic/softmax/Poisson losses for discrete x_{ij}.
- Can use change of basis to learn non-linear latent-factor models.

Recommender System Motivation: Netflix Prize

- Netflix Prize:
  - 100M ratings from 0.5M users on 18k movies.
  - Grand prize was $1M for first team to reduce squared error by 10%.
  - Started on October 2nd, 2006.
  - Netflix's system was first beat October 8th.
  - 1% error reduction achieved on October 15th.
  - Steady improvement after that.
    - ML methods soon dominated.
  - One obstacle was 'Napolean Dynamite' problem:
    - Some movie ratings seem very difficult to predict.
    - Should only be recommended to certain groups.

Lessons Learned from Netflix Prize

- Prize awarded in 2009:
  - Ensemble method that averaged 107 models.
  - Increasing diversity of models more important than improving models.

- Winning entry (and most entries) used collaborative filtering:
  - Method that only looks at ratings, not features of movies/users.
- A simple collaborative filtering method that does really well:
  - Regularized matrix factorization. Now adopted by many companies.

http://bits.blogs.nytimes.com/2009/09/21/netflix-awards-1-million-prize-and-starts-a-new-contest/?_r=0

Motivation: Other Recommender Systems

- Recommender systems are now everywhere:
  - Music, news, books, jokes, experts, restaurants, friends, dates, etc.
- Main types approaches:
  1. Content-based filtering.
     - Supervised learning:
       - Extract features x_i of users and items, building model to predict rating y_i given x_i.
       - Apply model to prediction for new users/items.
     - Example: G-mail's "important messages" (personalization with "local" features).
  2. Collaborative filtering.
     - "Unsupervised" learning (but have label matrix 'Y' but no features):
       - We only have labels y_{ij} (rating of user 'i' for movie 'j').
     - Example: Amazon recommendation algorithm.

Collaborative Filtering Problem

- Collaborative filtering is 'filling in' the user-item matrix:

$$Y = \text{user}\left\{\underbrace{\begin{bmatrix} ? & 4 & 3 & 2 & 3 & 3 \\ 2 & 1 & ? & 5 & ? & 5 \\ ? & 1 & ? & 5 & 5 & 5 \\ \vdots & & \vdots & & & \\ 2 & 3 & 3 & ? & ? & ? \end{bmatrix}}_{\text{movie}}\right.$$

"Green Lantern"

"Ryan Reynolds"

- We have some ratings available with values {1,2,3,4,5}.
- We want to predict ratings "?" by looking at available ratings.

Collaborative Filtering Problem

- Collaborative filtering is 'filling in' the user-item matrix:

$$Y = \text{user}\left\{ \underbrace{\begin{bmatrix} ? & 4 & 3 & 2 & 3 & 3 \\ 2 & 1 & ? & 5 & ? & 5 \\ ? & 1 & ? & 5 & 5 & 5 \\ 2 & 3 & 3 & ? & ? & ? \end{bmatrix}}_{\text{movie}} \right.$$

"Green Lantern"

"Ryan Reynolds"

Similar user

Similar movie

- What rating would "Ryan Reynolds" give to "Green Lantern"?
  – Why is this not completely crazy? We may have similar users and movies.

Matrix Factorization for Collaborative Filtering

- Our standard latent-factor model for entries in matrix 'Y':

$$y_{ij} \approx w_j^T z_i$$

- User 'i' has latent features z_i.

z_{ic} could mean "likes romantic comedies"

- Movie 'j' has latent features w_j.

w_{jc} could mean "has Nicholas Cage"

- Our loss functions sums over available ratings 'R':

$$f(Z,W) = \sum_{(i,j)\in R} (w_j^T z_i - y_{ij})^2 + \frac{\lambda_1}{2}\|Z\|_F^2 + \frac{\lambda_2}{2}\|W\|_F^2$$

- And we add L2-regularization to both types of features.

Adding Global/User/Movie Biases

- Our standard latent-factor model for entries in matrix 'Y':

$$y_{ij} \approx w_j^T z_i$$

- Sometimes we don't assume the y_{ij} have a mean of zero:
  - We could add bias β reflecting average overall rating:

$$y_{ij} \approx \beta + w_j^T z_i$$

  - We could also add a user-specific bias β_i and item-specific bias β_j.

$$y_{ij} \approx \beta + \beta_i + \beta_j + w_j^T z_i$$

    - Some users rate things higher on average, and movies are rated better on average.
    - These might also be regularized.

Beyond Accuracy in Recommender Systems

- Winning system of Netflix Challenge was never adopted.
- Other issues important in recommender systems:
  - Diversity: how different are the recommendations?
    - If you like 'Battle of Five Armies Extended Edition', recommend Battle of Five Armies?
    - Even if you really really like Star Wars, you might want non-Star-Wars suggestions.
  - Persistence: how long should recommendations last?
    - If you keep not clicking on 'Hunger Games', should it remain a recommendation?
  - Trust: tell user *why* you made a recommendation.
  - Social recommendation: what did your friends watch?
  - Freshness: people tend to get more excited about *new/surprising* things.
    - Collaborative filtering does not predict well for new users/movies.

Hybrid Approaches

- Collaborative filtering can't predict ratings for new users/movies.
- Hybrid approaches combine content-based/collaborative filtering:
  - SVDfeature (won "KDD Cup" in 2011 and 2012)

Extra factors we learn for specific users and movies.

$$y_{ij} \approx \beta + \beta_i + \beta_j + w^T x_{ij} + w_j^T z_i$$

Average rating across all users/movies

Average rating for user 'i'

Average for movie 'j'

Linear model based on user/movie features x_{ij}.

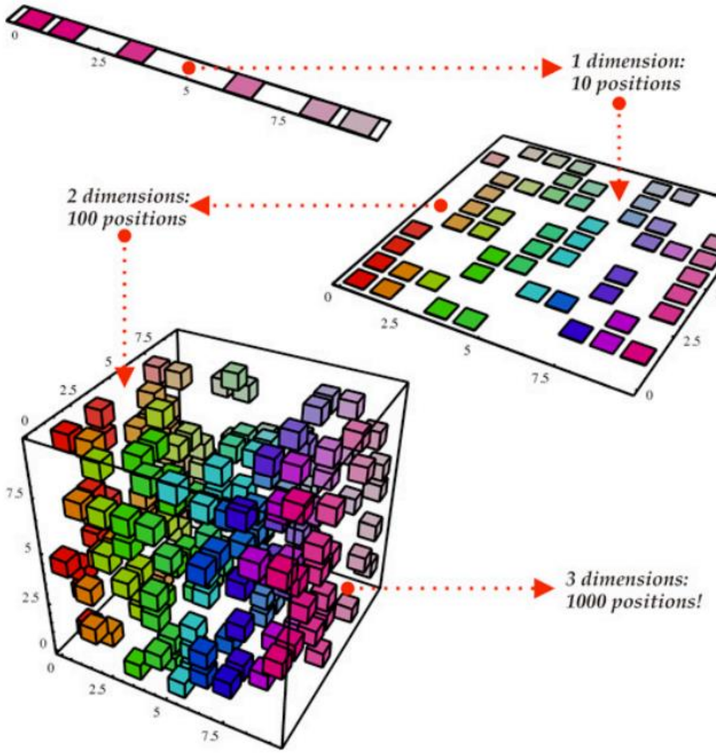
Latent features z_i for user 'i' and latent features w_j for movie 'j'

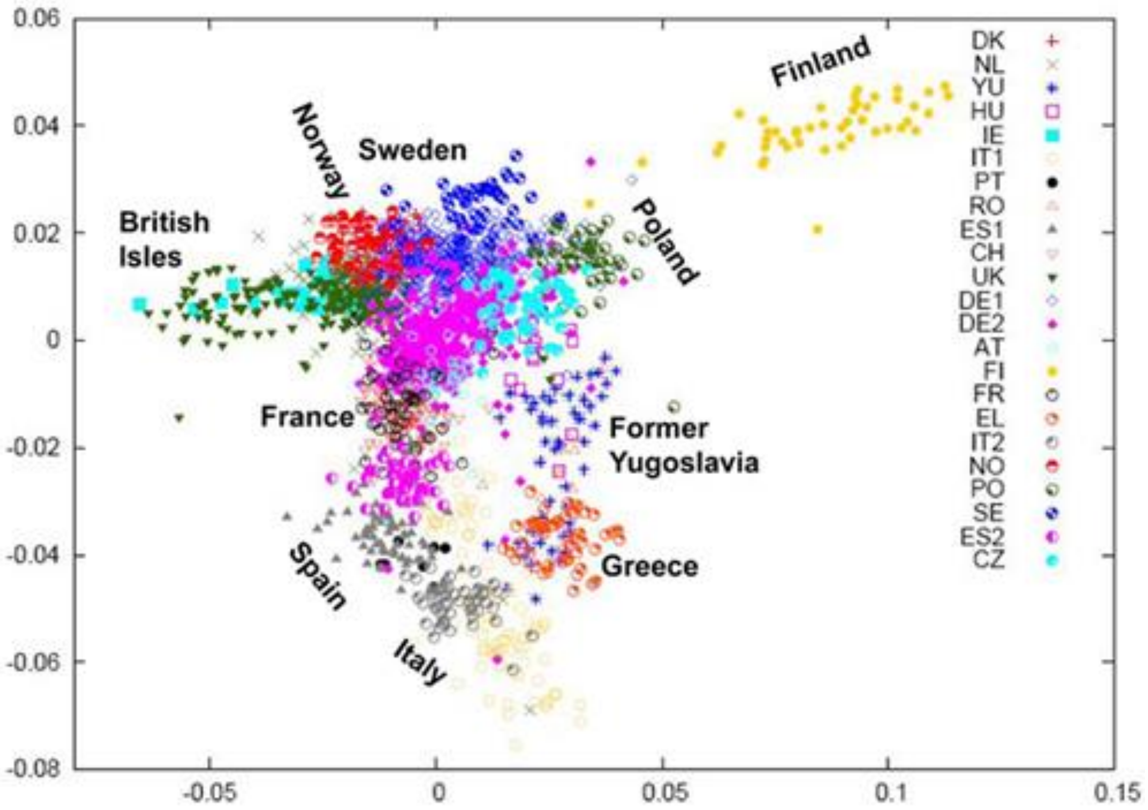
Standard supervised learning: can predict for new users/movies

(pause)

Latent-Factor Models for Visualization

- PCA takes features x_i and gives k-dimensional approximation z_i.
- If k is small, we can use this to visualize high-dimensional data.

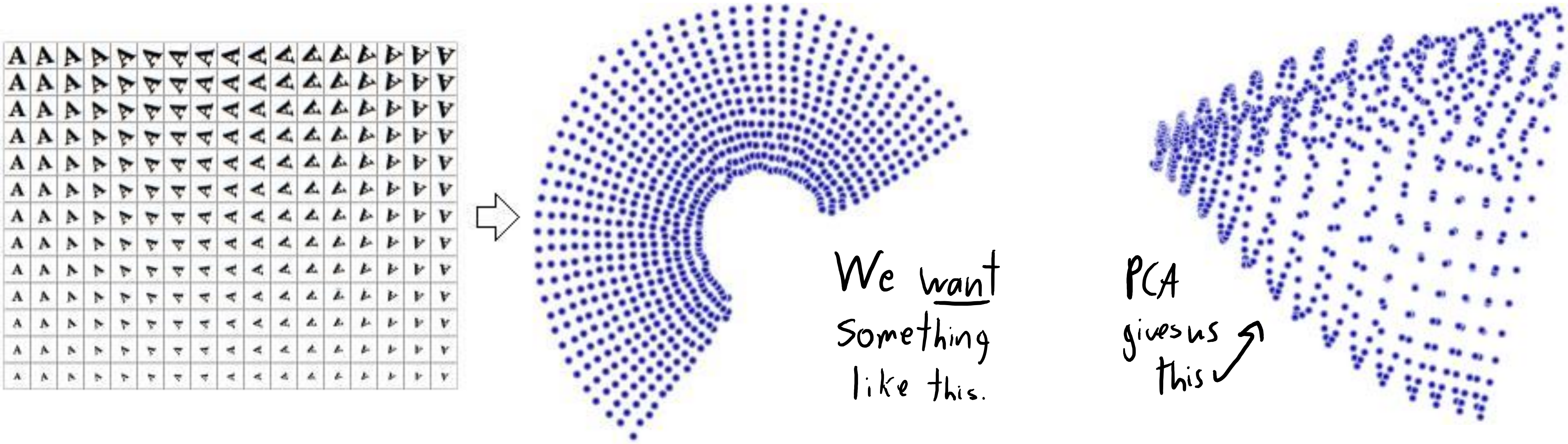




http://www.turingfinance.com/artificial-intelligence-and-statistics-principal-component-analysis-and-self-organizing-maps/
http://scienceblogs.com/gnxp/2008/08/14/the-genetic-map-of-europe/

Motivation for Non-Linear Latent-Factor Models

- But PCA is a parametric linear model
- PCA may not find obvious low-dimensional structure.



- We could use change of basis or kernels: but still need to pick basis.

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

Multi-Dimensional Scaling

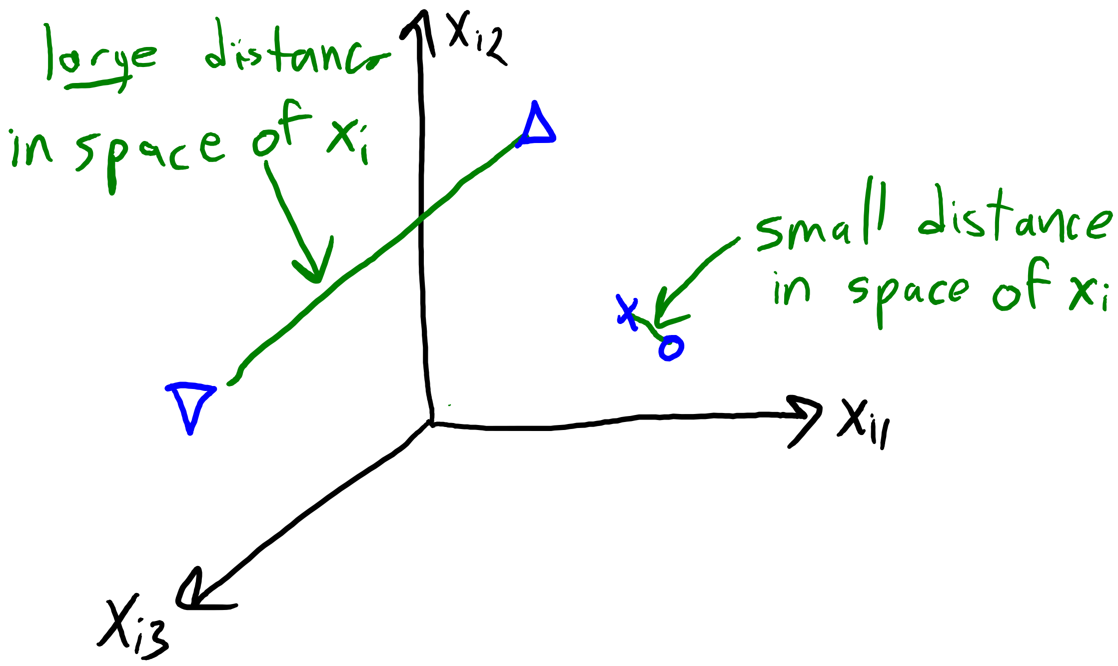
- PCA for visualization:
  - We're using PCA to get the location of the z_i values.
  - We then plot the z_i values as locations in a scatterplot.
- Multi-dimensional scaling (MDS) is a crazy idea:
  - Let's directly optimize the locations of the z_i values.
    - "Gradient descent on the points in a scatterplot".
  - Needs a"cost" function saying how "good" the z_i locations are.
    - Classic MDS cost function:

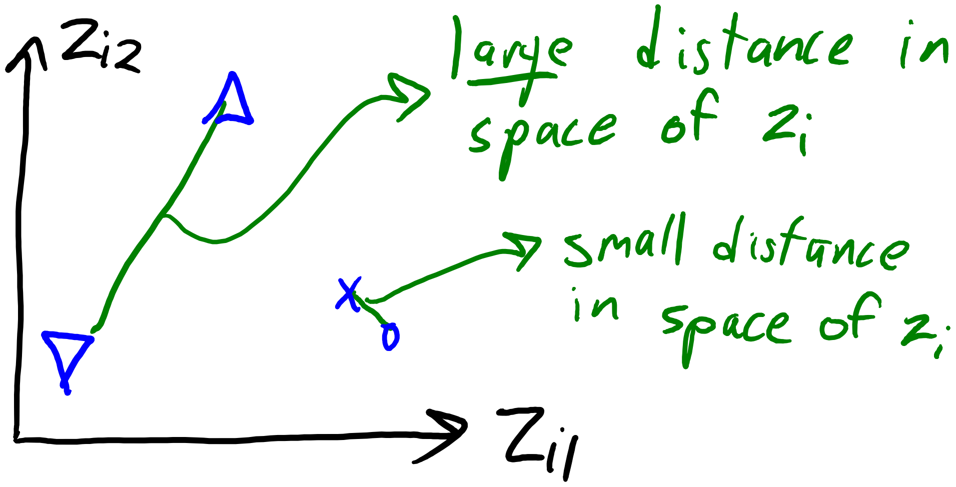
$$f(Z) = \sum_{i=1}^{n}\sum_{j=i+1}^{n}\left(\underbrace{\|z_i - z_j\|}_{\text{Distance between points in 'k' dimensions}} - \underbrace{\|x_i - x_j\|}_{\text{Distance between points in original 'd' dimensions}}\right)^2$$

Multi-Dimensional Scaling

- Multi-dimensional scaling (MDS):
  - Directly optimize the final locations of the z_i values.

$$f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left(\|z_i - z_j\| - \|x_i - x_j\| \right)^2$$



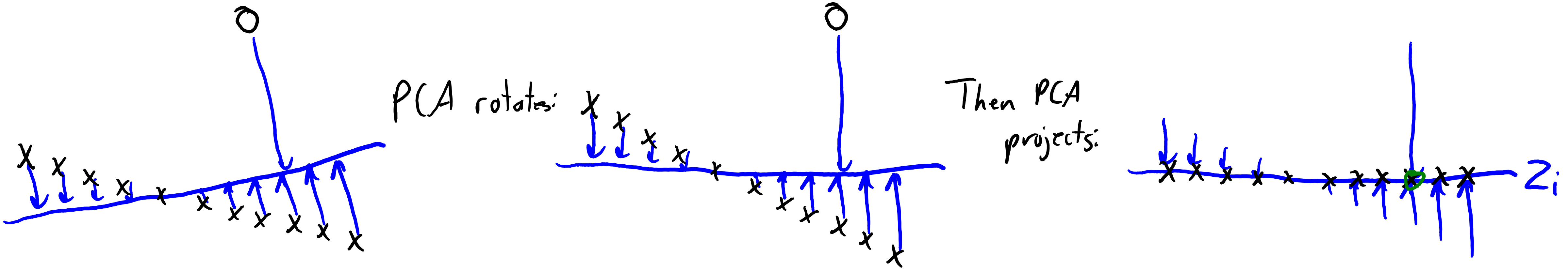


Multi-Dimensional Scaling

- Multi-dimensional scaling (MDS):
  - Directly optimize the final locations of the z_i values.

$$f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left(\|z_i - z_j\| - \|x_i - x_j\| \right)^2$$

  - Non-parametric dimensionality reduction and visualization:
    - No 'W': just trying to make z_i preserve high-dimensional "distances" between x_i.

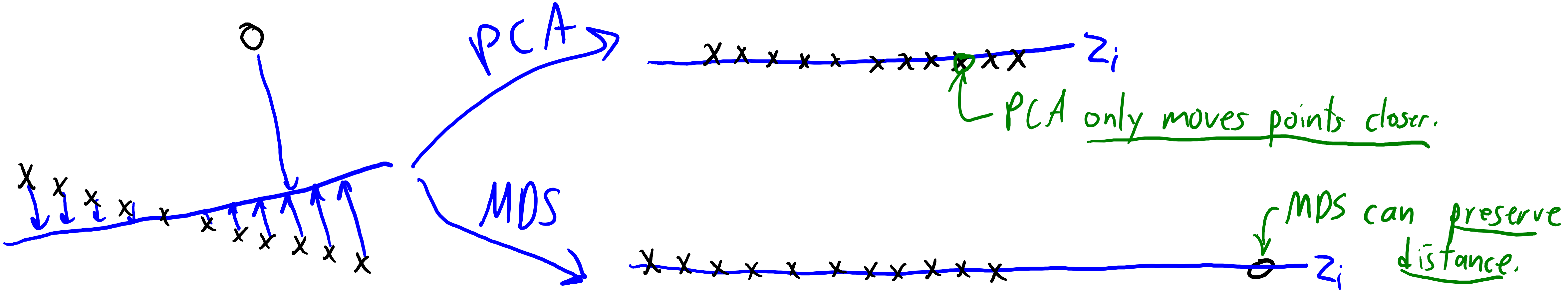


Multi-Dimensional Scaling

- Multi-dimensional scaling (MDS):
  - Directly optimize the final locations of the z_i values.

$$f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left(\|z_i - z_j\| - \|x_i - x_j\| \right)^2$$

  - Non-parametric dimensionality reduction and visualization:
    - No 'W': just trying to make z_i preserve high-dimensional distances between x_i.

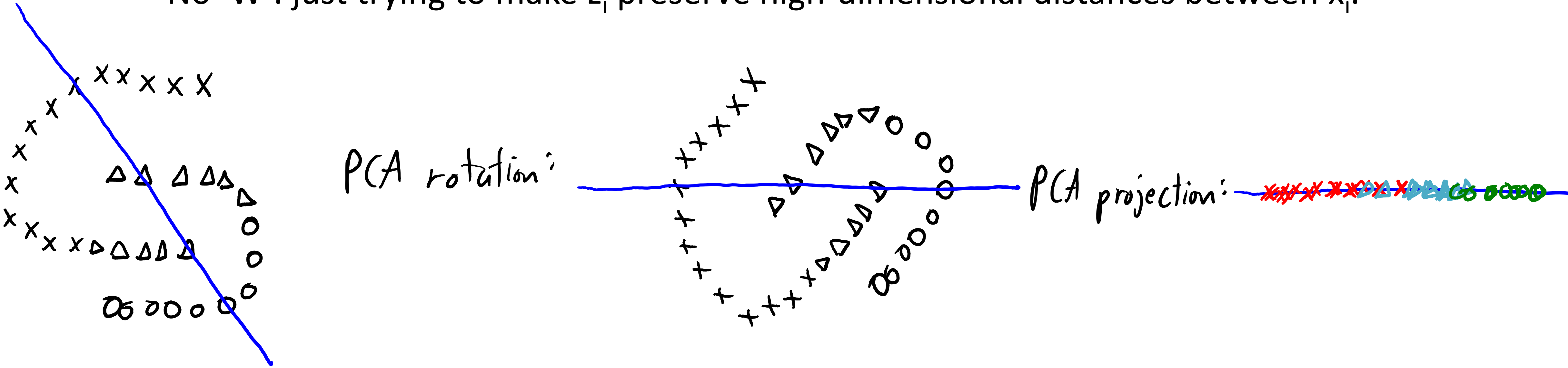


Multi-Dimensional Scaling

- Multi-dimensional scaling (MDS):
  - Directly optimize the final locations of the z_i values.

$$f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left(\|z_i - z_j\| - \|x_i - x_j\| \right)^2$$

  - Non-parametric dimensionality reduction and visualization:
    - No 'W': just trying to make z_i preserve high-dimensional distances between x_i.

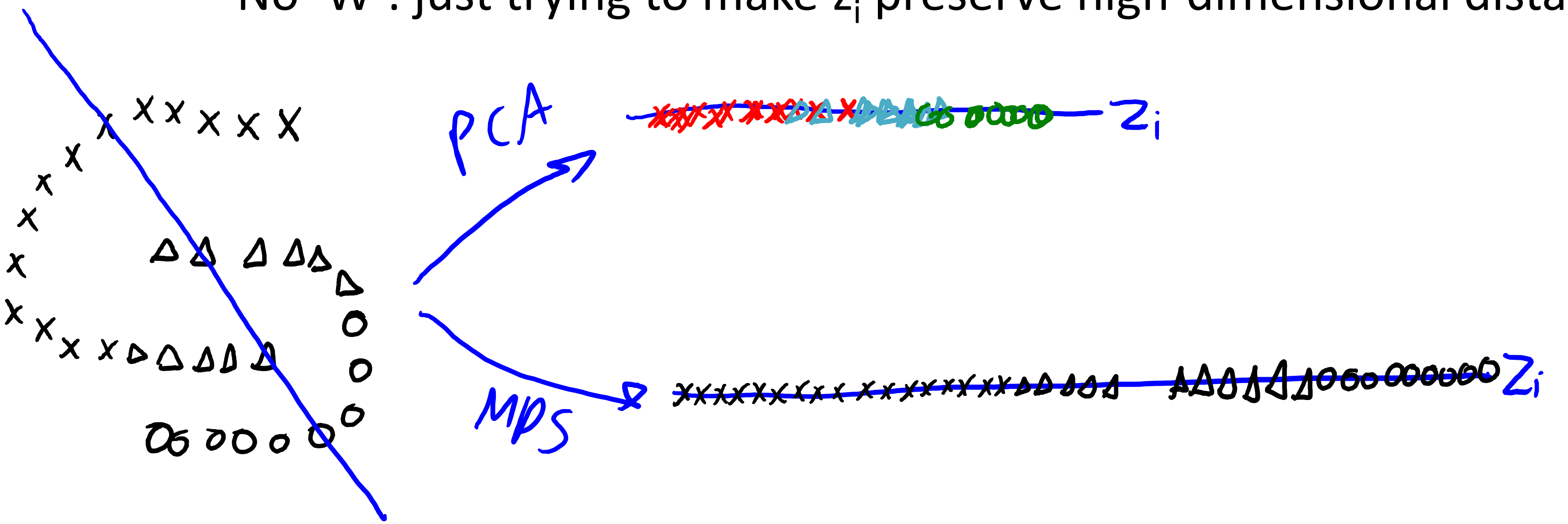


Multi-Dimensional Scaling

- Multi-dimensional scaling (MDS):
  - Directly optimize the final locations of the z_i values.

$$f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left(\|z_i - z_j\| - \|x_i - x_j\| \right)^2$$

  - Non-parametric dimensionality reduction and visualization:
    - No 'W': just trying to make z_i preserve high-dimensional distances between x_i.



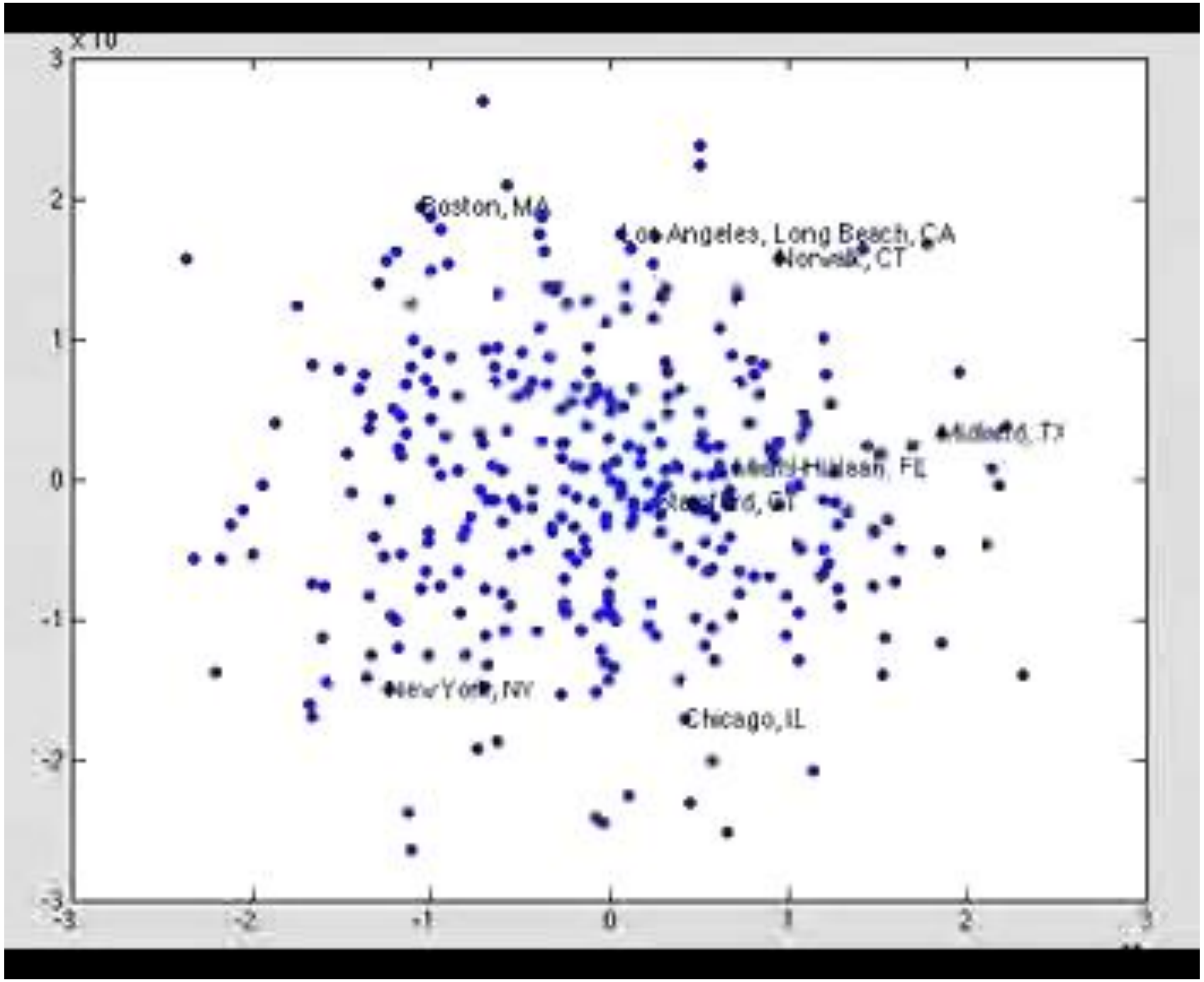
MDS Optimization

- Multi-dimensional scaling (MDS):
  - Directly optimize the final locations of the z_i values.

$$f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left(\|z_i - z_j\| - \|x_i - x_j\| \right)^2$$

- Cannot use SVD to compute solution:
  - Gradient descent on z_i values.
  - You "learn" a scatterplot that tries to visualize high-dimensional data.
  - But not convex and sensitive to initialization.

MDS Method ("Sammon Mapping") in Action



Different MDS Cost Functions

- MDS default objective: squared difference of Euclidean norm:

$$f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left(\|z_i - z_j\| - \|x_i - x_j\| \right)^2$$

- But we can make z_i match different distances/similarities:

$$f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} d_3(d_2(z_i, z_j) - d_1(x_i, x_j))$$

  – Where the functions are not necessarily the same:
    - d_1 is the high-dimensional distance we want to match.
    - d_2 is the low-dimensional distance we can control.
    - d_3 controls how we compare high-/low-dimensional distances.

Classic Multi-Dimensional Scaling (MDS)

- MDS default objective function with general distances/similarities:

$$f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} d_3(d_2(z_i, z_j) - d_1(x_i, x_j))$$

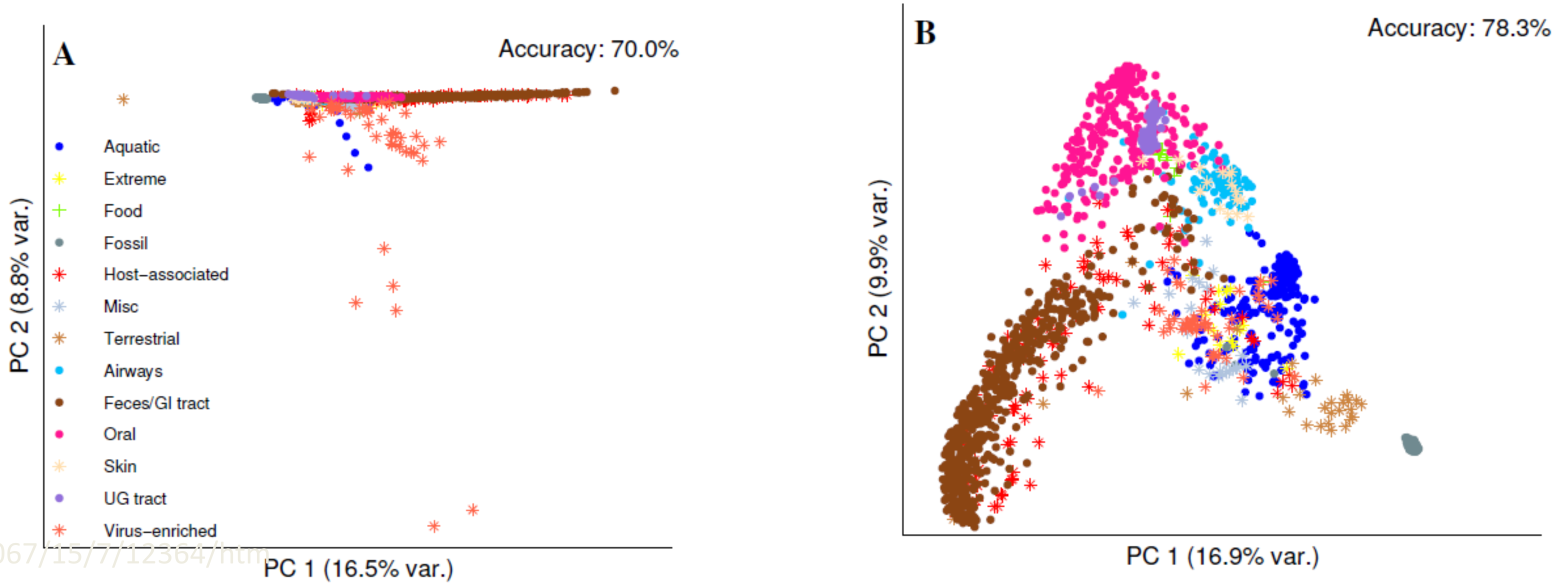
- "Classic" MDS uses $d_1(x_i,x_j) = x_i^T x_j$ and $d_2(z_i,z_j) = z_i^T z_j$.
  - We obtain PCA in this special case (for centered x_i).
  - Not a great choice because it's a linear model.

Non-Euclidean Multi-Dimensional Scaling (MDS)

- MDS default objective function with general distances/similarities:

$$f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} d_3(d_2(z_i, z_j) - d_1(x_i, x_j))$$

- Another possibility: $d_1(x_i,x_j) = ||x_i - x_j||_1$ and $d_2(z_i,z_j) = ||z_i - z_j||$.
  - The z_i approximate the high-dimensional L_1-norm distances.



Sammon's Mapping

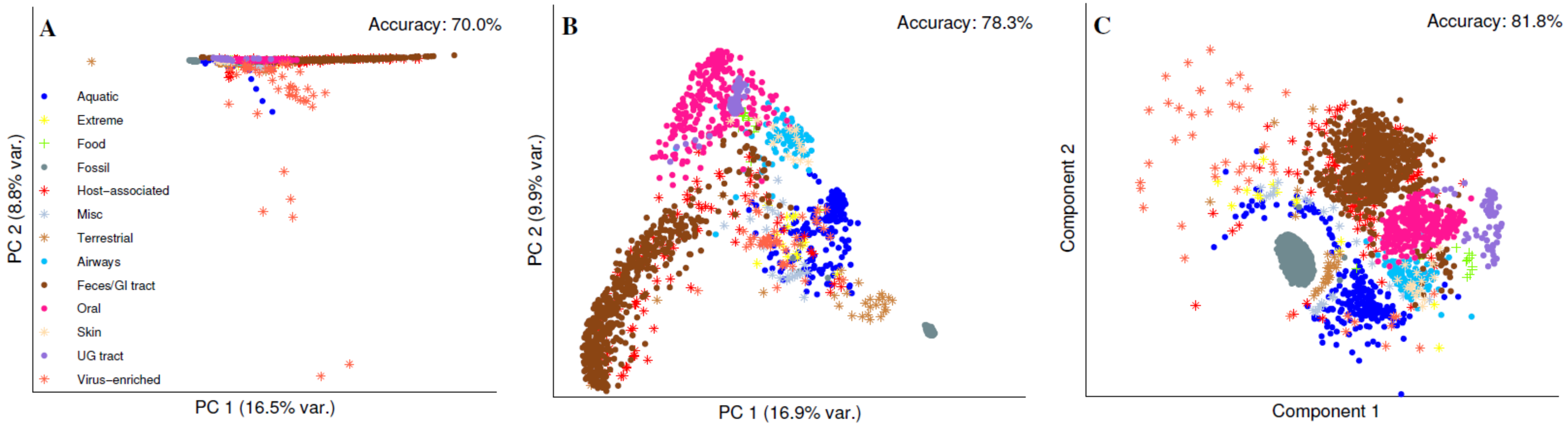
- Challenge for most MDS models: they focus on large distances.
  - Leads to "crowding" effect like with PCA.
- Early attempt to address this is Sammon's mapping:
  - Weighted MDS so large/small distances more comparable.

$$f(Z) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left(\frac{d_2(z_i, z_j) - d_1(x_i, x_j)}{d_1(x_i, x_j)} \right)^2$$

  - Denominator reduces focus on large distances.

Sammon's Mapping

- Challenge for most MDS models: they focus on large distances.
  - Leads to "crowding" effect like with PCA.
- Early attempt to address this is Sammon's mapping:
  - Weighted MDS so large/small distances more comparable.



Summary

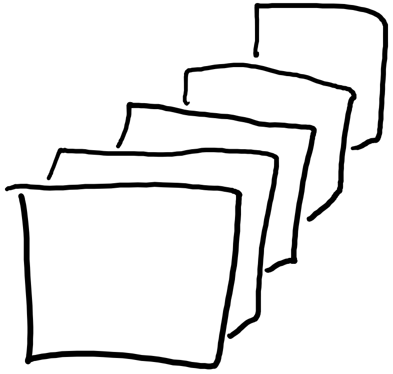
- Recommender systems try to recommend products.
- Collaborative filtering tries to fill in missing values in a matrix.
  - Matrix factorization is a common approach.
- SVDfeature combines linear regression and matrix factorization.
- Multi-dimensional scaling is non-parametric latent-factor model.
- Different distances/losses/weights usually gives better results.

- Next time: fixing MDS and discovering new types of Leukemia cells.

Bonus Slide: Tensor Factorization

- Tensors are higher-order generalizations of matrices:

Scalar $\alpha = \underset{1\times 1}{[\ ]}$ Vector $a = \begin{bmatrix} \\ \\ \end{bmatrix}_{d\times 1}$ Matrix $A = \begin{bmatrix} & \\ & \end{bmatrix}_{d\times d}$ Tensor $A =$



$d\times d\times d$

- Generalization of matrix factorization is tensor factorization:

$$y_{ijm} \approx \sum_{c=1}^{k} w_{jc} z_{ic} v_{mc}$$

- Useful if there are other relevant variables:
  - Instead of ratings based on {user,movie}, ratings based {user,movie,age}.
  - Useful if ratings change over time.

Bonus Slide: Multivariate Chain Rule

- Recall the univariate chain rule: $$\frac{d}{dw}[f(g(w))] = f'(g(w))\,g'(w)$$
- The multivariate chain rule: $$\underbrace{\nabla[f(g(w))]}_{d\times 1} = \underbrace{f'(g(w))}_{1\times 1}\underbrace{\nabla g(w)}_{d\times 1}$$
- Example:

$$\nabla\left[\tfrac{1}{2}(w^Tx_i - y_i)^2\right]$$

$$= \nabla[f(g(w))]$$

with $g(w) = w^Tx_i - y_i$ $\longrightarrow$ $\nabla g(w) = x_i$

and $f(r_i) = \frac{1}{2}r_i^2$ $\longrightarrow$ $f'(r_i) = r_i$

$$\nabla[f(g(w))] = r_i\, x_i$$

$$= (w^Tx_i - y_i)x_i$$

Bonus Slide: Multivariate Chain Rule for MDS

- General MDS formulation:

$$\underset{Z \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \sum_{i=1}^{n} \sum_{j=i+1}^{n} g(d_1(x_i, x_j), d_2(z_i, z_j))$$

- Using multivariate chain rule we have:

$$\nabla_{z_i} g(d_1(x_i, x_j), d_2(z_i, z_j)) = g'(d_1(x_i, x_j), d_2(z_i, z_j)) \nabla_{z_i} d_2(z_i, z_j)$$

- Example: If $d_1(x_i, x_j) = \|x_i - x_j\|$ and $d_2(z_i, z_j) = \|z_i - z_j\|$ and $g(d_1, d_2) = \frac{1}{2}(d_1 - d_2)^2$

$$\nabla_{z_i} g(d_1(x_i, x_j), d_2(z_i, z_j)) = \underbrace{-(d_1(x_i, x_j) - d_2(z_i, z_j))}_{g'(d_1, d_2)} \underbrace{\left[-\frac{(z_i - z_j)}{2\|z_i - z_j\|}\right]}_{\nabla_{z_i} d_2(z_i, z_j)}$$

↳ Assuming $z_i \neq z_j$

(move distances closer) (how distance changes in z-space)