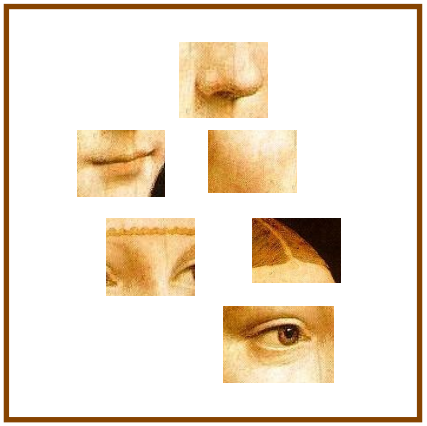
Feature Visualization

Dictionary Learning:
Learn Visual Words using clustering

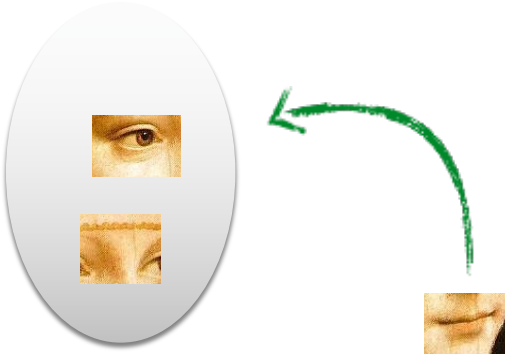
1. extract features (e.g., SIFT) from images

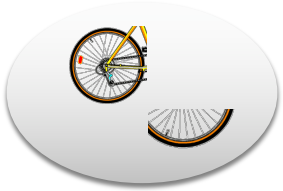


2. Learn visual dictionary (e.g., K-means clustering)

Encode:

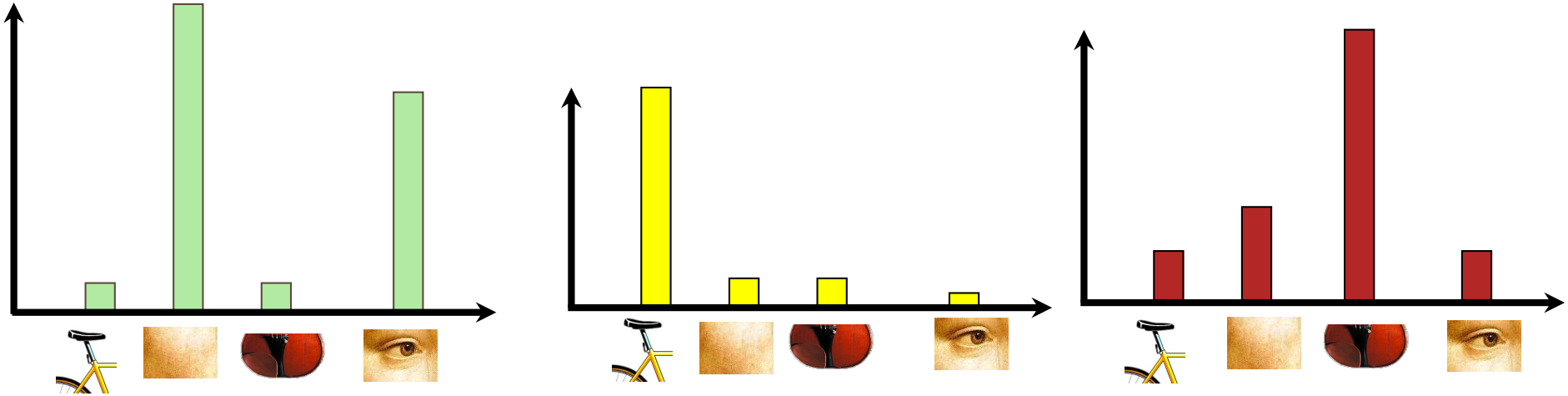
build Bags-of-Words (BOW) vectors for each image





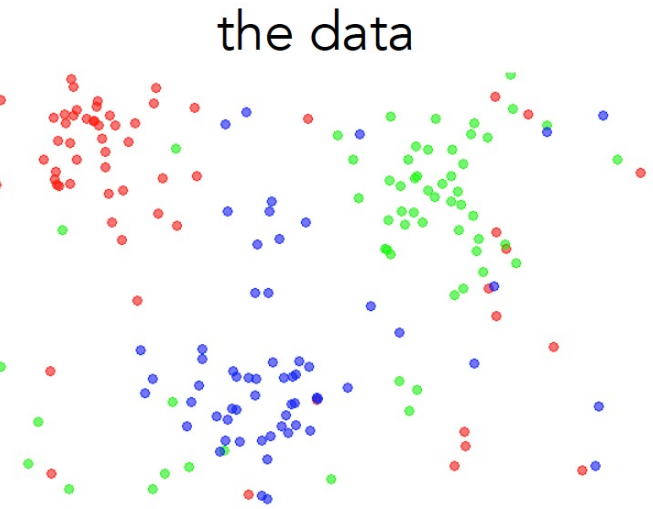
1. Quantization: image features get associated to a visual word (nearest cluster center)

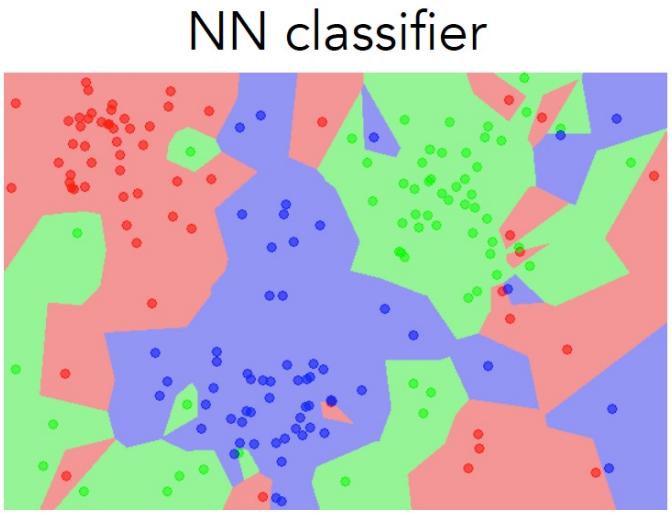
2. Histogram: count the number of visual word occurrences

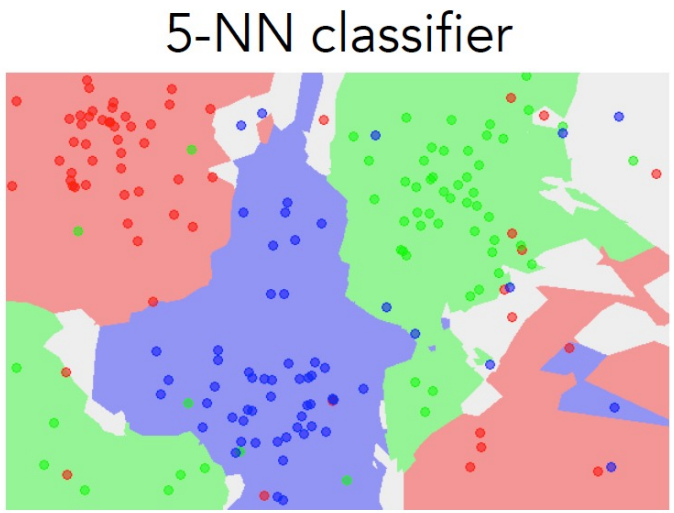


k-nearest neighbor

- Find the k closest points from training data
- Labels of the k points “vote” to classify



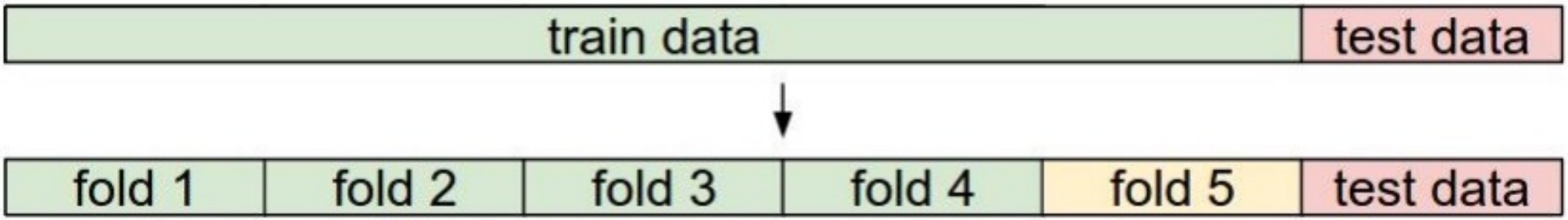




What is the best distance to use?

What is the best value of k to use?

Cross-validation



Support Vector Machine

define a **score function**

data (image)

$$f(x_i, W, b) = Wx_i + b$$

class scores

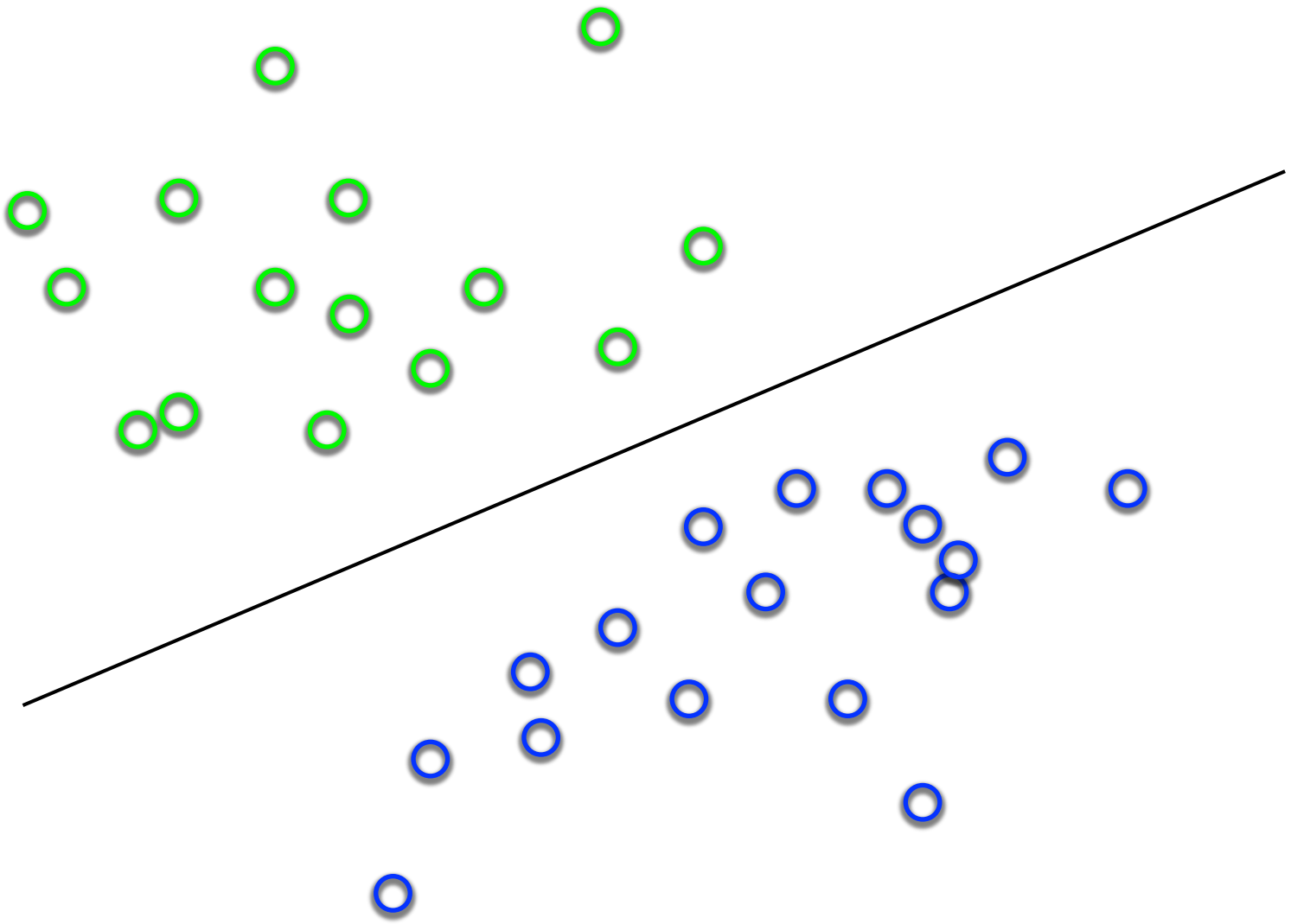
"weights"

"bias vector"

"parameters"

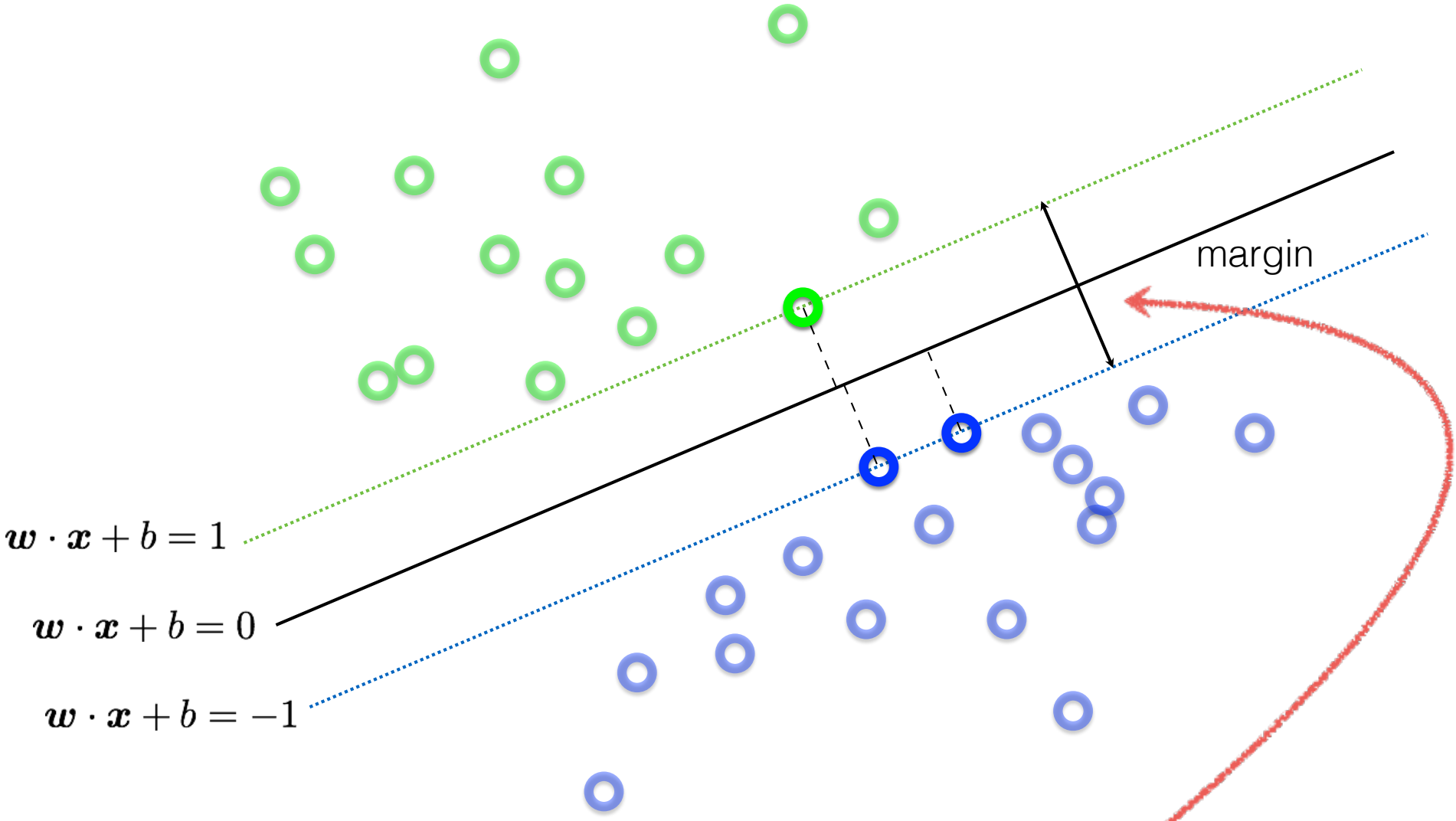
What's the best **w**?

Intuitively, the line that is the farthest from all interior points



Maximum Margin solution:
most stable to perturbations of data

Find hyperplane **w** such that …



the gap between parallel hyperplanes $\frac{2}{\|\boldsymbol{w}\|}$ is maximized

Can be formulated as a maximization problem

$$\max_{\boldsymbol{w}} \frac{2}{\|\boldsymbol{w}\|}$$

$$\text{subject to } \boldsymbol{w} \cdot \boldsymbol{x}_i + b \begin{array}{l} \geq +1 \text{ if } y_i = +1 \\ \leq -1 \text{ if } y_i = -1 \end{array} \text{ for } i = 1, \ldots, N$$

Equivalently,

Where did the 2 go?

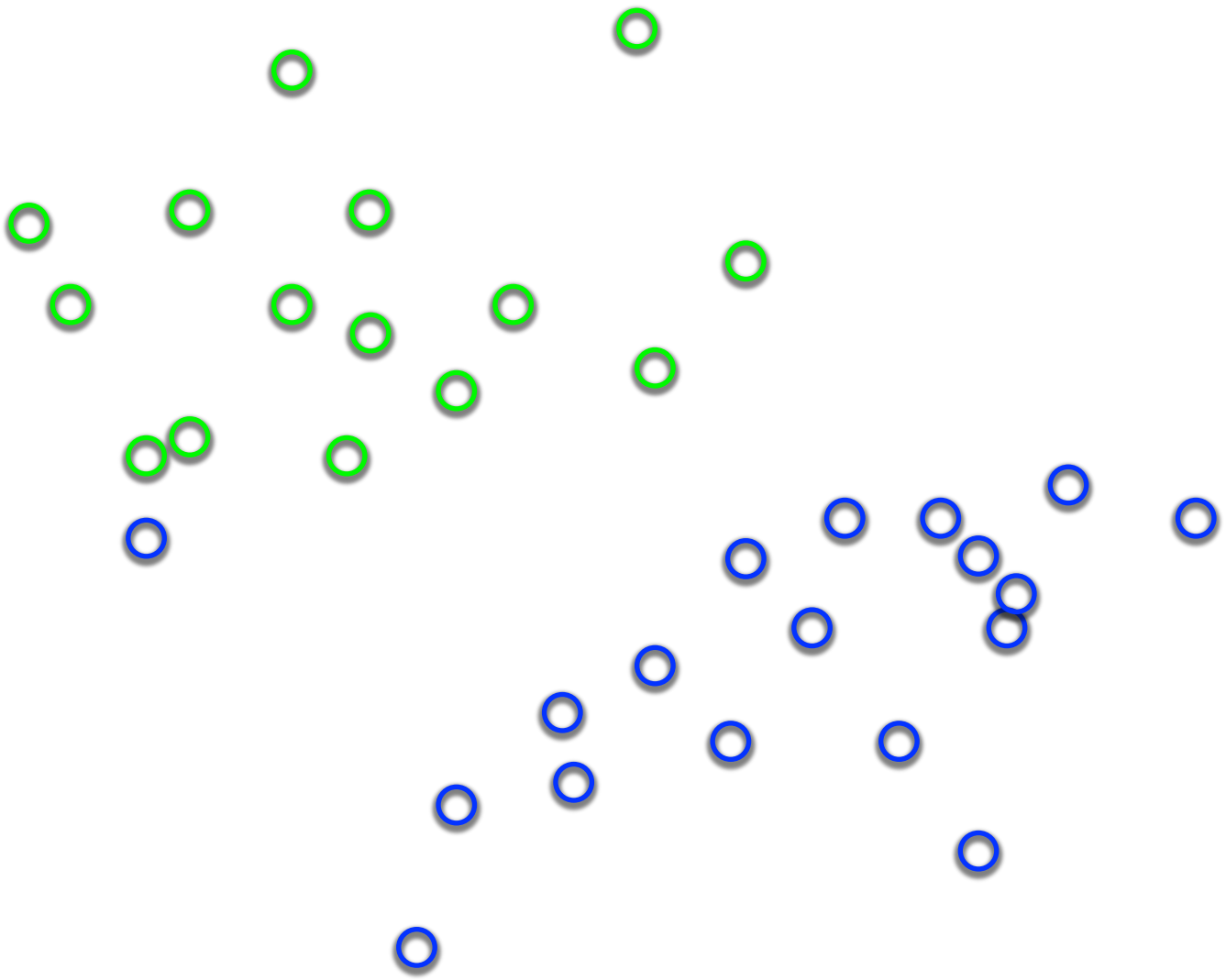
$$\min_{\boldsymbol{w}} \|\boldsymbol{w}\|$$

$$\text{subject to } y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 \text{ for } i = 1, \ldots, N$$

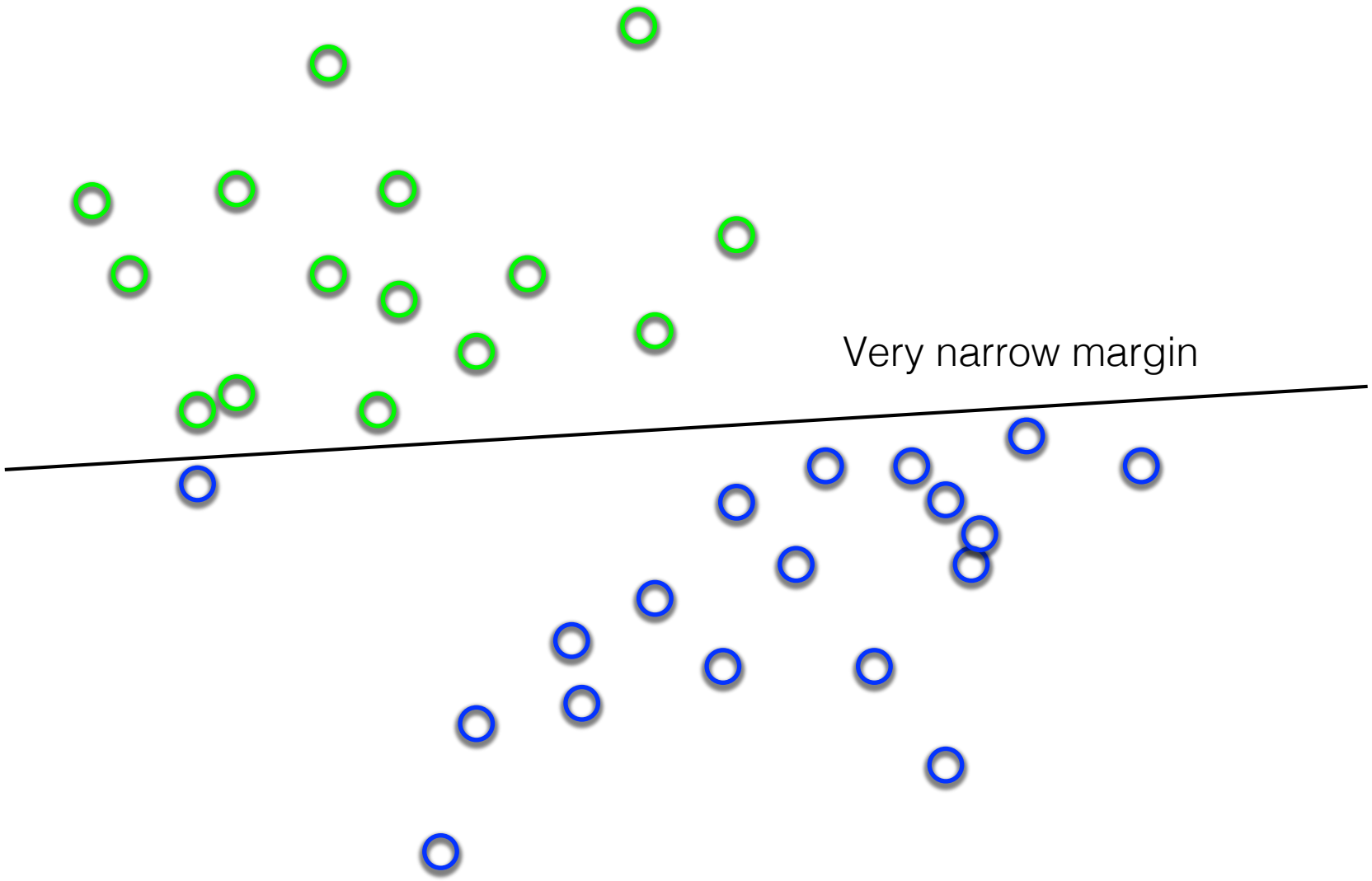
What happened to the labels?

‘soft’ margin

What's the best **w**?

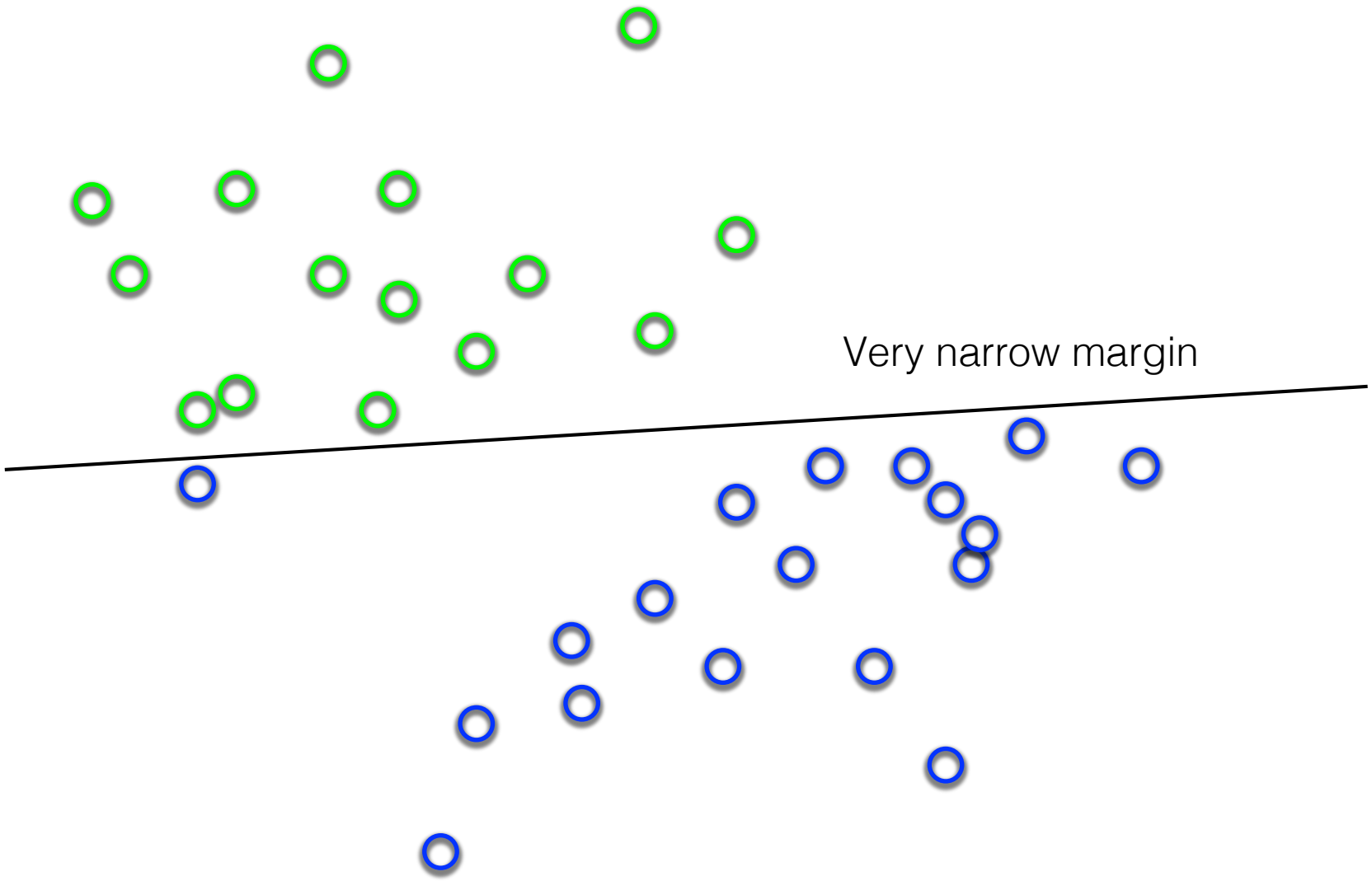


What's the best **w**?



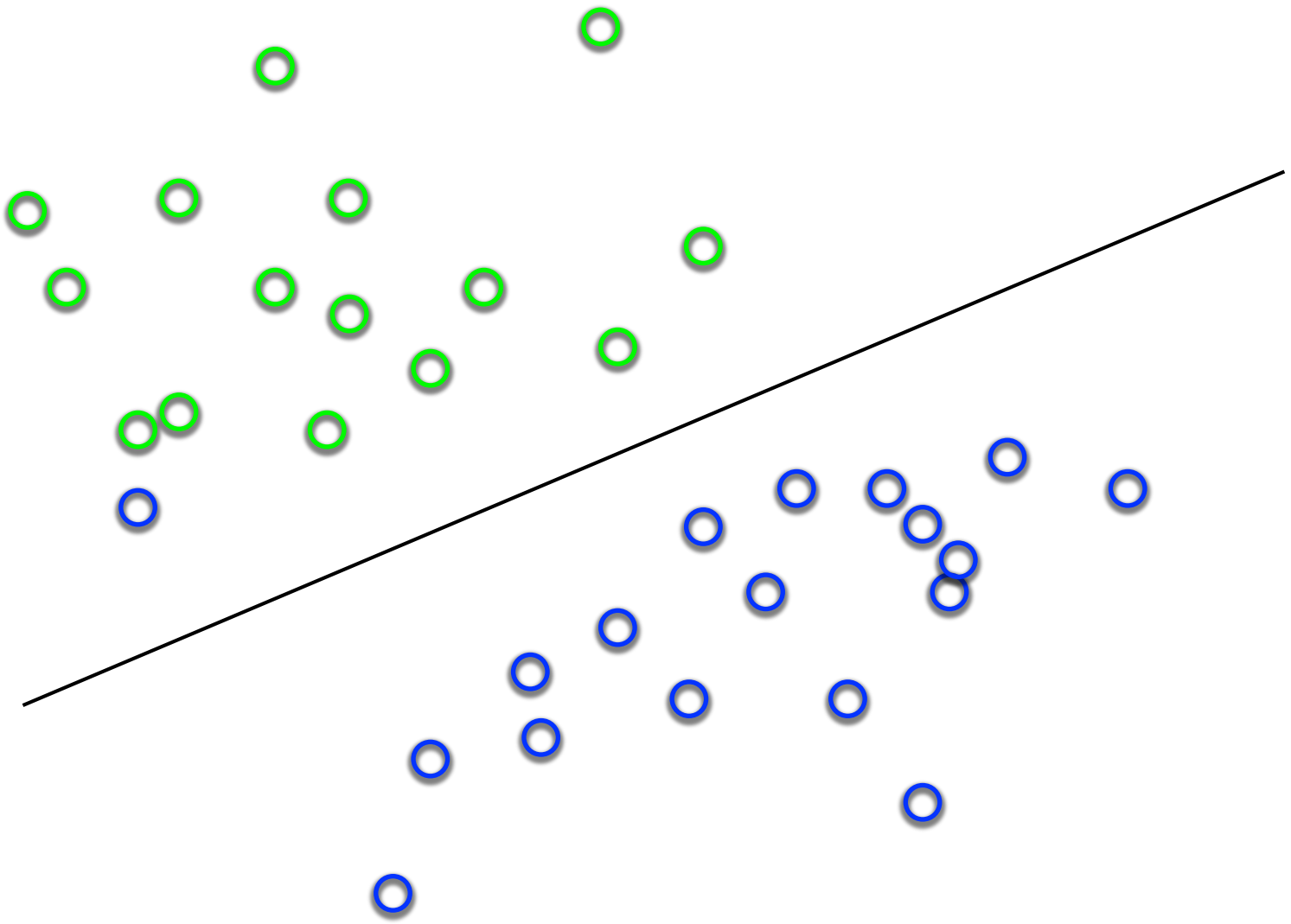
Separating cats and dogs

What's the best **w**?



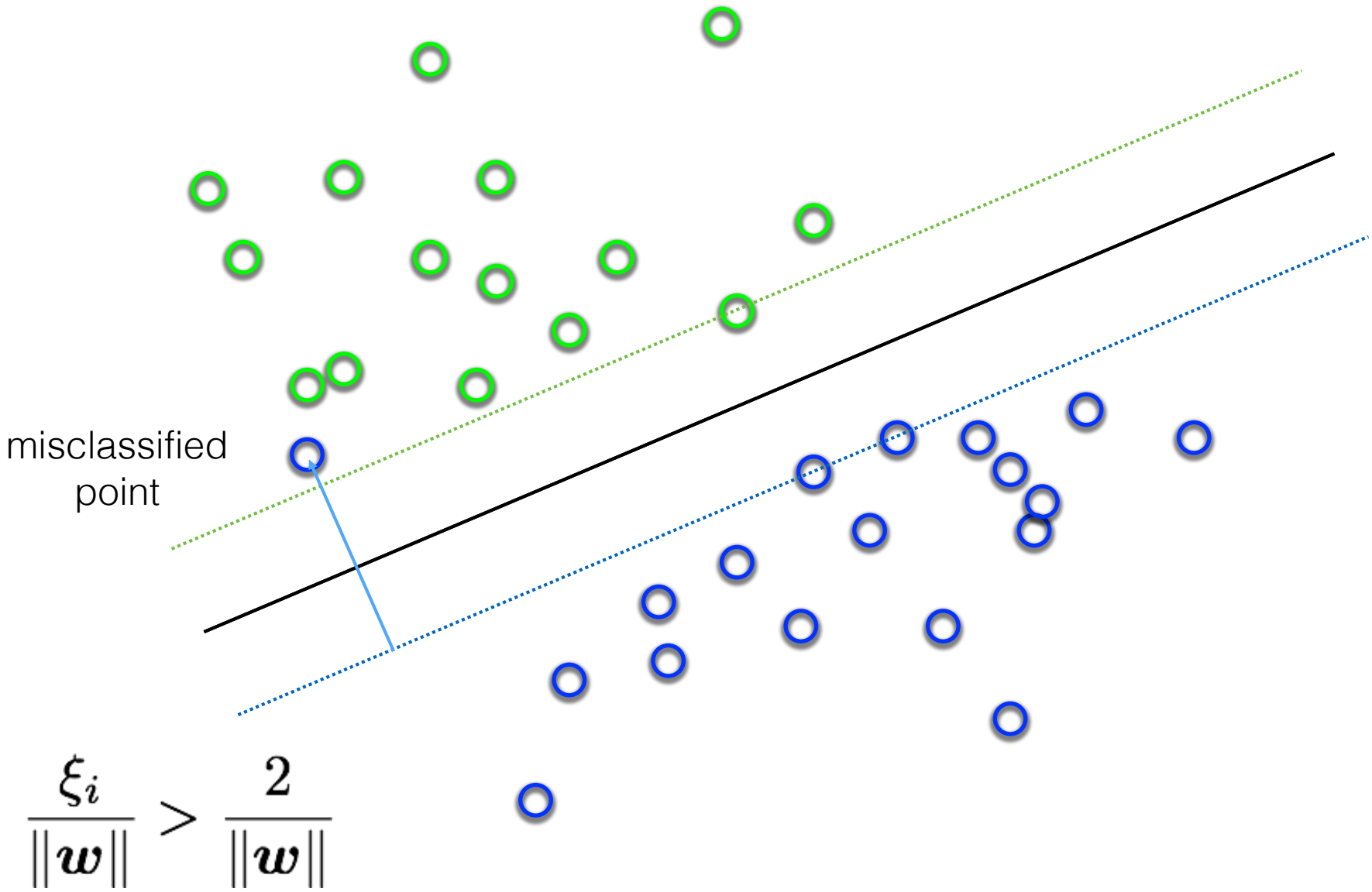
Intuitively, we should allow for some misclassification if we can get more robust classification

What's the best **w**?



Trade-off between the MARGIN and the MISTAKES
(might be a better solution)

Adding slack variables $\xi_i \geq 0$



$$\frac{\xi_i}{\|\boldsymbol{w}\|} > \frac{2}{\|\boldsymbol{w}\|}$$

‘soft’ margin

objective

$$\min_{\boldsymbol{w},\boldsymbol{\xi}} \|\boldsymbol{w}\|^2 + C\sum_i \xi_i$$

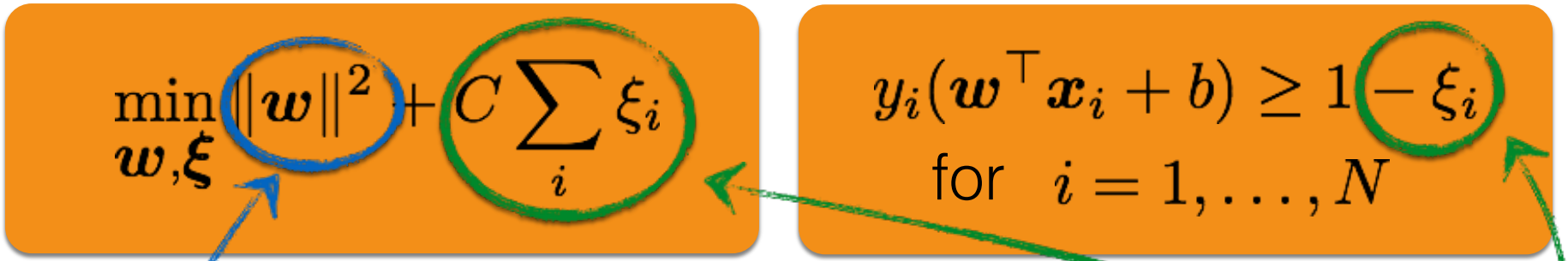
subject to

$$y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 - \xi_i$$
$$\text{for} \quad i = 1, \ldots, N$$

‘soft’ margin

objective

subject to



The slack variable allows for mistakes,
as long as the inverse margin is minimized.

‘soft’ margin

objective

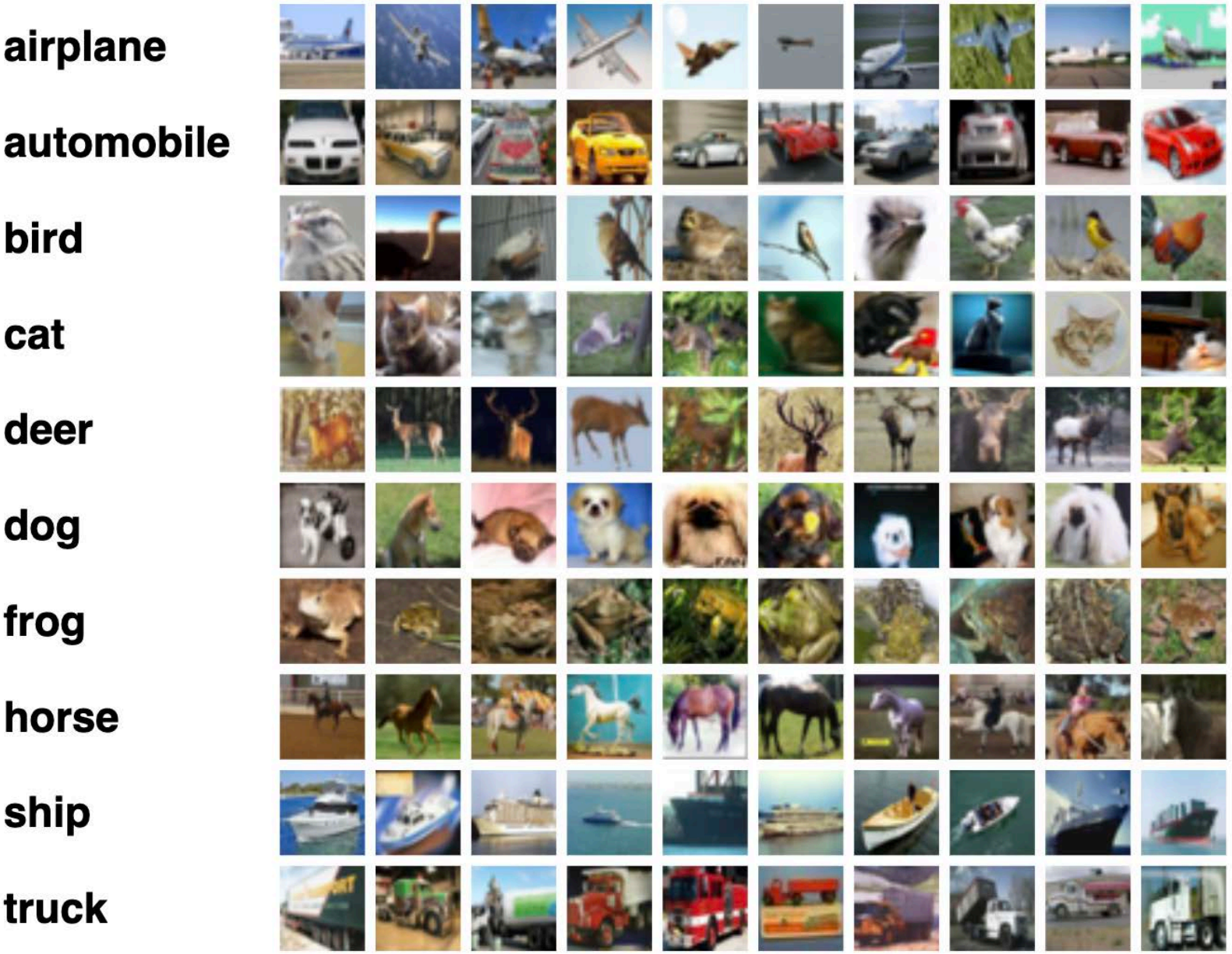
$$\min_{\boldsymbol{w},\boldsymbol{\xi}} \|\boldsymbol{w}\|^2 + C\sum_i \xi_i$$

subject to

$$y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 - \xi_i \quad \text{for} \quad i = 1, \ldots, N$$

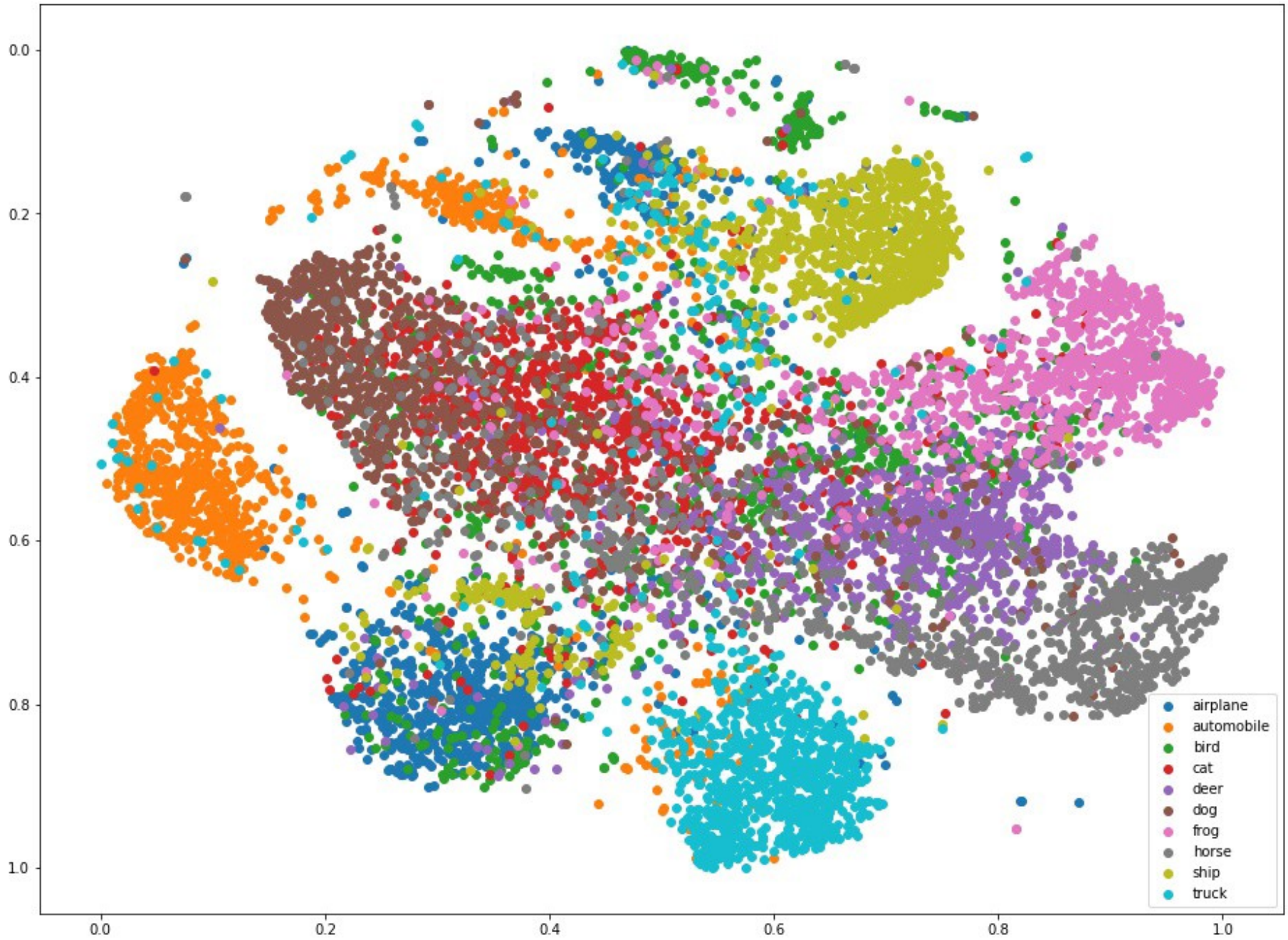
- Every constraint can be satisfied if slack is large
- C is a regularization parameter
  - Small C: ignore constraints (larger margin)
  - Big C: constraints (small margin)
- Still QP problem (unique solution)

Where are we heading?



https://www.cs.toronto.edu/~kriz/cifar.html

CIFAR-10 [50k+10k]



Dimensionality of input

- Number of Features [Consider ***SIFT Point of Interest***]
- If number of features is increased
  - More time to compute
  - More memory to store inputs and intermediate results
  - More complicated explanations (knowledge from learning)
    - Classification from 100 vs. 2 parameters
  - No simple visualization
    - 2D vs. 10D graph
  - Need much more data (curse of dimensionality)
    - 1M of 1-d inputs is not equal to 1 input of dimension 1M

Dimensionality reduction

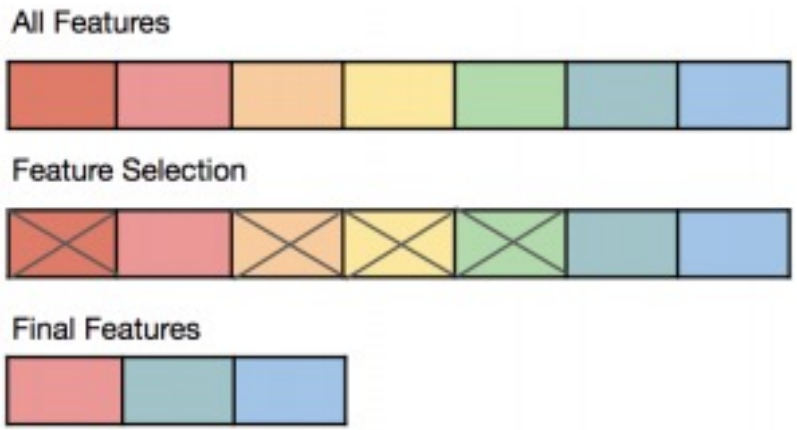
- Some features (dimensions) bear little or nor useful information
  - Can drop some features
  - Have to estimate which features can be dropped from data

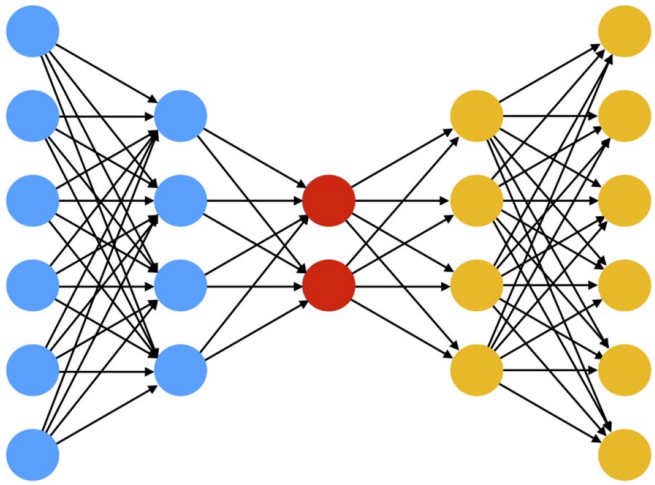
- Several features can be combined together without loss or even with gain of information
  - Some features can be combined together
  - Have to estimate which features to combine from data

Feature Selection *v.s.* Extraction

- Feature selection:
  - Choosing $k<d$ important features, ignoring the remaining $d - k$
    Subset selection algorithms

- Feature extraction:
  - Project the original x_i , i =1,...,d dimensions to new $k<d$ dimensions, z_j , j =1,...,k





Subset-selection

- Forward search
  - Start from empty set of features
  - Try each of remaining features
  - Estimate classification/regression error for adding specific feature
  - Select feature that gives maximum improvement in validation error
  - Stop when no significant improvement

- Backward search
  - Start with original set of size d
  - Drop features with smallest impact on error

Floating Search

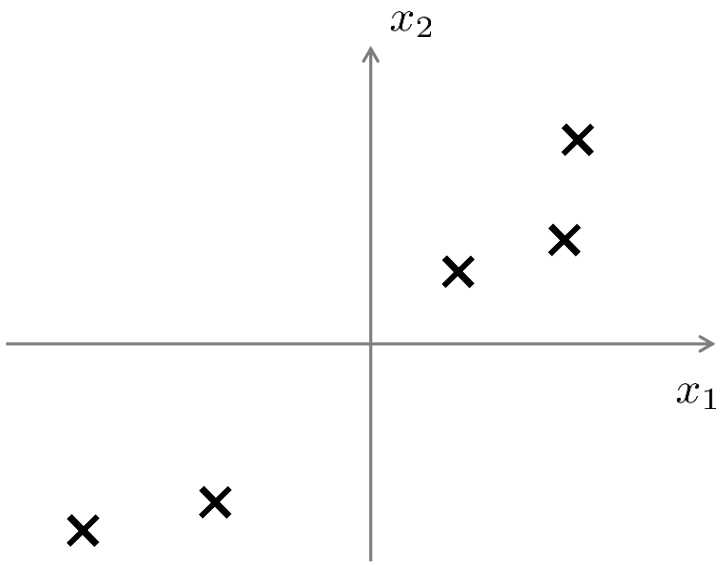
Forward and backward search are “greedy” algorithms

- ❖ Select best options at single step
- ❖ Do not always achieve optimum value

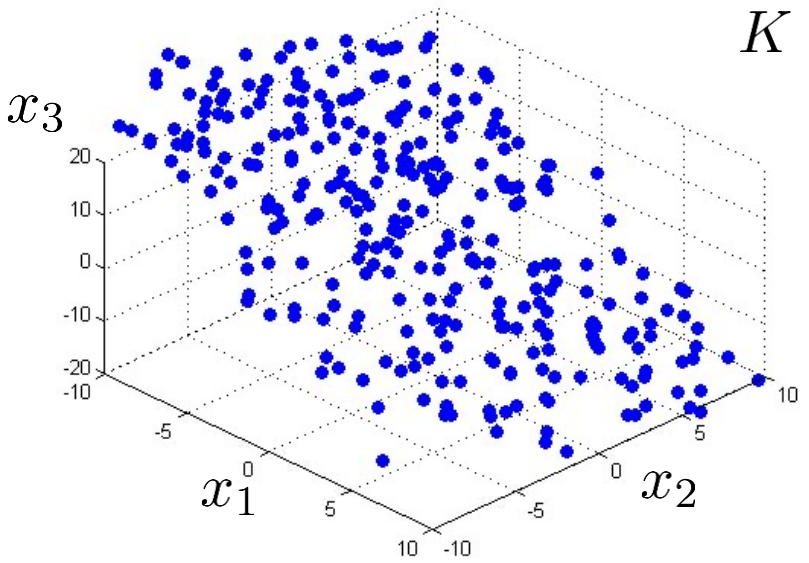
Floating search

- ➢ Two types of steps: Add *k*, remove *l*
- ➢ *More computations*

Principal Components Analysis



$3D \to 2D$
$K = 2$



Reduce from 2-D to 1-D: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$)
onto which to project the data so as to minimize the projection error.

Reduce from n-D to k-D: Find k vectors $u^{(1)}, u^{(2)}, \ldots, u^{(k)}$
onto which to project the data, so as to minimize the projection error.

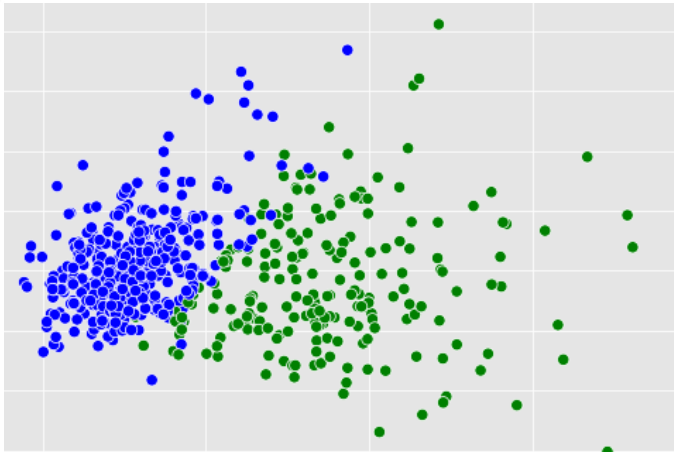
Principal Components Analysis

Goal: Find r-dim projection that best preserves variance

1. Compute mean vector μ and covariance matrix Σ of original points
2. Compute eigenvectors and eigenvalues of Σ
3. Select top r eigenvectors
4. Project points onto subspace spanned by them:

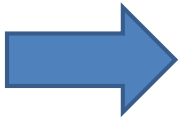
$$y = A(x - \mu)$$

where y is the new point, x is the old one, and the rows of A are the eigenvectors

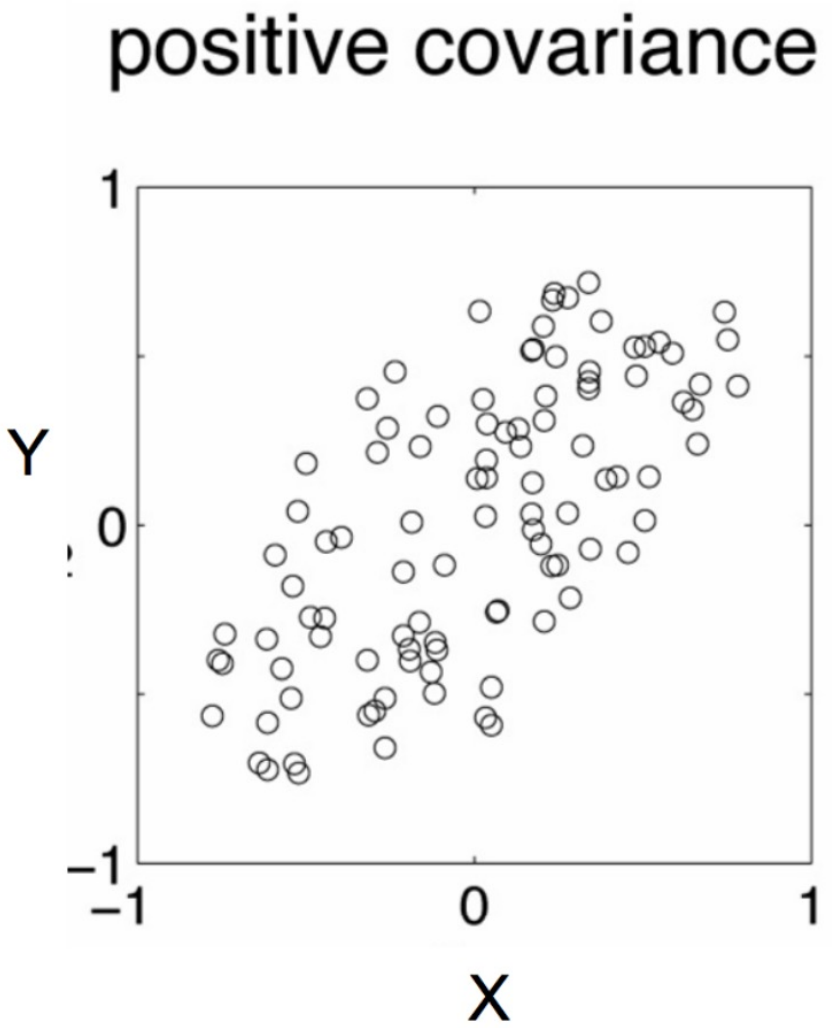


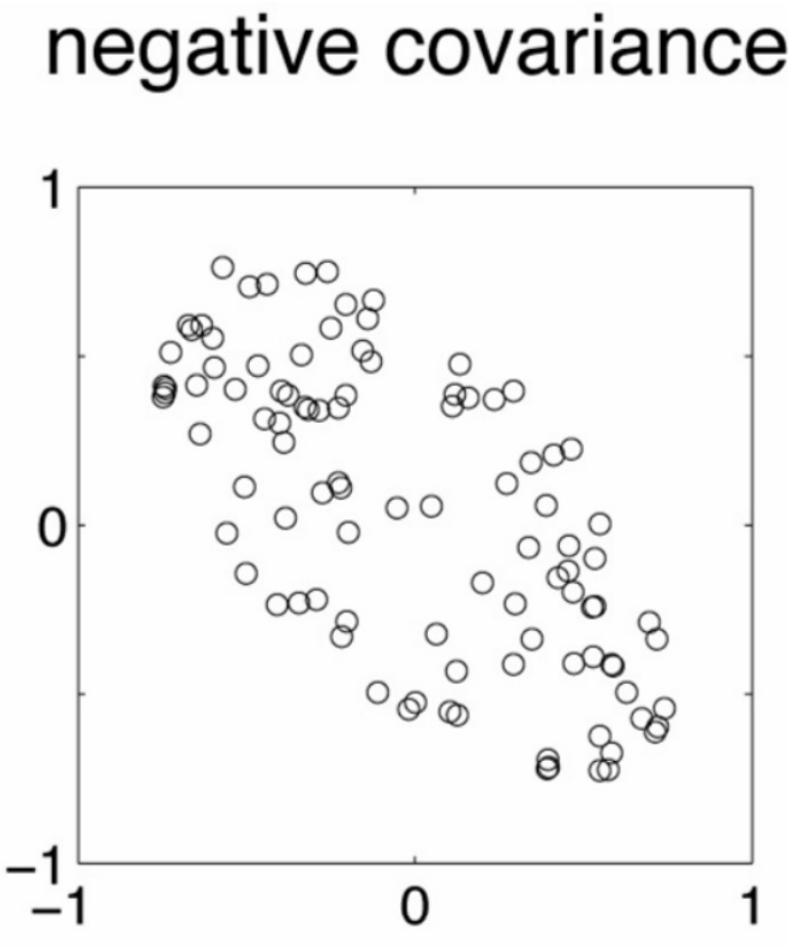
Covariance

- **Variance and Covariance:**
  Measure of the "spread" of a set of points around their center of mass (mean)
- **Variance:**
  Measure of the deviation from the mean for points in one dimension
- **Covariance:**
  Measure of how much each of the dimensions vary from the mean with **respect to each other**



- **Covariance is measured between two dimensions**
- **Covariance sees if there is a relation between two dimensions**
- **Covariance between one dimension is the variance**





Positive: Both dimensions increase or decrease together

Negative: While one increase the other decrease

Covariance

Used to find relationships between dimensions in high dimensional data sets

$$q_{jk} = \frac{1}{N} \sum_{i=1}^{N} (X_{ij} - E(X_j))(X_{ik} - E(X_k))$$

The Sample mean

Eigenvector and Eigenvalue

$$Ax = \lambda x$$

A: Square Matrix

λ: Eigenvector or characteristic vector

X: Eigenvalue or characteristic value

- ***The zero vector can not be an eigenvector***
- ***The value zero can be eigenvalue***

Principal Component Analysis

Input: $\mathbf{x} \in \mathbb{R}^D$: $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$

Set of basis vectors: $\mathbf{u}_1, \ldots, \mathbf{u}_K$

Summarize a *D* dimensional vector *X* with *K* dimensional feature vector *h(x)*

$$h(\mathbf{x}) = \begin{bmatrix} \mathbf{u}_1 \cdot \mathbf{x} \\ \mathbf{u}_2 \cdot \mathbf{x} \\ \cdots \\ \mathbf{u}_K \cdot \mathbf{x} \end{bmatrix}$$

Principal Component Analysis

$$\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_K]$$

New data representation *h(x)*

$$h(\mathbf{x}) = \mathbf{U}^T \mathbf{x}$$

$$h(\mathbf{x}) = \mathbf{U}^T (\mathbf{x} - \mu_0)$$

Empirical mean of the data $\mu_0 = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i$

The space of all face images

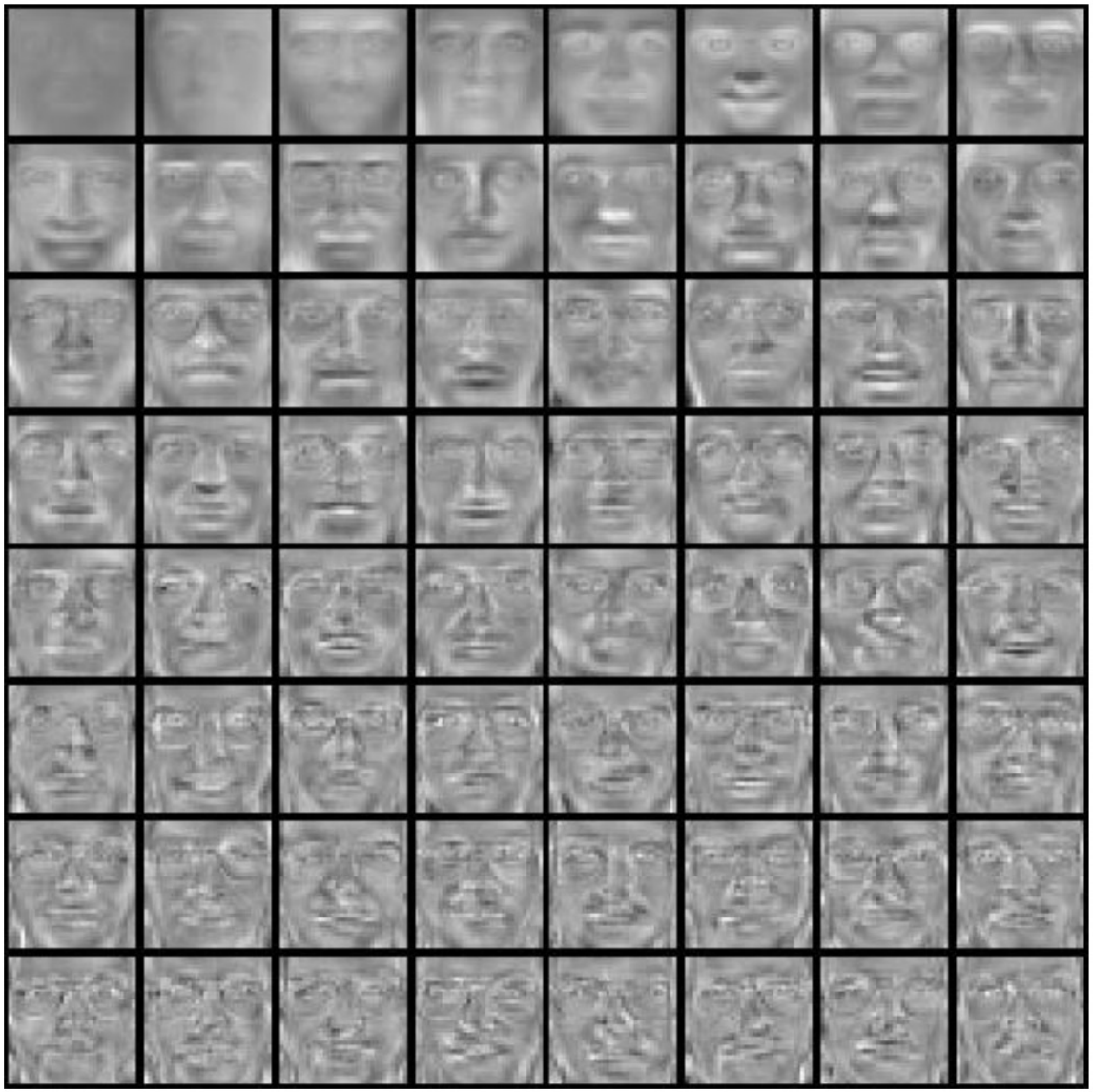
- When viewed as vectors of pixel values, face images are extremely high-dimensional
  - 100x100 image = 10,000 dimensions
  - Slow and lots of storage
- But very few 10,000-dimensional vectors are valid face images
- We want to effectively model the subspace of face images

slide by Derek Hoiem

Eigenfaces example

Mean: μ

Top eigenvectors: $u_1,\ldots u_k$



slide by Derek Hoiem

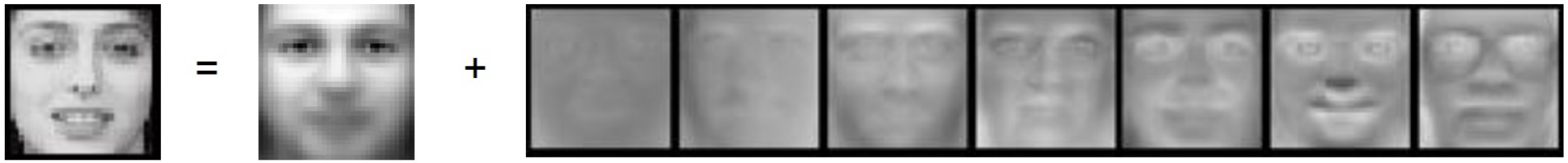
Representation and reconstruction

- Face **x** in "face space" coordinates:

$$\mathbf{x} \rightarrow \left[\mathbf{u}_1^{\mathrm{T}}(\mathbf{x}-\mu), \ldots, \mathbf{u}_k^{\mathrm{T}}(\mathbf{x}-\mu)\right]$$

$$= \quad w_1, \ldots, w_k$$

- Reconstruction:



$$\hat{x} = \mu + w_1 u_1 + w_2 u_2 + w_3 u_3 + w_4 u_4 + \ldots$$

slide by Derek Hoiem

Reconstruction

P = 4

P = 200

P = 400

After computing eigenfaces using 400 face images from ORL face database

slide by Derek Hoiem

Application: Image compression

Original Image

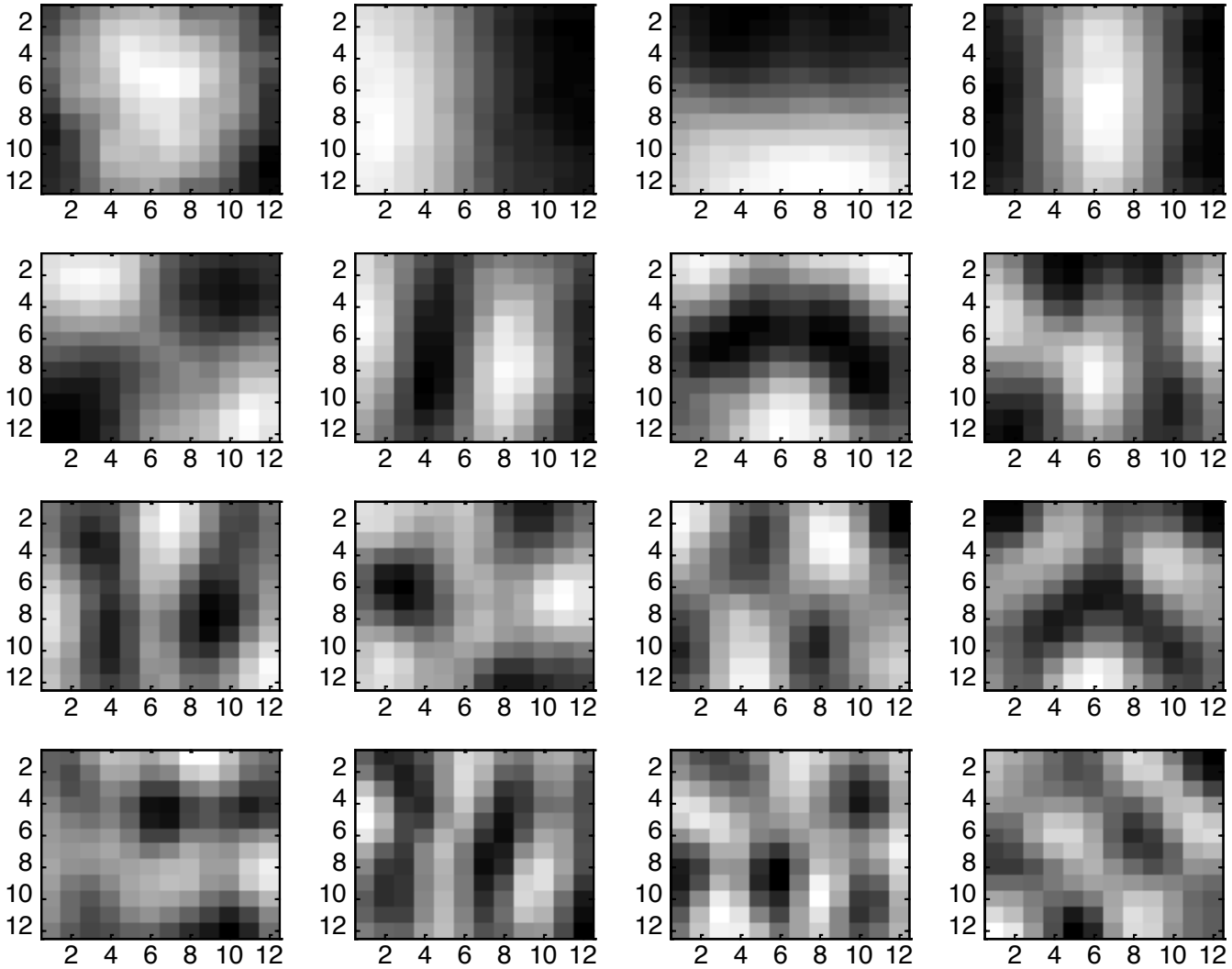
- Divide the original 372x492 image into patches:
- Each patch is an instance that contains 12x12 pixels on a grid
- View each as a 144-D vector

31×41

PCA compression: 144D → 60D

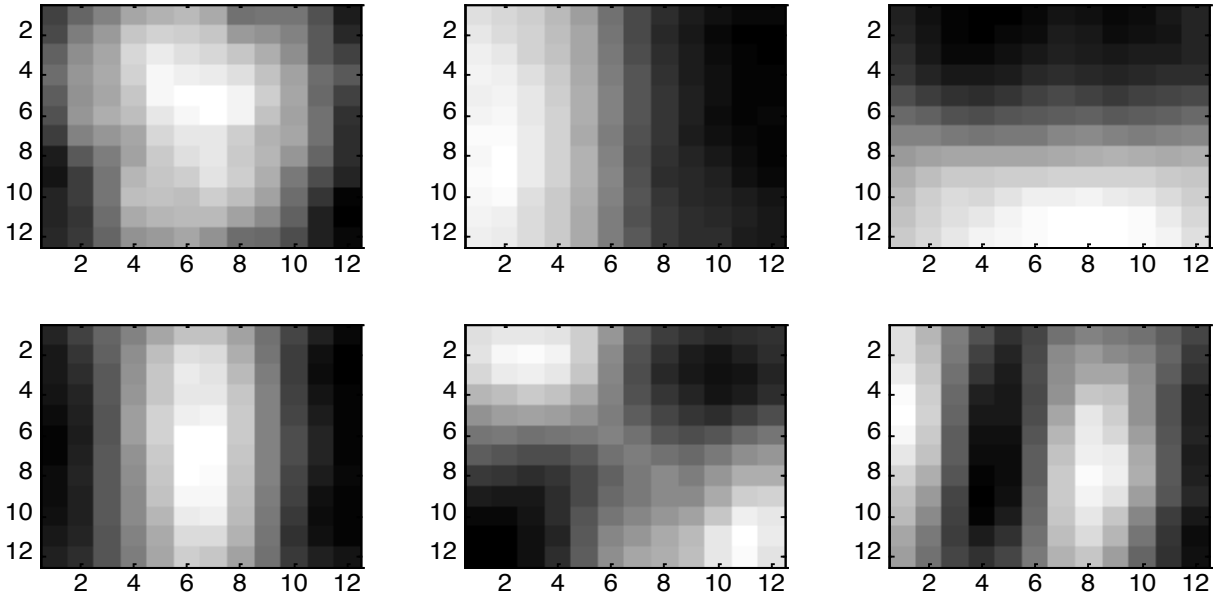
PCA compression: 144D → 16D

16 most important eigenvectors



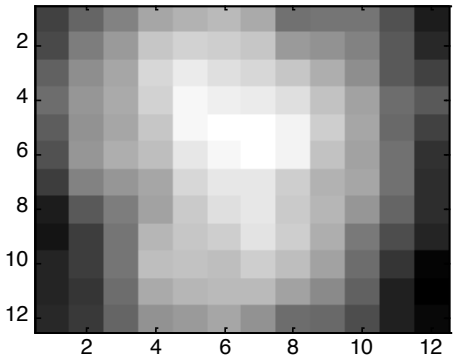
PCA compression: 144D -> 6D

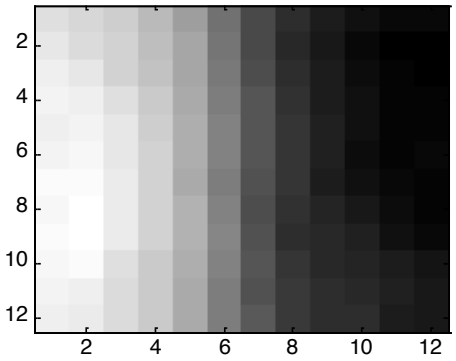
6 most important eigenvectors

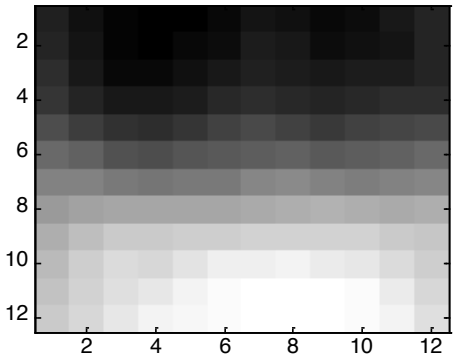


PCA compression: 144D → 3D

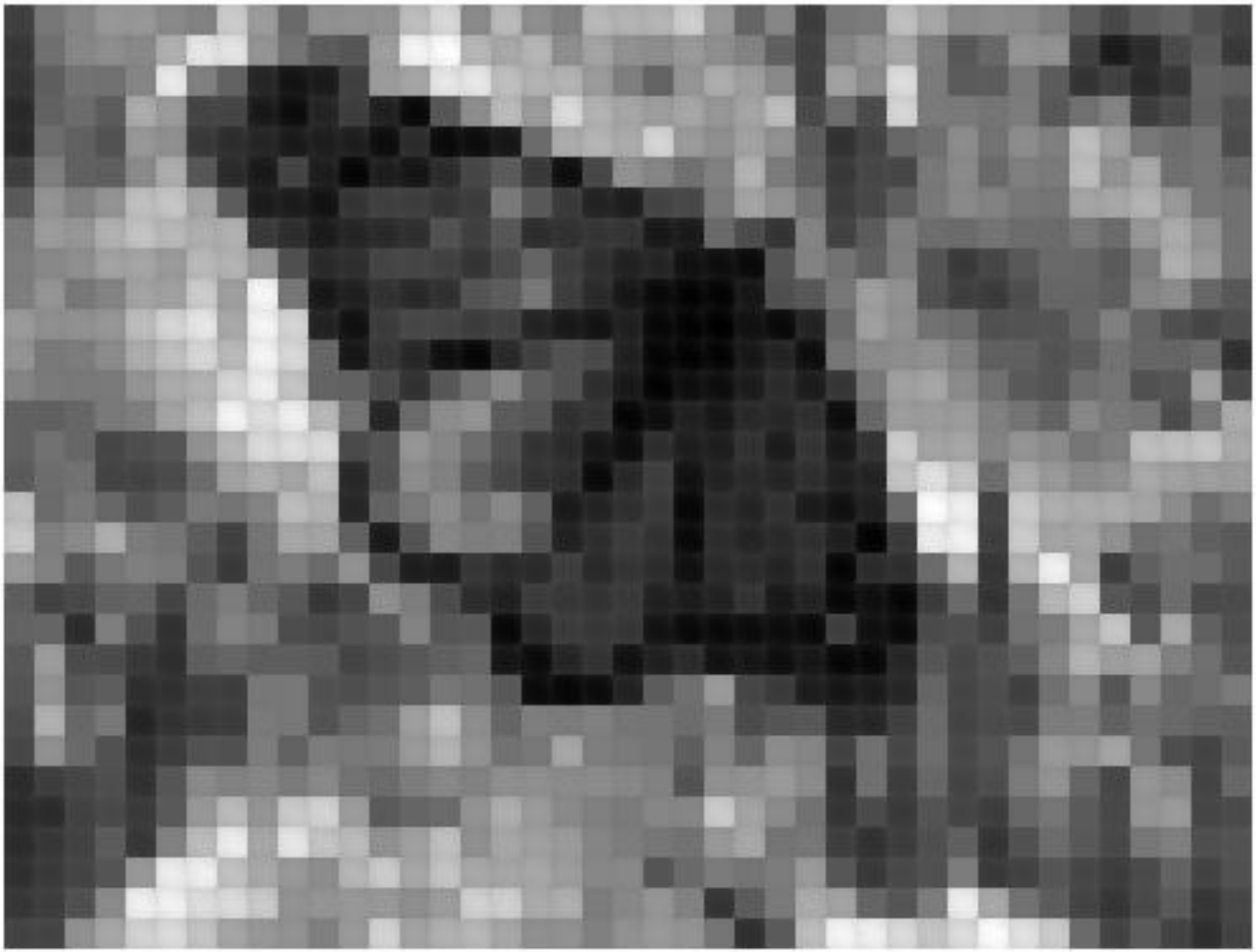
3 most important eigenvectors



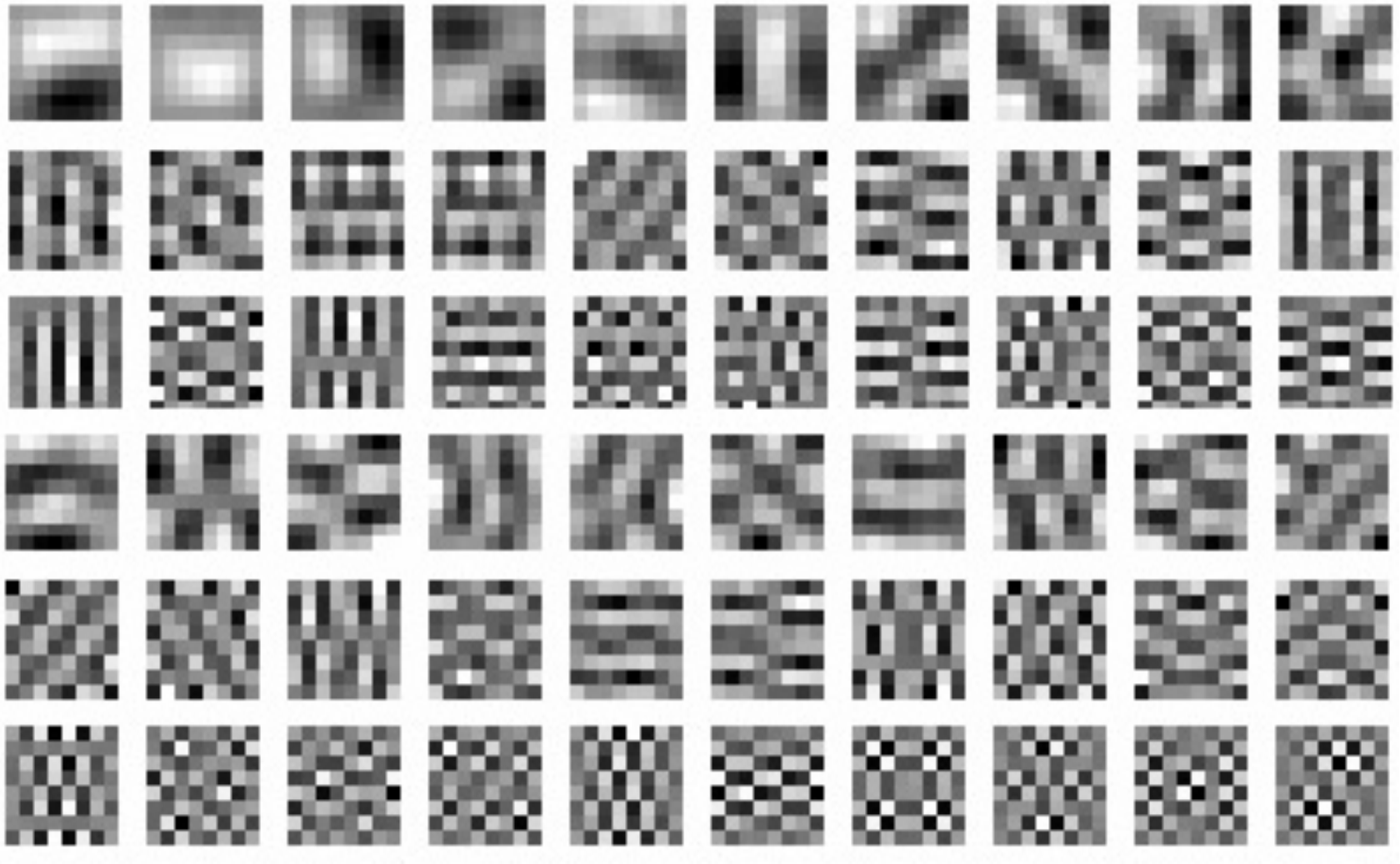




PCA compression: 144D → 1D

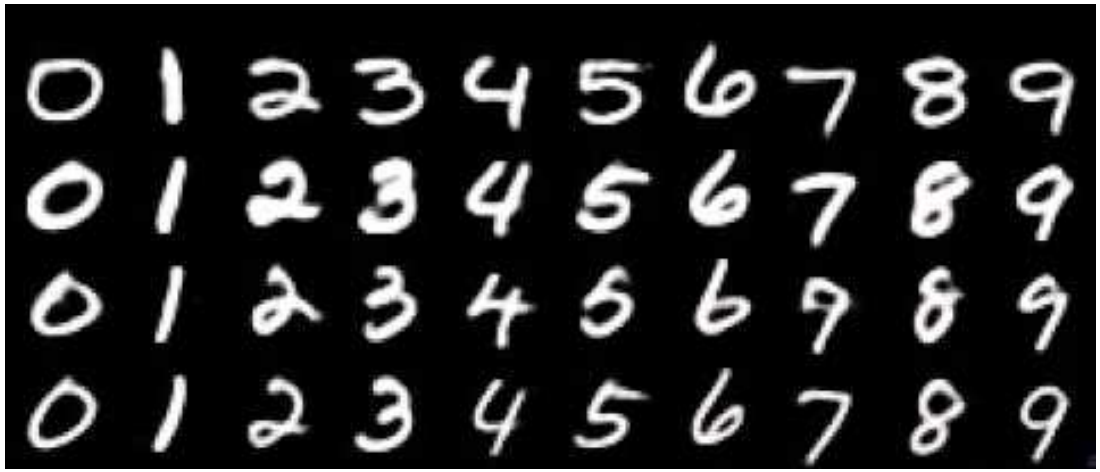


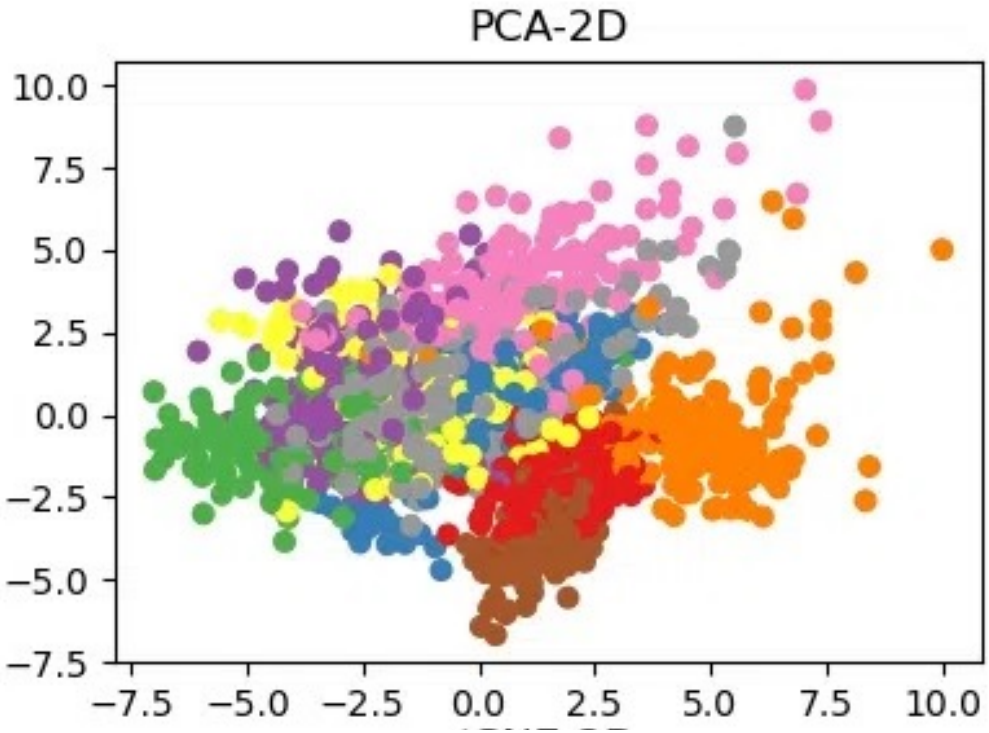
60 most important eigenvectors

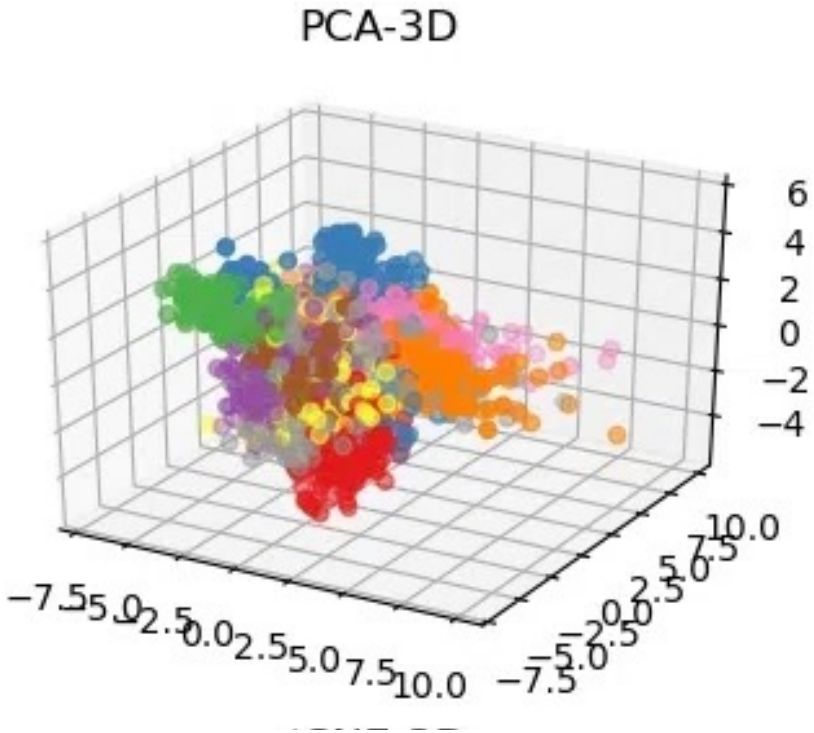


Looks like the discrete cosine bases of JPG!...

PCA on MNIST





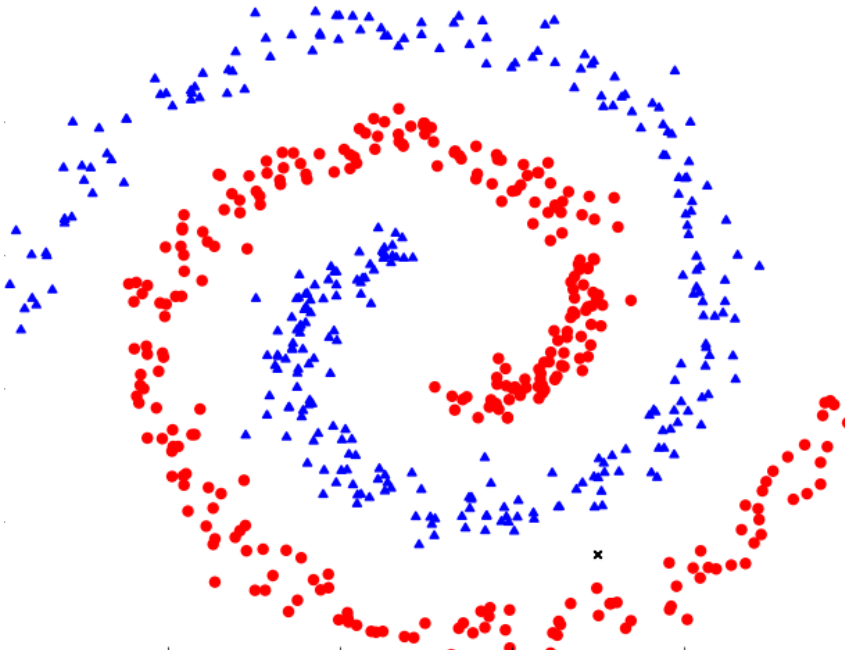


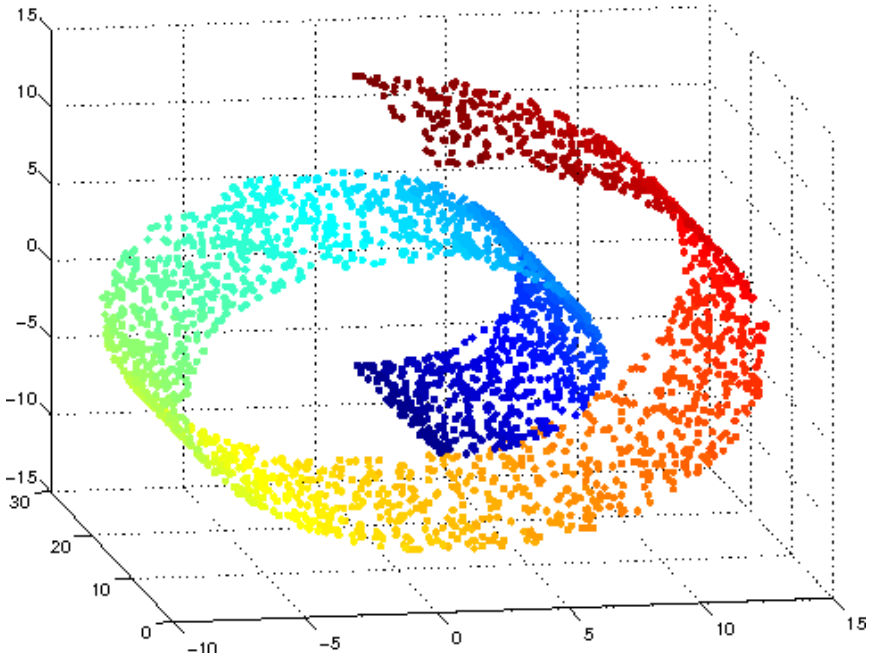
http://yann.lecun.com/exdb/mnist/
https://in2techs.com/mnist-visualization-using-pca-and-tsne-in-python/

Why manifold learning?

Why PCA fails to properly reduce dimensions of MNIST?

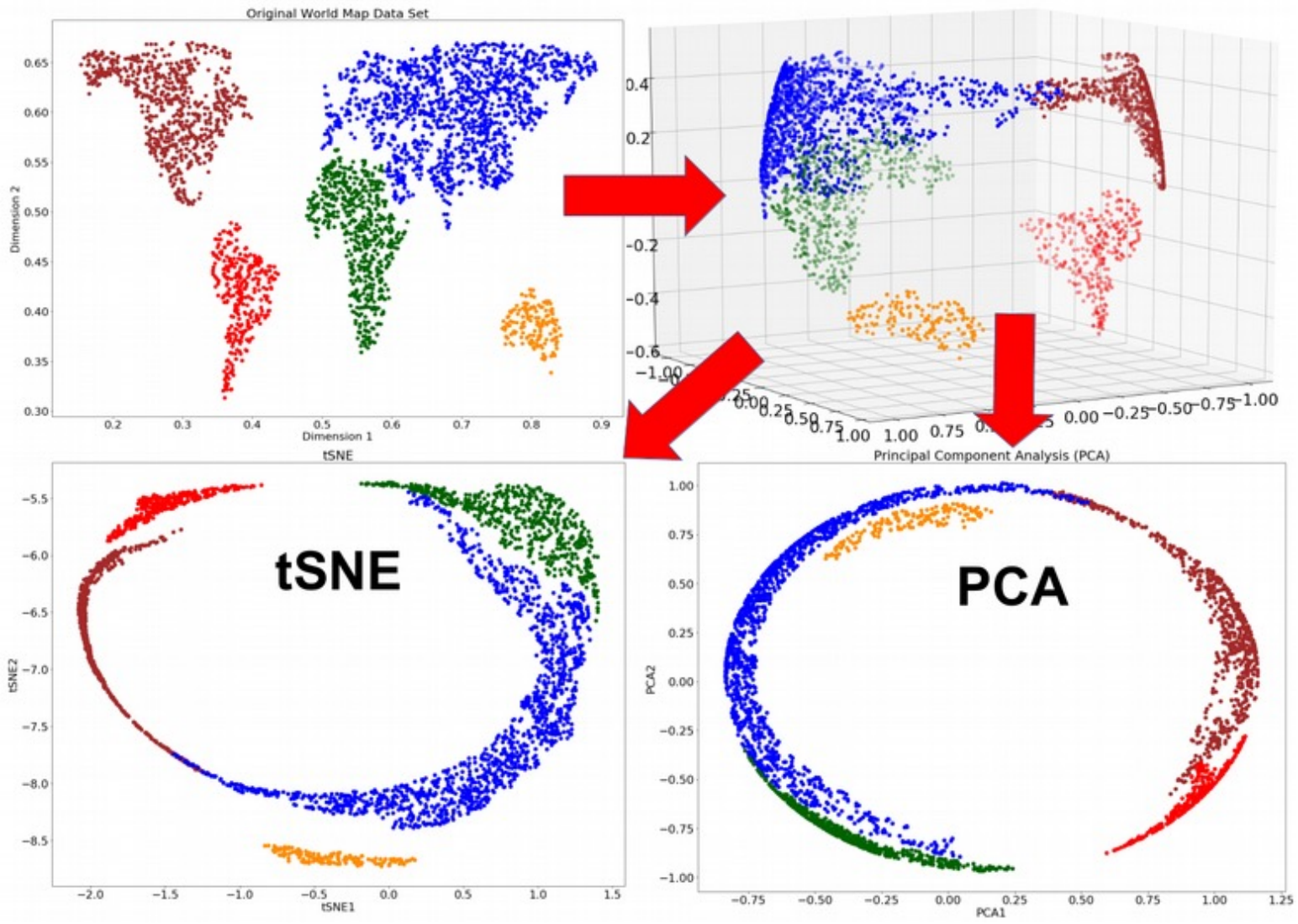
- PCA is good, but it is a linear algorithm, meaning that it cannot represent complex relationship between features





Why manifold learning?

t-SNE is non-linear dimensionality reduction technique that has better performance. It is designed for visualization purposes.



Good visualization

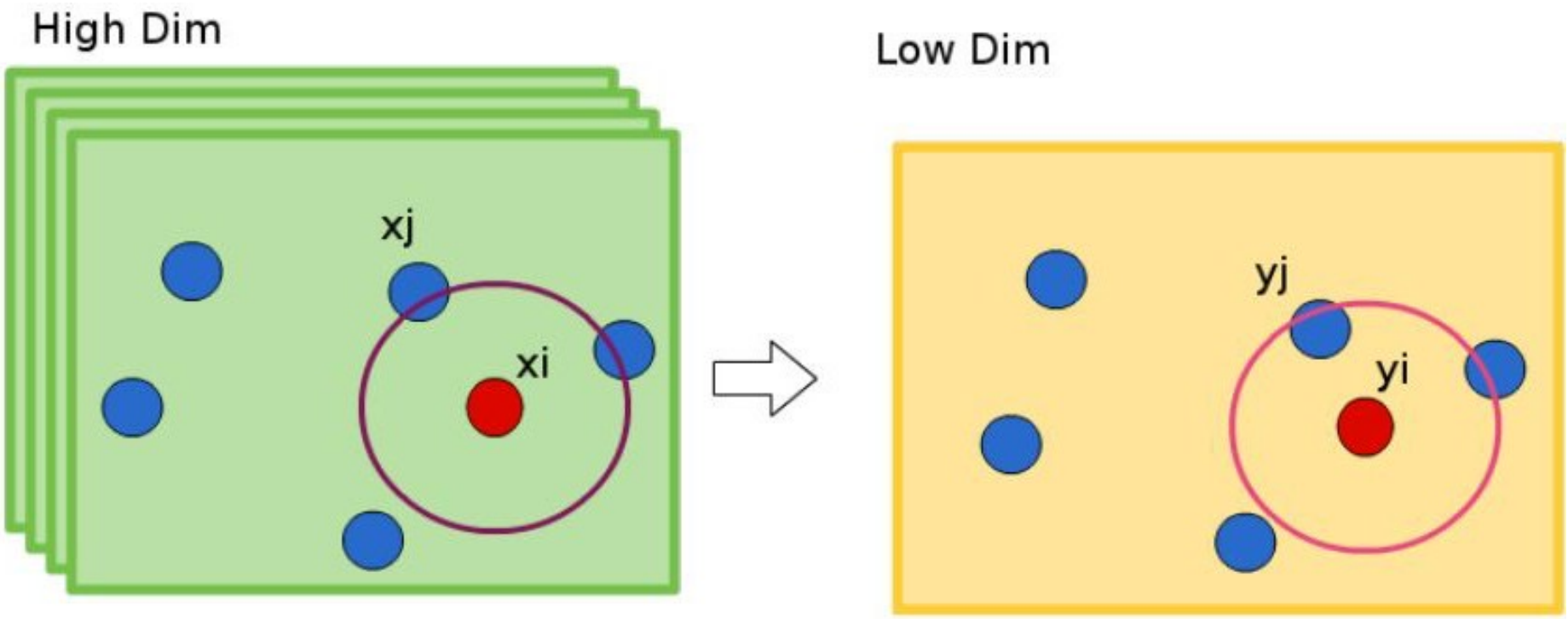
Patterns

- Discover natural clusters
- Linear relationships
- Visualize embeddings

Technical Requirements

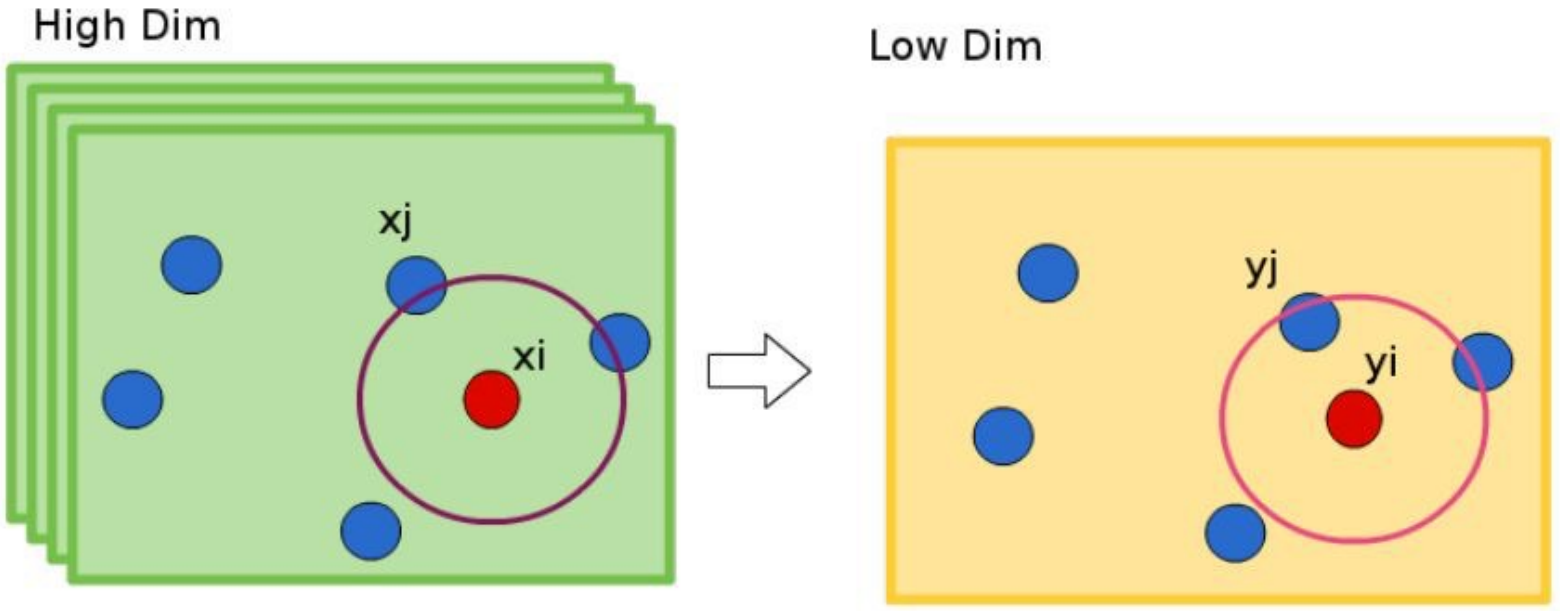
- Each high dimensional object is represented by a low-dimensional object
- Preserve the neighborhood
- Distant points correspond to dissimilar objects
- Scalability: large, high-dimensional data sets

Underlying idea of t-SNE



Stochastic Neighbor Embedding

Measure pairwise similarities between high-dimensional and low-dimensonal objects



$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i^2)}$$

Stochastic Neighbor Embedding

Converting the high-dimensional Euclidean distances into conditional probabilities that represent similarities

- Similarity of datapoints in High Dimension

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i^2)}$$

- Similarity of datapoints in Low Dimension

$$q_{j|i} = \frac{exp(-||y_i - y_j||^2)}{\sum_{k \neq i} exp(-||y_i - y_k||^2)}$$

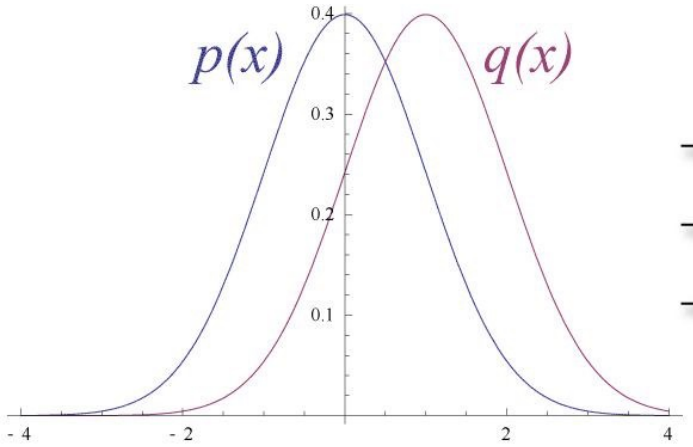
- Cost function

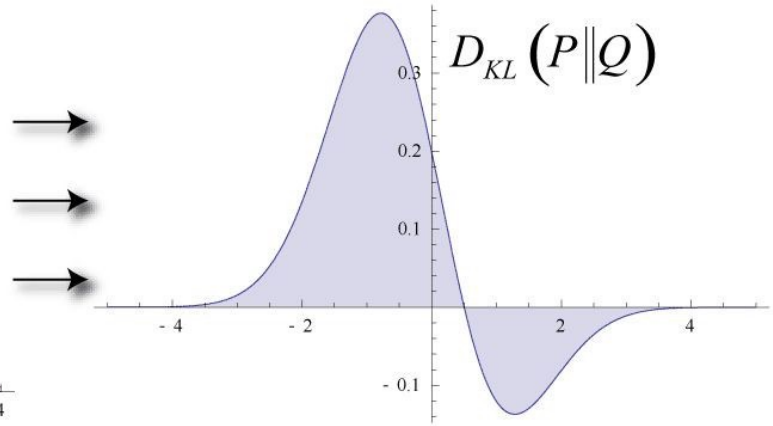
$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} log \frac{p_{j|i}}{q_{j|i}}$$

KL Divergence

Measures the similarity between two probability distributions

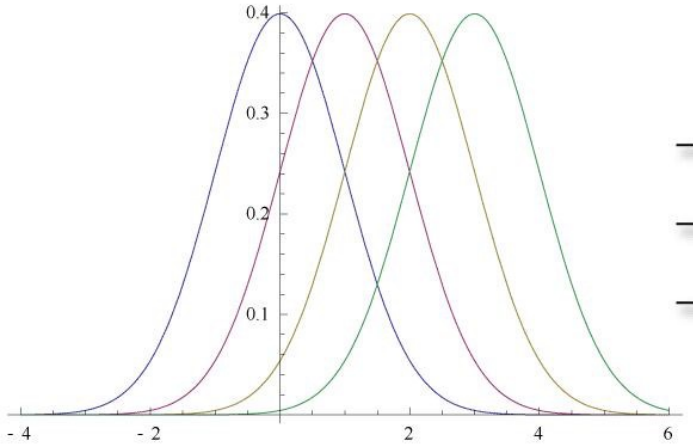
It is asymmetric

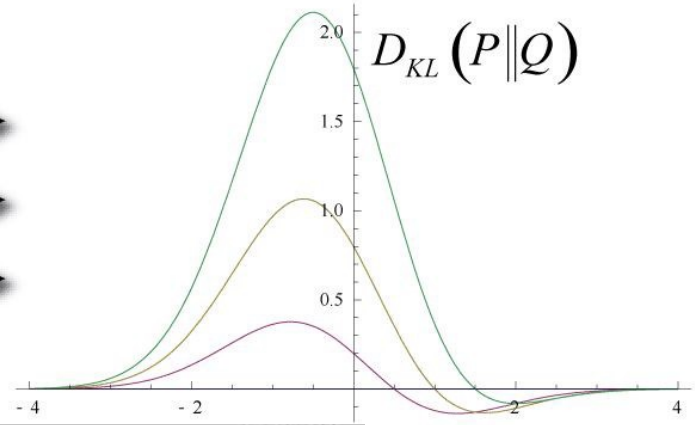




Original Gaussian PDF's

KL Area to be Integrated





$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i)\,\log\frac{P(i)}{Q(i)}.$$

Stochastic Neighbor Embedding

Gradient has a surprisingly simple form

$$\frac{\partial C}{\partial y_i} = \sum_{j \neq i} (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j)$$

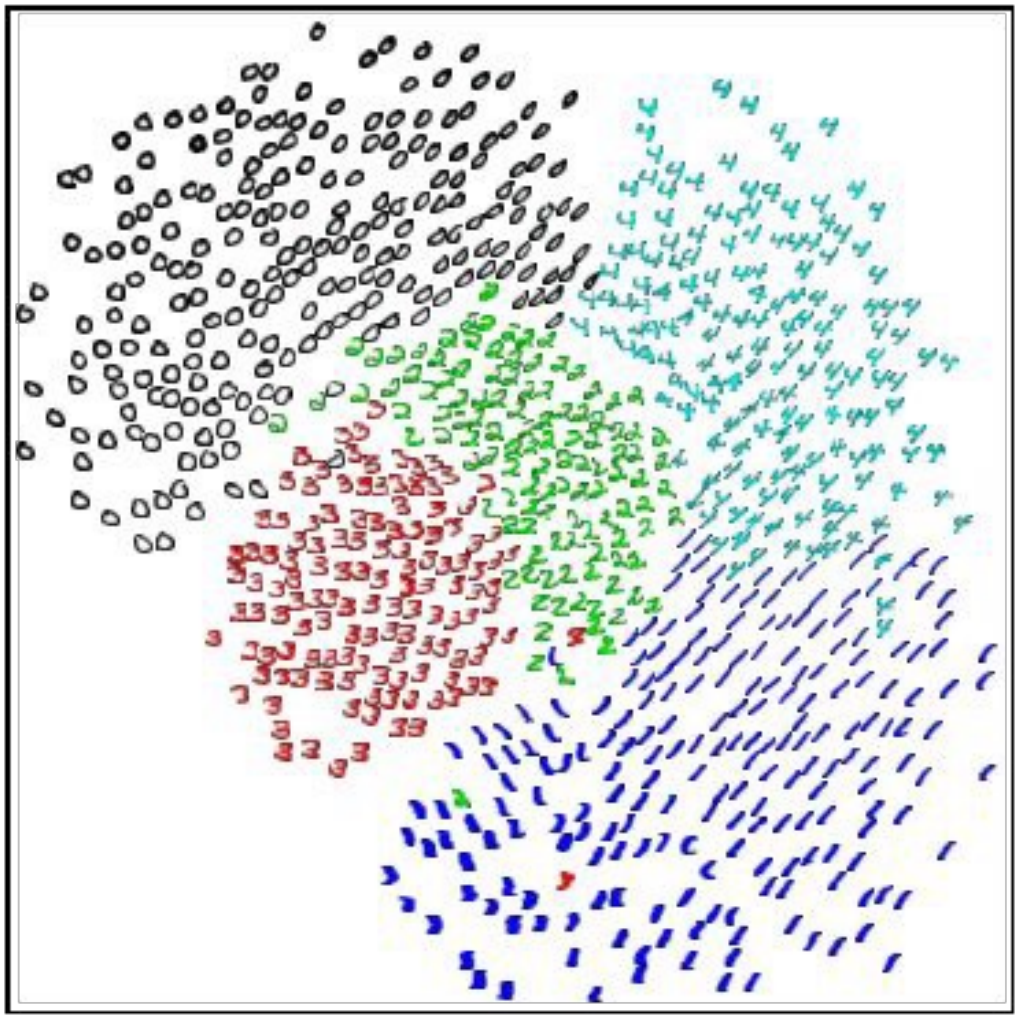
The gradient update with momentum term is given by

$$Y^{(t)} = Y^{(t-1)} + \eta \frac{\partial C}{\partial y_i} + \beta(t)(Y^{(t-1)} - Y^{(t-2)})$$

Stochastic Neighbor Embedding

The result of running the SNE algorithm on 3000 256-dimensional grayscale images of handwritten digits.

The classes are quite well separated even though SNE had no information about class labels. Furthermore, within each class, properties like orientation, skew and stroke thickness tend to vary smoothly across the space.



Symmetric SNE

Such that $p_{ij} = p_{ji}, q_{ij} = q_{ji}$, the main advantage is simplifing the gradient

$$\frac{\partial C}{\partial y_i} = 2 \sum_j (p_{ij} - q_{ij})(y_i - y_j)$$

However, in practice we symmetrize (or average) the conditionals

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

Set the bandwidth σ_i such that the conditional has a fixed perplexity (effective number of neighbors) $Perp(P_i) = 2^{H(P_i)}$, typical value is about 5 to 50

t-Distribution

Use heavier tail distribution than Gaussian in low-dim space, we choose

$$q_{ij} \propto (1 + ||y_i - y_j||^2)^{-1}$$

Then the gradient could be

$$\frac{\partial C}{\partial y_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij})(1 + ||y_i - y_j||^2)^{-1}(y_i - y_j)$$

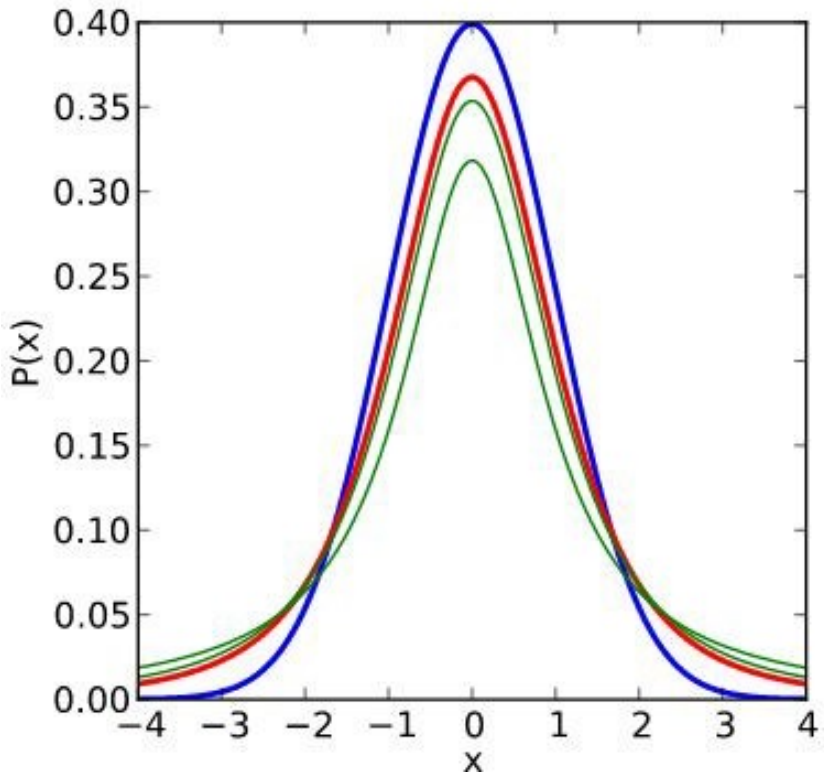
Why Student-t Distribution?

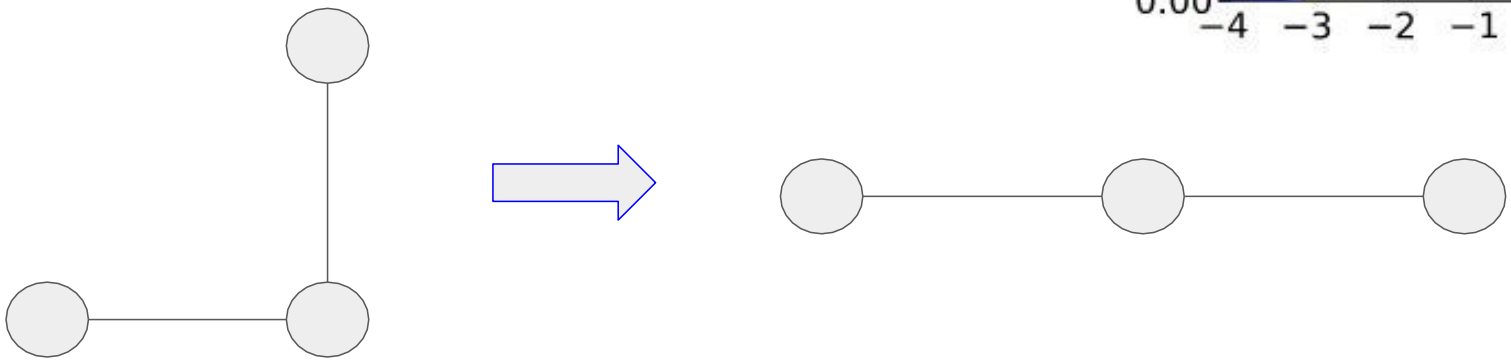
Why do we define map similarities as $q_{ij} \propto (1 + ||y_i - y_j||^2)^{-1}$

Suppose data is intrinsically high dimensional

We try to model the local structure of this data in the map

Result: Dissimilar points have to be modeled as too far apart in the map!





t-Distributed Stochastic Neighbor Embedding

- Similarity of datapoints in High Dimension

$$p_{ij} = \frac{exp(-||x_i - x_j||^2/2\sigma^2)}{\sum_{k \neq l} exp(-||x_l - x_k||^2/2\sigma^2)}$$

- Similarity of datapoints in Low Dimension

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}}$$

t-Distributed Stochastic Neighbor Embedding

- Cost function

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}}$$

  - Large p_{ij} modeled by small q_{ij}: Large penalty
  - Small p_{ij} modeled by large q_{ij}: Small penalty
  - t-SNE mainly preserves local similarity structure of the data

- Gradient

$$\frac{\partial C}{\partial y_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij})(1 + ||y_i - y_j||^2)^{-1}(y_i - y_j)$$

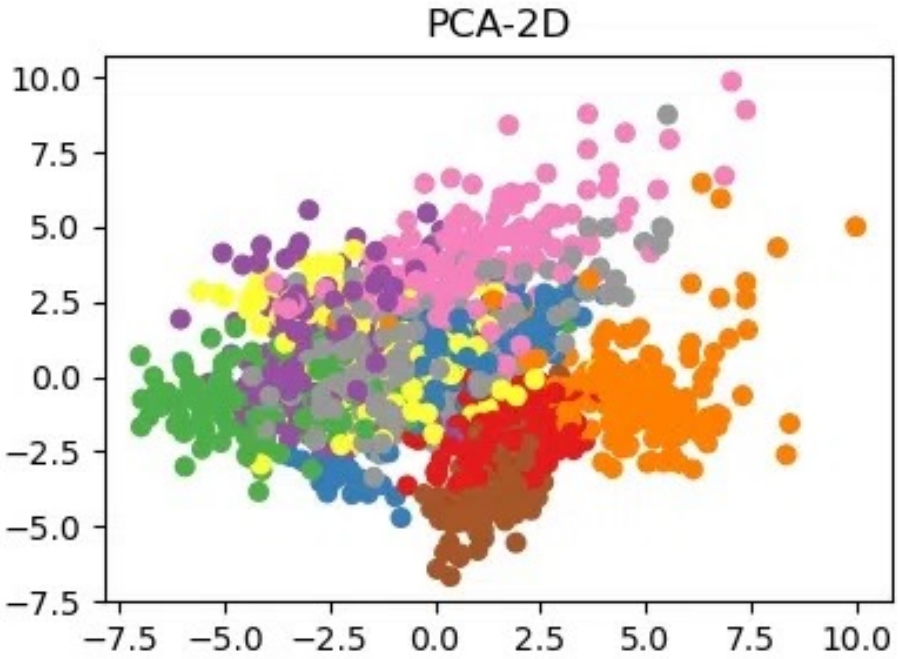
t-SNE Algorithm

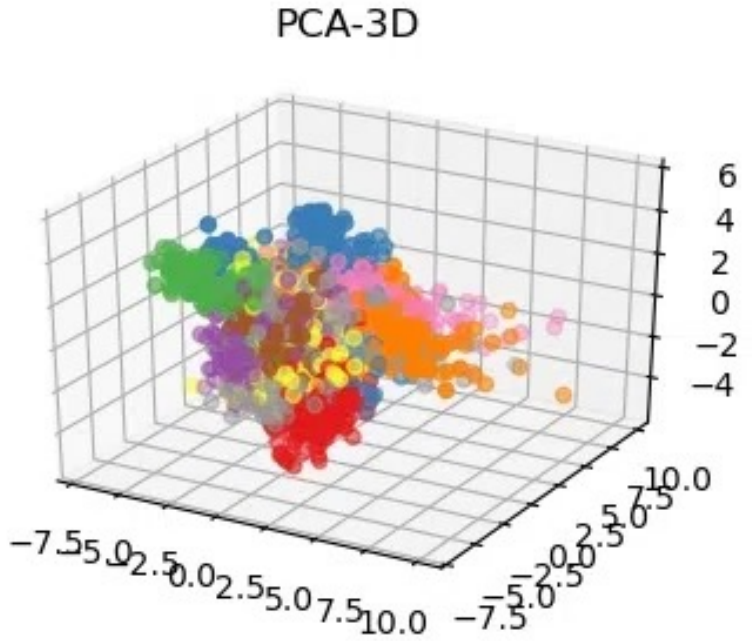
Algorithm 1: Simple version of t-Distributed Stochastic Neighbor Embedding.

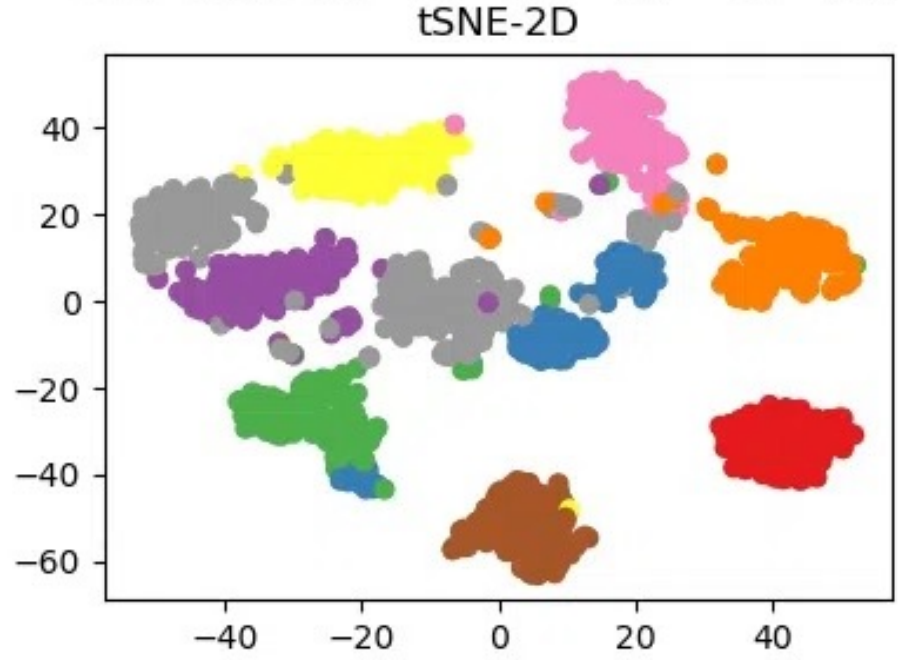
Data: data set $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$,
cost function parameters: perplexity *Perp*,
optimization parameters: number of iterations T, learning rate η, momentum $\alpha(t)$.
Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, \ldots, y_n\}$.
begin
  compute pairwise affinities $p_{j|i}$ with perplexity *Perp* (using Equation 1)
  set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$
  sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, \ldots, y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$
  for t=*1* **to** T **do**
    compute low-dimensional affinities q_{ij} (using Equation 4)
    compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (using Equation 5)
    set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)\left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right)$
  end
end

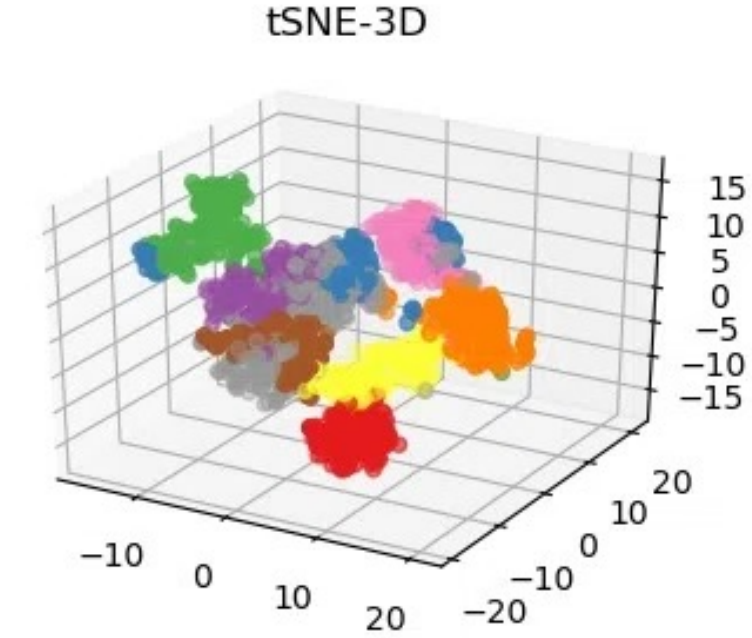
sklearn.manifold.TSNE

Results: MNIST



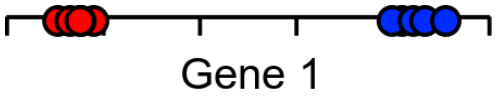




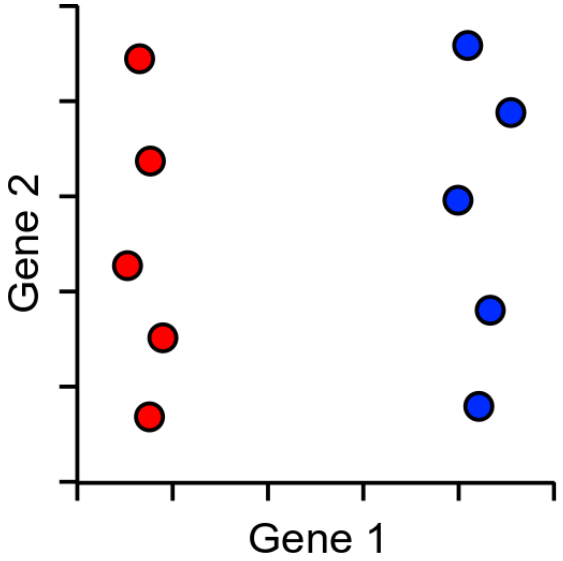


So t-SNE is great then?

- Kind of…
- Imagine a dataset with only one super informative gene



Distance within cluster = low
Distance between clusters = high



Distance within cluster = higher
Distance between clusters = lower

Now 3 genes
Now 3,000 genes

Everything is the same distance from everything

So everything sucks?

PCA

Requires more than 2 dimensions
Thrown off by quantised data
Expects linear relationships

tSNE

Can't cope with noisy data
Loses the ability to cluster

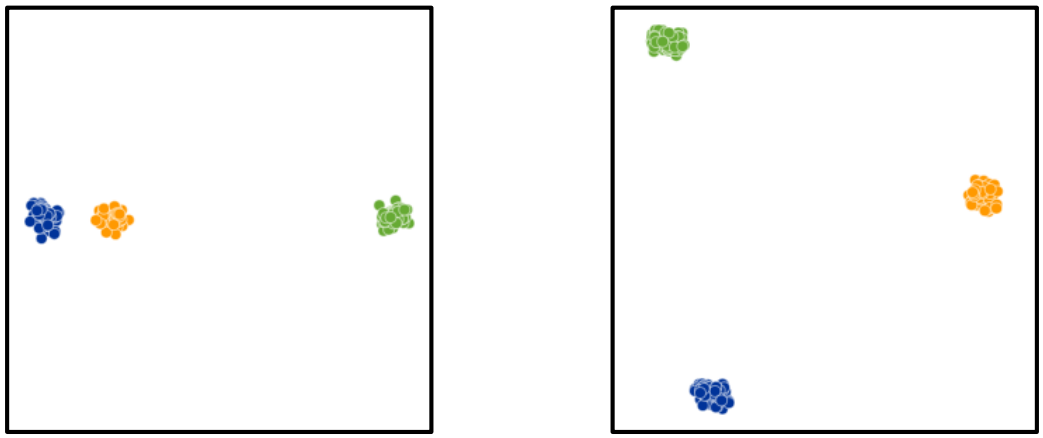
Answer: Combine the two methods, get the best of both worlds

- PCA
  - Good at extracting signal from noise
  - Extracts informative dimensions
- tSNE
  - Can reduce to 2D well
  - Can cope with non-linear scaling

This is what CellRanger does in its default analysis

So PCA + tSNE is great then?

- Kind of…
  - t-SNE is slow. This is probably it's biggest crime
    - t-SNE doesn't scale well to large numbers of cells (10k+)
  - T-SNE only gives reliable information on the closest neighbours large distance information is almost irrelevant



UMAP: Uniform Manifold Approximation and Projection

UMAP is a replacement for tSNE to fulfil the same role

Conceptually very similar to tSNE, but with a couple of relevant (and somewhat technical) changes

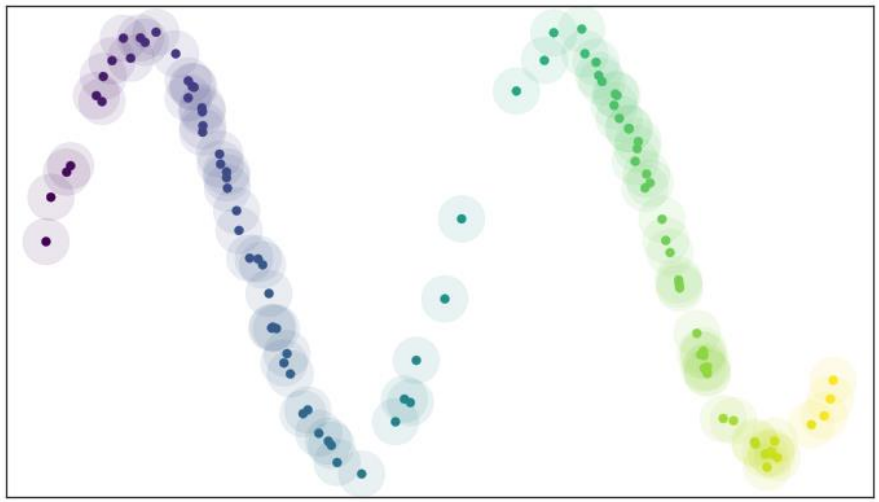
Practical outcome is:

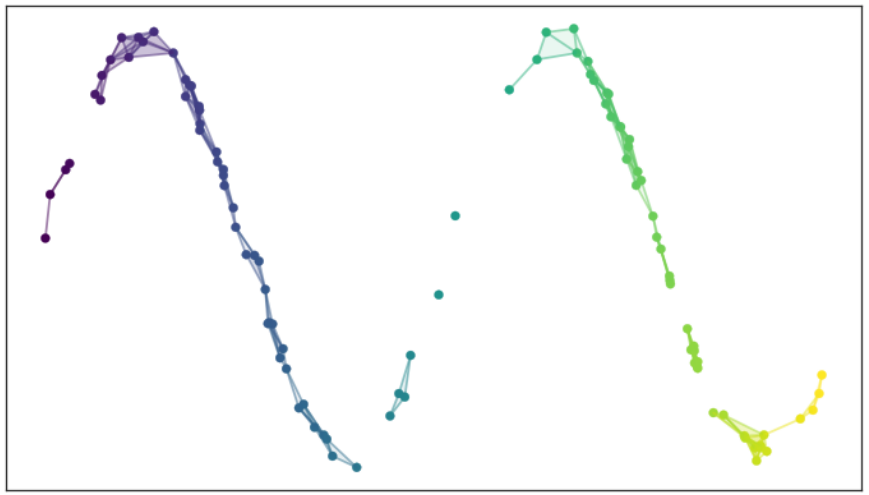
- UMAP is quite a bit quicker than tSNE
- UMAP can preserve more global structure than tSNE*
- UMAP can run on raw data without PCA preprocessing*
- UMAP can allow new data to be added to an existing projection

https://umap-learn.readthedocs.io/en/latest/

Uniform Manifold Approximation and Projection (UMAP)

- Starts with a ‘fuzzy simplicial complex’ but ends with a simple weighted neighbourhood graph





Constructing weighted neighborhood graph

For each point *i,* define distance to the nearest neighbour

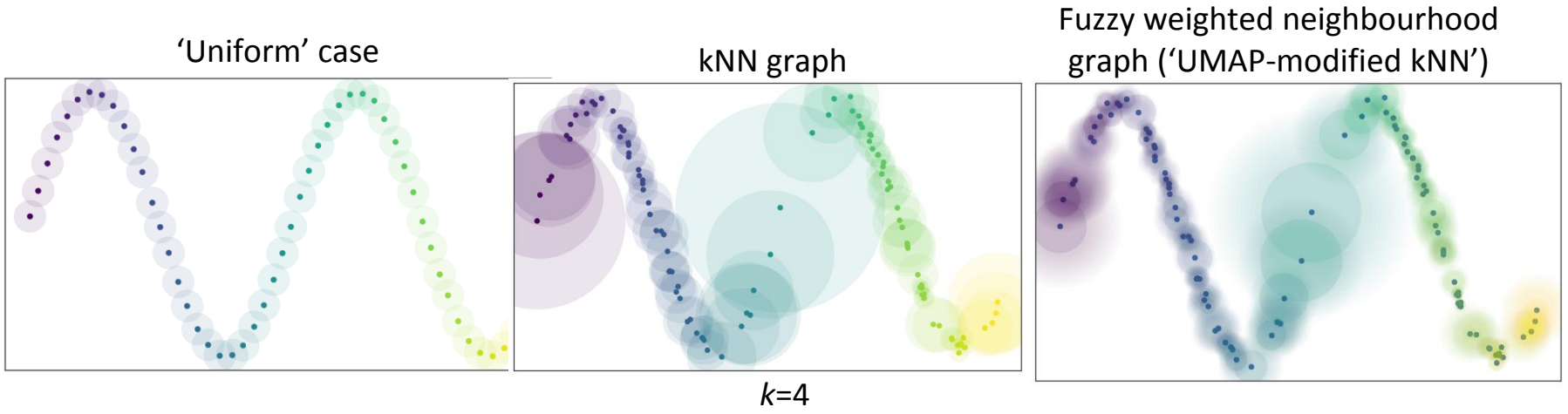
$$\rho_i = \min\{d(x_i, x_{i_j}) \mid 1 \leq j \leq k, d(x_i, x_{i_j}) > 0\},$$

Define σ_i using the following equation (local dispersion)

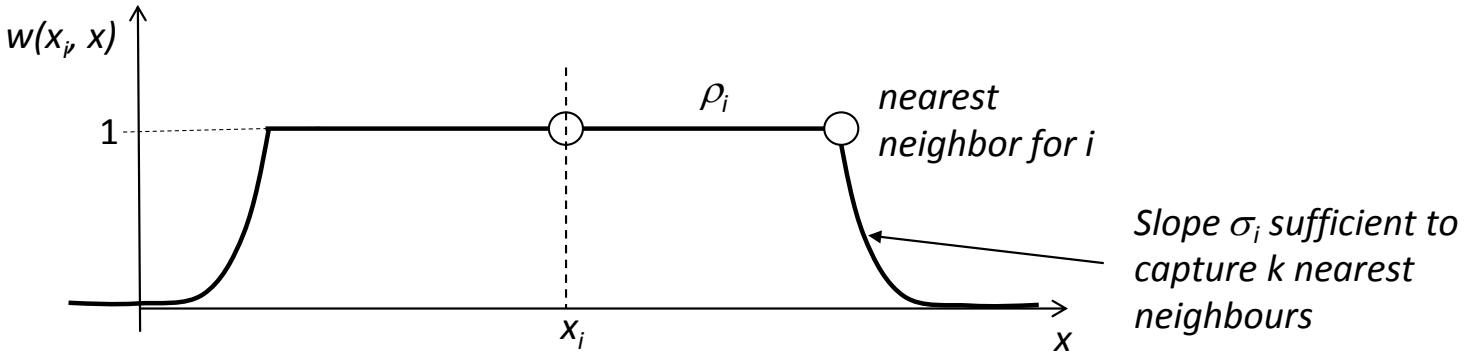
$$\sum_{j=1}^{k} \exp\left(\frac{-\max(0, d(x_i, x_{i_j}) - \rho_i)}{\sigma_i}\right) = \log_2(k)$$

Weight of the neighbourhood graph

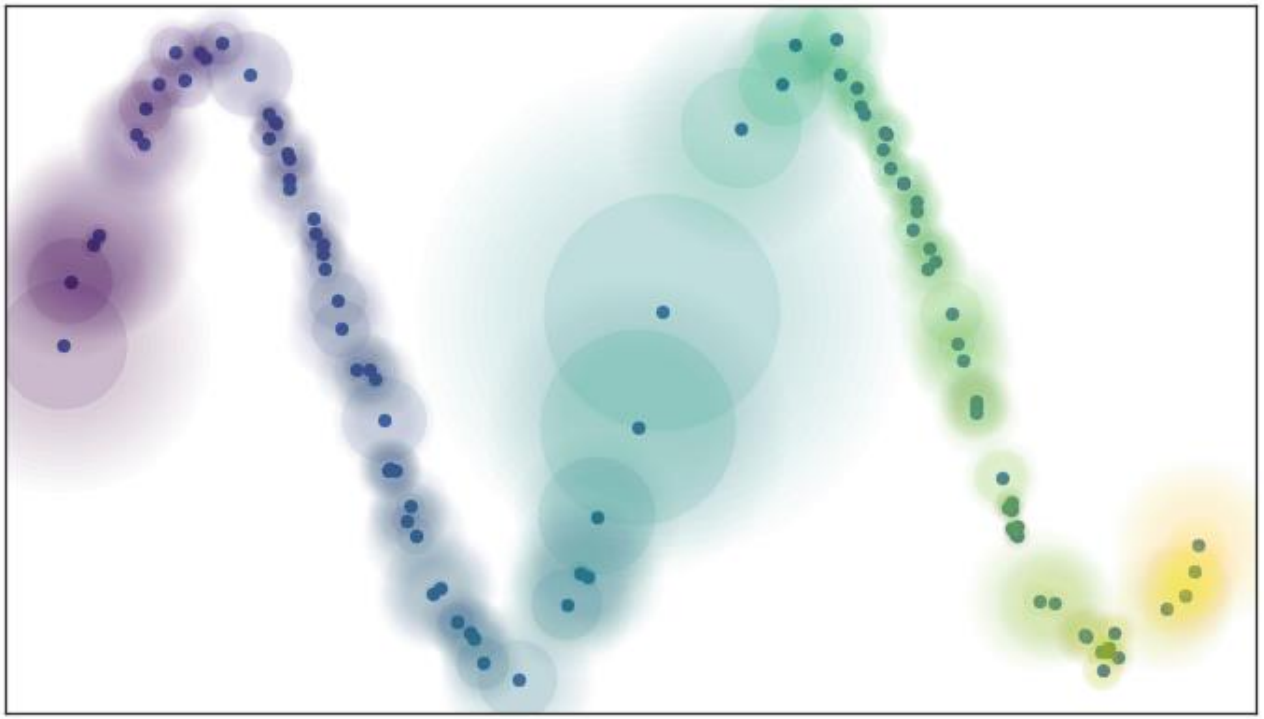
$$w((x_i, x_{i_j})) = \exp\left(\frac{-\max(0, d(x_i, x_{i_j}) - \rho_i)}{\sigma_i}\right)$$



$$w((x_i, x_{i_j})) = \exp\left(\frac{-\max(0, d(x_i, x_{i_j}) - \rho_i)}{\sigma_i}\right)$$



Fuzzy weighted neighbourhood graph ('UMAP-modified kNN')



Minimizing ‘cross-entropy’

Preservation of small distances

Preservation of large distances

$$\sum_{e \in E} \left[w_h(e) \log \left(\frac{w_h(e)}{w_l(e)} \right) + (1 - w_h(e)) \log \left(\frac{1 - w_h(e)}{1 - w_l(e)} \right) \right]$$

$1 \geq w_h > 0$ – weights in high-dimensional space

$1 \geq w_l > 0$ – weights in low-dimensional space

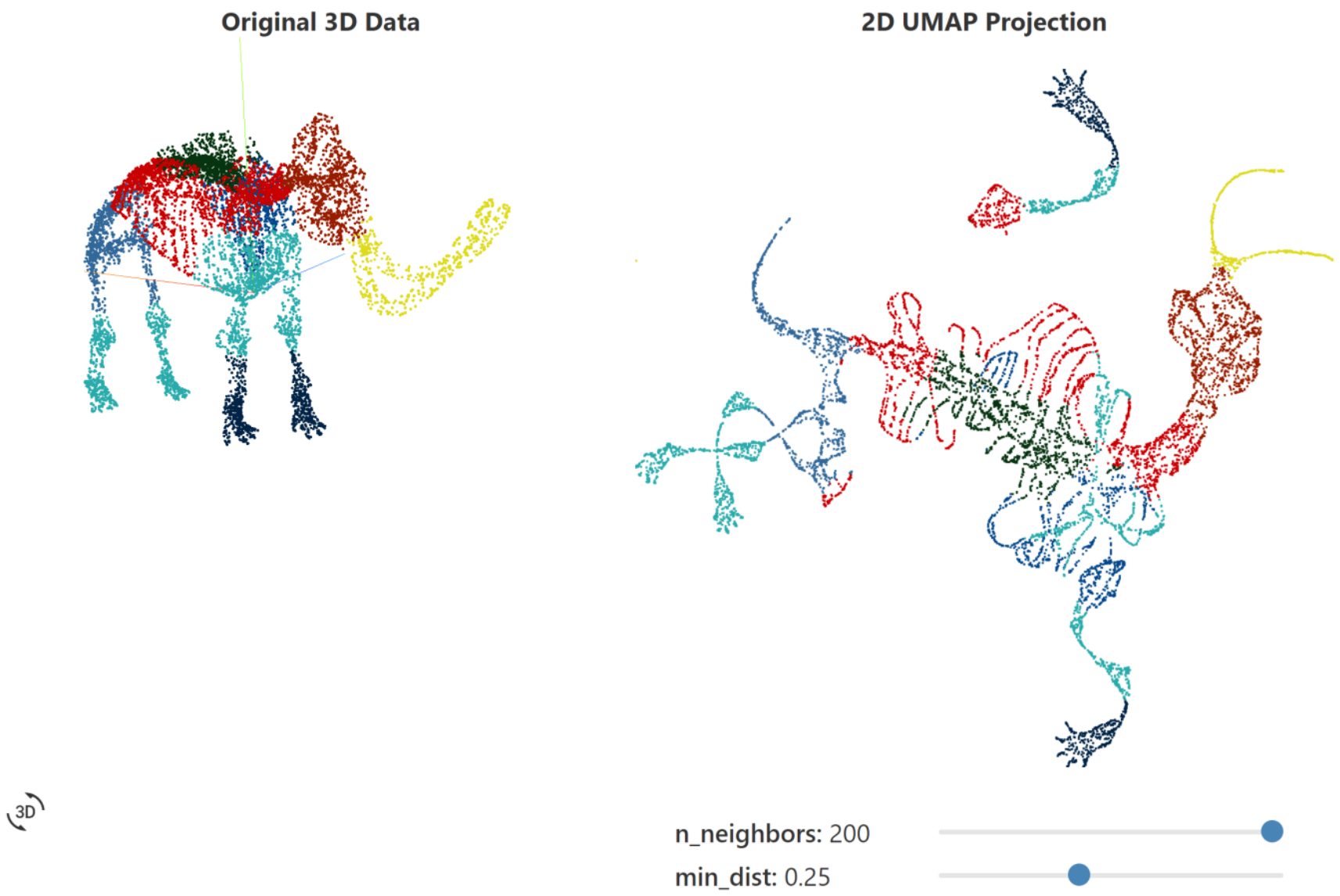
Actual algorithm is very close to force-directed layout algorithm

Hyperparameters of UMAP

- Min-dist: minimal distance between points in low-dimensional space
- N_neighbours: k in the kNN graph

https://pair-code.github.io/understanding-umap/supplement.html

Toy example to play with parameters: *Projecting a mammoth from 3D to 2D*



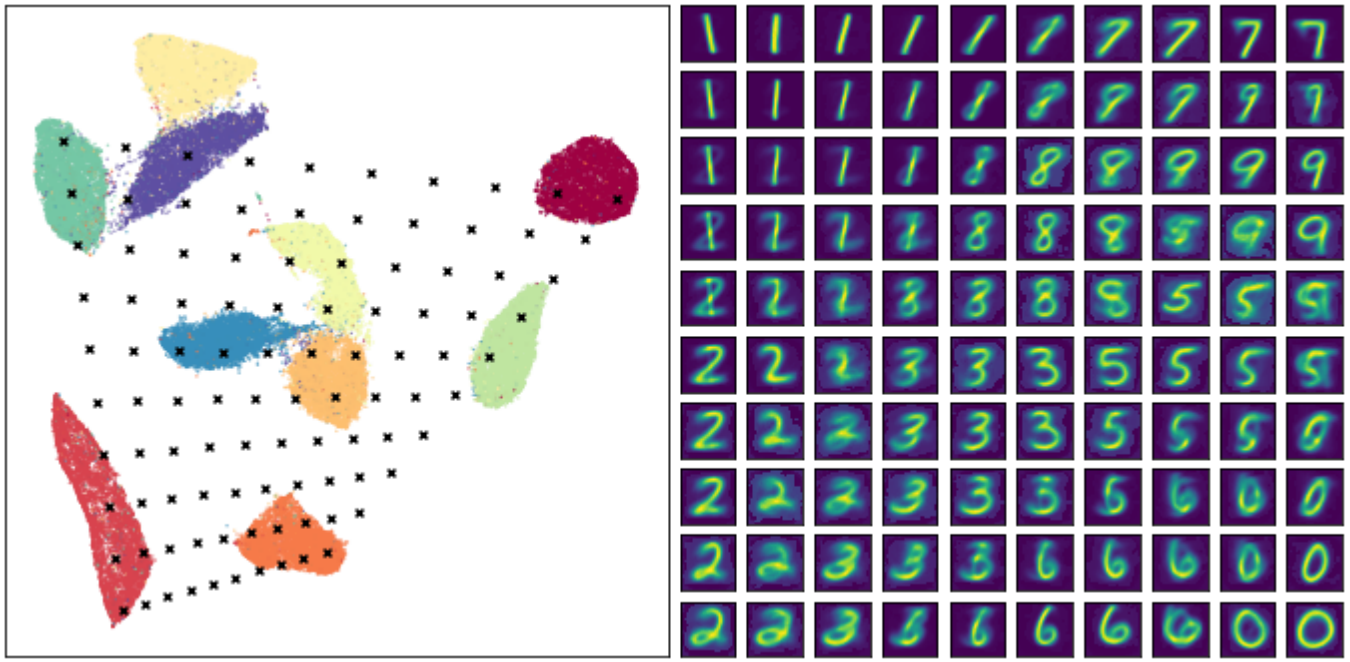
https://pair-code.github.io/understanding-umap/

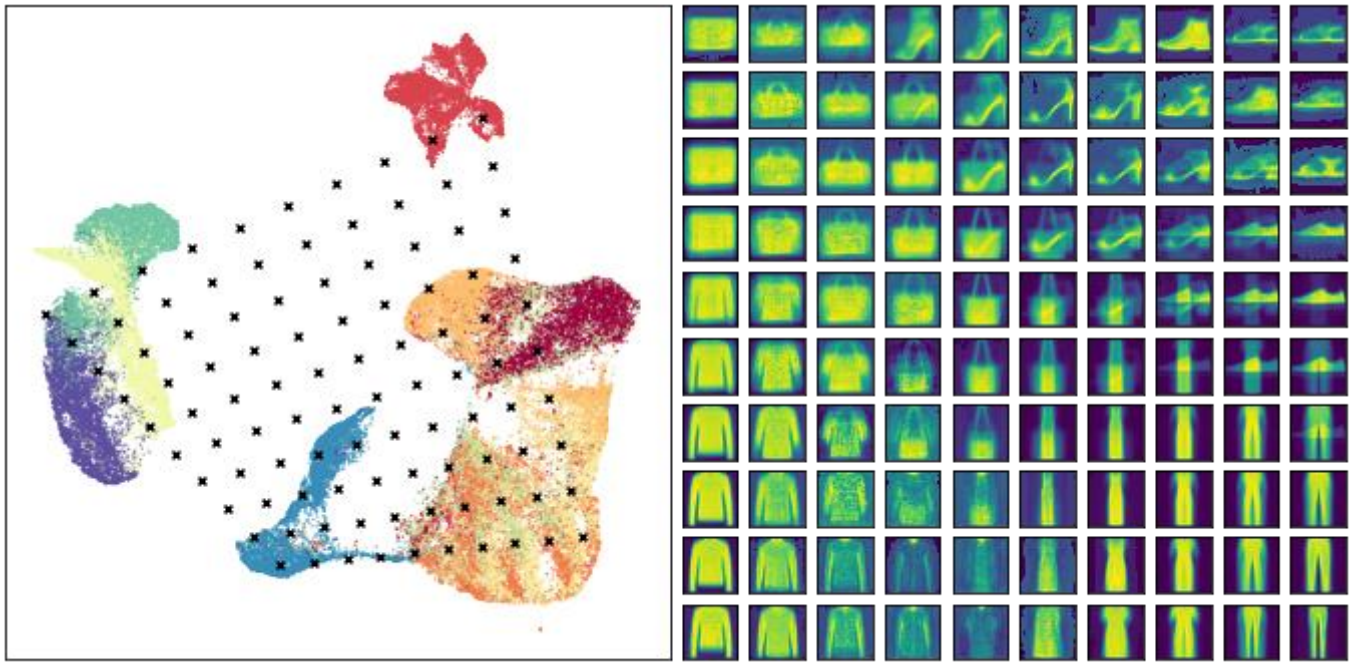
https://duhaime.s3.amazonaws.com/apps/umap-zoo/

UMAP ‘inverse transform’:

https://umap-learn.readthedocs.io/en/latest/inverse_transform.html

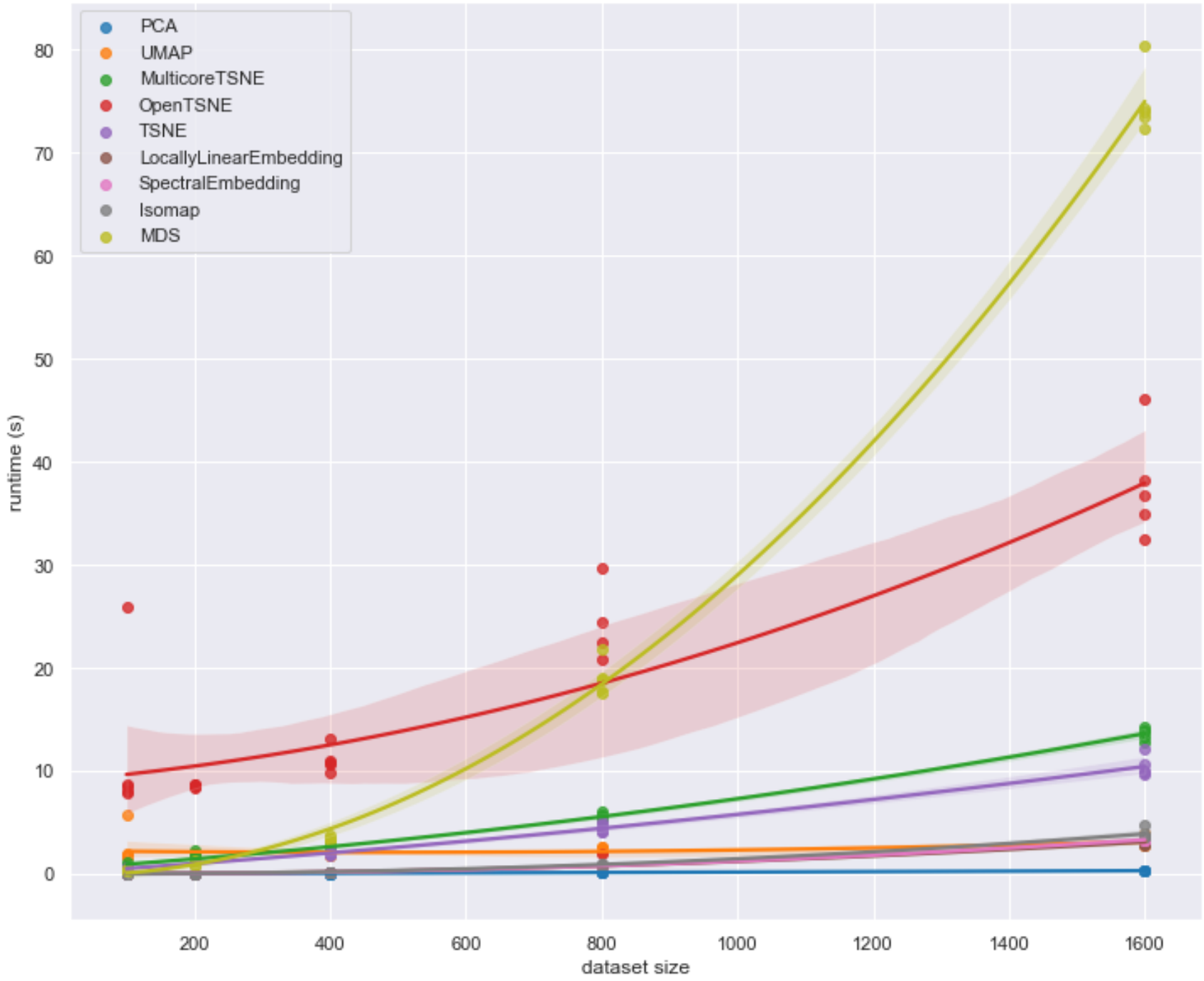
provide a sort of DECODE/INJECT function





Comparing tSNE and UMAP

- Speed



Comments to both t-SNE and UMAP methods

- Hyperparameters really matter
- Cluster sizes in a UMAP plot mean nothing
- Distances between clusters might not mean anything
- Random noise doesn’t always look random
- You may need more than one plot
- For large ‘neighbourhood’ parameters, both methods give results similar to Multi-dimensional scaling
- Both can work with non-Euclidean metrics in R^p

Minimum Distortion Embedding (MDE)

Agrawal, Ali and Boyd, ***arXiv 2021***

- "Modern" version of metric MDS
- Avoids computing the complete distance matrix
- Quadratic MDE ($f_{ij}(d_{ij}) = w_{ij}d_{ij}^2$, $\boldsymbol{w_{ij}}$ represents similarity between $\boldsymbol{i}$ and $\boldsymbol{j}$) can be reduced to an eigenproblem
- pyMDE Python package

f_{ij}: Distortion function between i and j.

d_{ij}: Embedding distance between X_i and X_j.

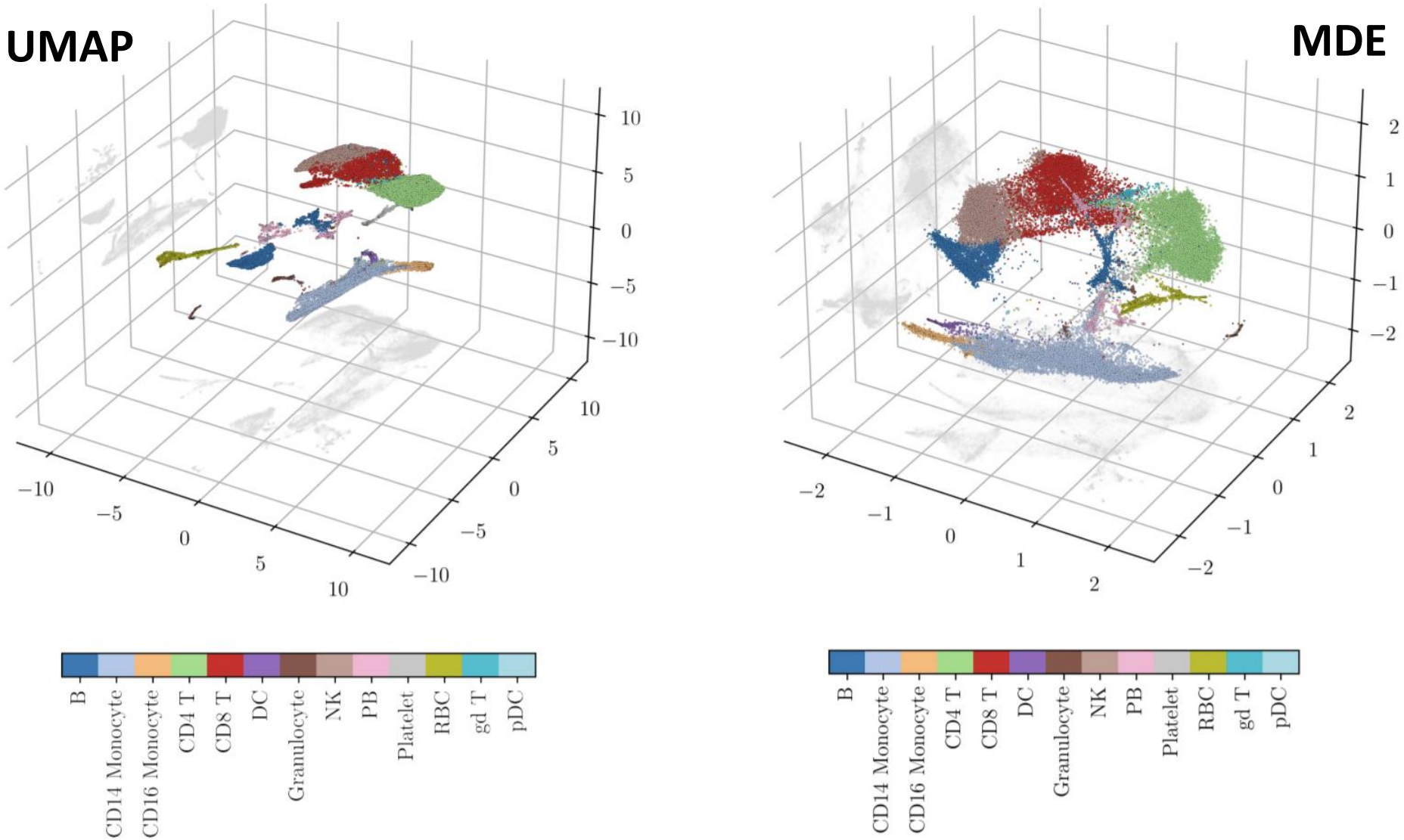
$$\mathbf{E}(X) = \frac{1}{|E|} \sum_{(i,j) \in E} f_{ij}(d_{ij})$$

X: Objective embedding

E: All sample pairs if tractable, otherwise a set of edges

Of course, for positive f_{ij} the problem is trivial. The authors propose several constraints (e.g. standardized) to avoid $X = 0$.

Usage example: RNA-seq dataset, $n > 40{,}000$



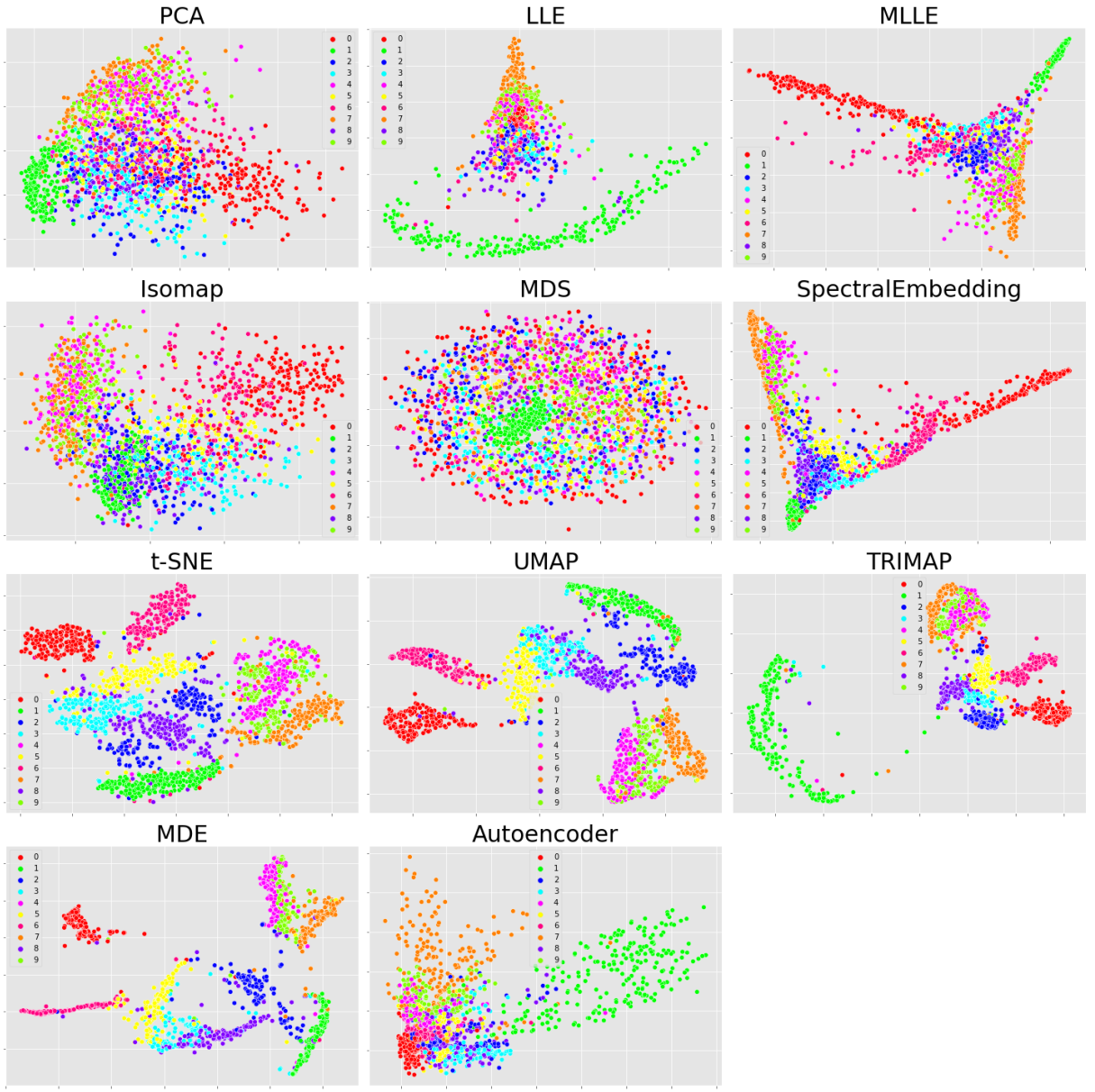
Comparison of (many) methods:

https://colab.research.google.com/drive/1miQpnYAa9pZa-YWng1C7V78hM-nPR8Ly?usp=sharing

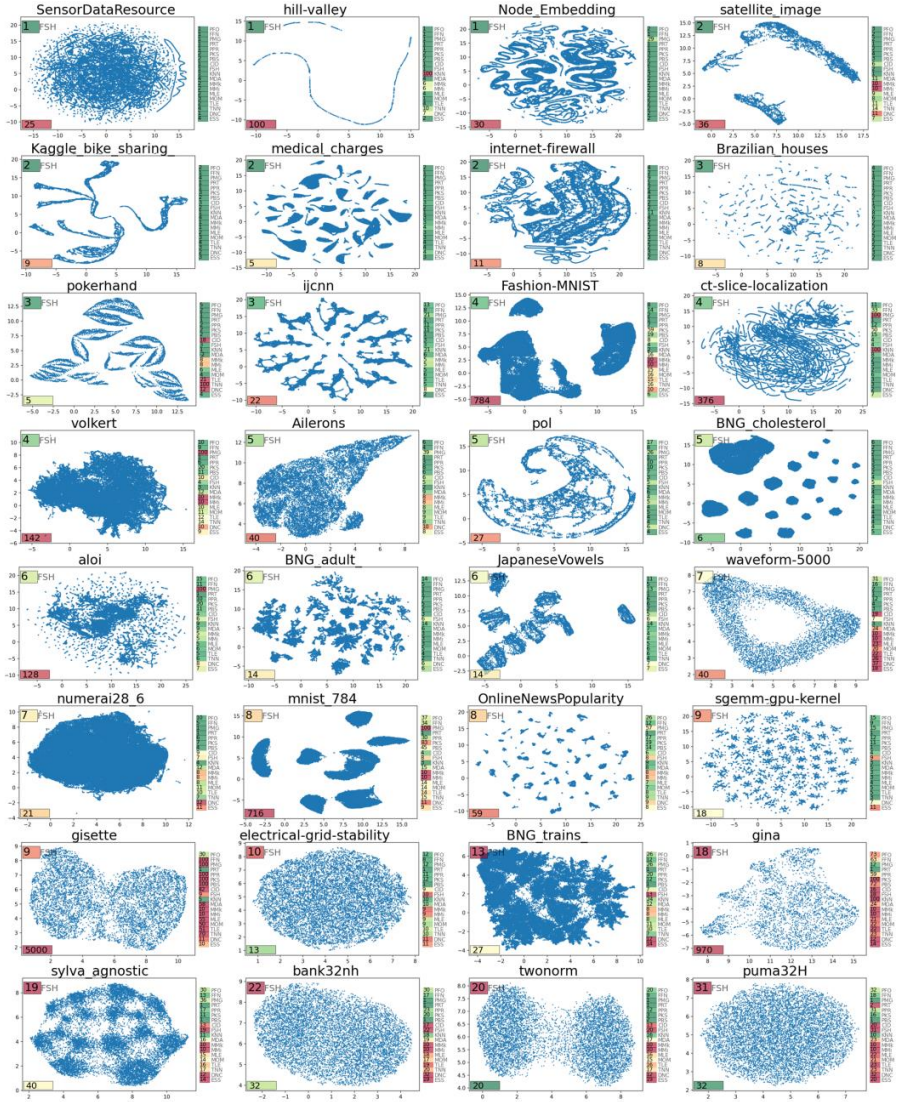
viz_results =
apply_panel_of_manifold_learning_methods(X,color, methods_to_apply=['PCA','UMAP','TRIMAP','MDE','TSNE',
'LLE','MLLE','ISOMAP','MDS','SE', 'AUTOENCODER'])

MNIST dataset (downsampled to 2000 points)

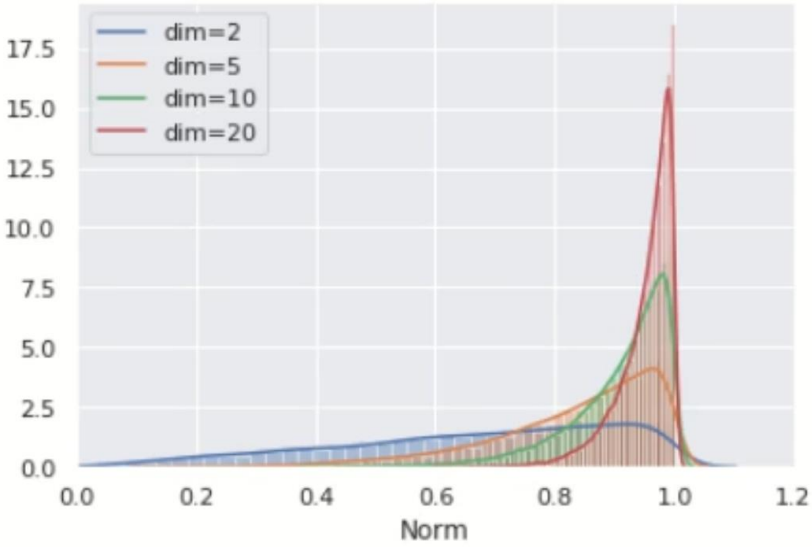
PCA: 0.19 sec
LLE: 9.9 sec
Modified LLE: 11 sec
Isomap: 11 sec
MDS: 11 sec
SpectralEmbedding: 8.1 sec
t-SNE: 21 sec
UMAP: 9.8 sec
TRIMAP: 2.6 sec
MDE: 2.3 sec
Autoencoder: 34 sec



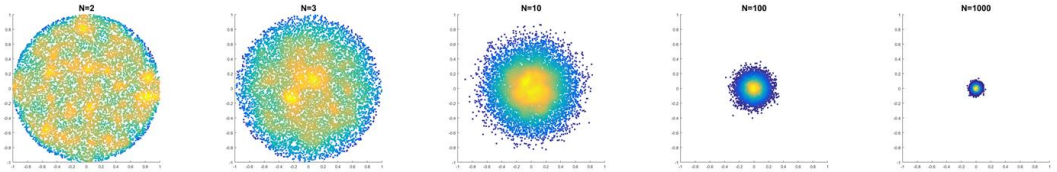
UMAP (or any other methods) become less informative in higher intrinsic dimensions



Some simple intuition for this



Uniformly sampled ball in R^N *observed in* R^2



Does not matter what distribution, it will look normal in any 2D or 3D projection (law of big numbers)

Practical approach
PCA + tSNE/UMAP

Filter heavily before starting

- Nicely behaving cells
- Expressed genes
- Variable genes

Do PCA

- Extract most interesting signal
- Take top PCs. Reduce dimensionality (but not to 2)

Do tSNE/UMAP

- Calculate distances from PCA projections
- Scale distances and project into 2-dimensions