CS5228: Knowledge Discovery and Data Mining

Lecture 7 — Dimensionality Reduction

Outline

- **Dimensionality Reduction**
  - Motivation
  - Naive approaches
- Dimensionality Reduction Techniques
  - PCA — Principal Component Analysis
  - LDA — Linear Discriminant Analysis
  - t-SNE — t-distributed Stochastic Neighbor Embedding

Dimensionality Reduction — Motivation

- High number of dimensions = high number of data features
  - m ("mass") — number of data points
  - V ("volume") — data space described by dimensions

$$\text{density } \rho = \frac{m}{V}$$

➜ High-dimensional data quickly becomes very sparse!

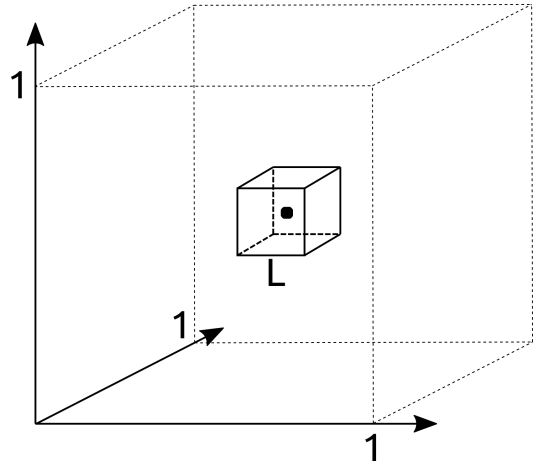
- Effects of many features and data sparsity
  - Higher computational cost
  - Problematic for any method that requires statistical significance ➜ high risk of overfitting
    ("good" data points and noise/outliers look more and more indistinguishable)
  - Obscures similarities between data points
    (points similar in low dimensions but not in high dimensions)

Curse of Dimensionality

- Effect of high dimensions (i.e., many features)
  - Data points tend to never be close together
  - Average distance between points converges

- Intuition
  - Assume *N* data points uniformly distributed within a unit cube with *d* dimensions
  - Let *L* be the length of the smallest cube containing the *k*-NN of a data point

$$L^d \approx \frac{k}{N} \Rightarrow L \approx \left(\frac{k}{N}\right)^{\frac{1}{d}}$$



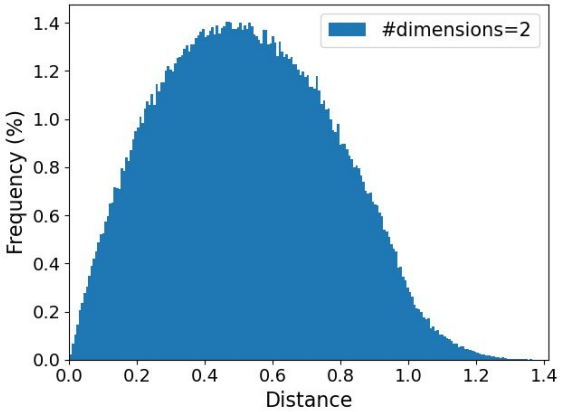
N=1,000, k=10

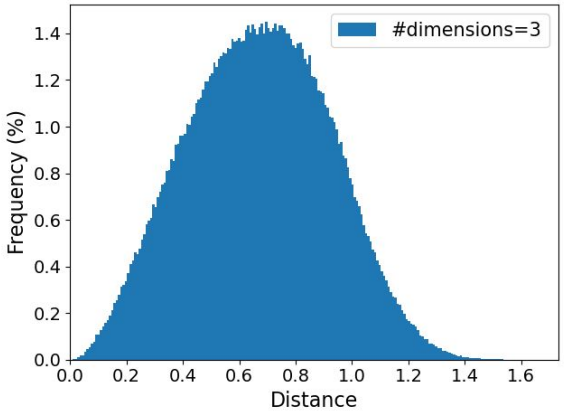
d	*L*
2	0.100
3	0.215
10	0.631
100	0.955
1,000	0.995
10,000	0.999

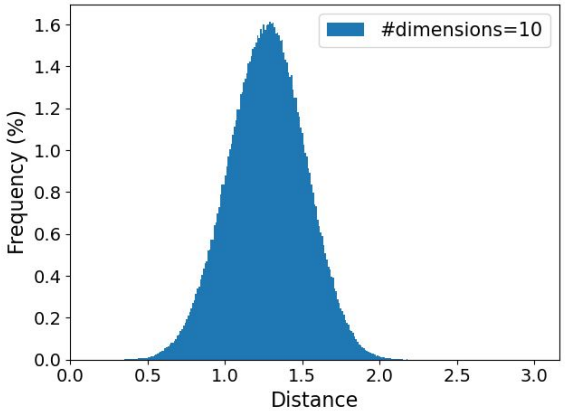
The cube with the k-NN is almost the whole unit cube!

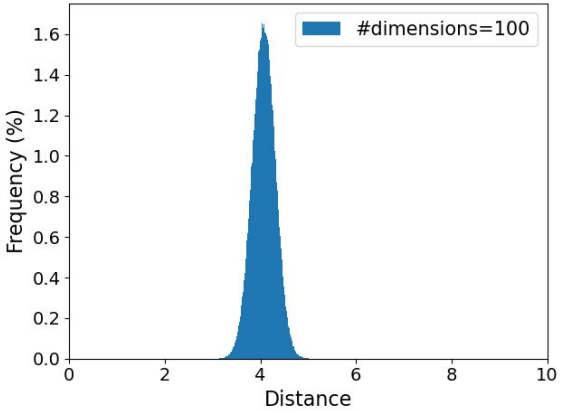
Curse of Dimensionality

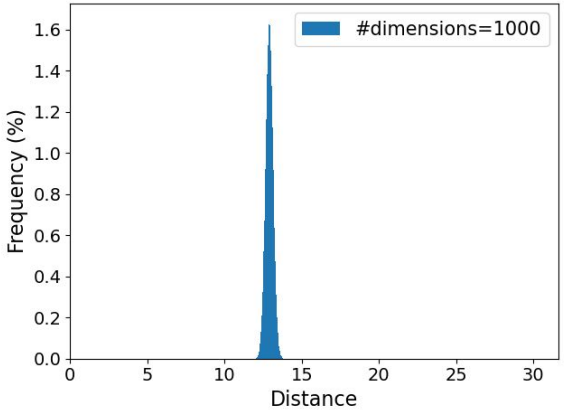
Distribution of pairwise distances between 1,000 random data points and different number of dimensions

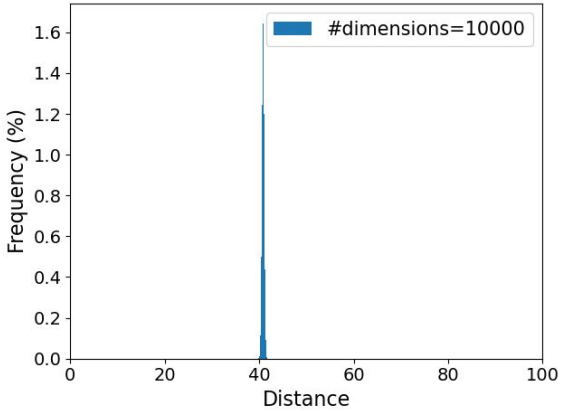












Dimensionality Reduction — Feature Selection

- Feature Selection — Removal of features before analysis
  (alternatively: keep only a certain subset of features)

- Common feature selection
  - Remove "unimportant" features based on expert knowledge
    (e.g., birth date of a person is unlikely to affect his/her spending behavior)
  - Remove features that might introduce ethical biases
    (e.g., sexual orientation and ethnicity for credit card approval)
  - Remove features with very low variance ➜ not useful
  - Remove features that are strongly correlated with other features ➜ not needed
    (basic method: calculate pairwise correlations and remove one of the features in case of high correlation)
  - Remove features with low ranks w.r.t. to their power to discriminate data points
    (basic approach of Decision Trees where feature near the root node yield purer subtreets)

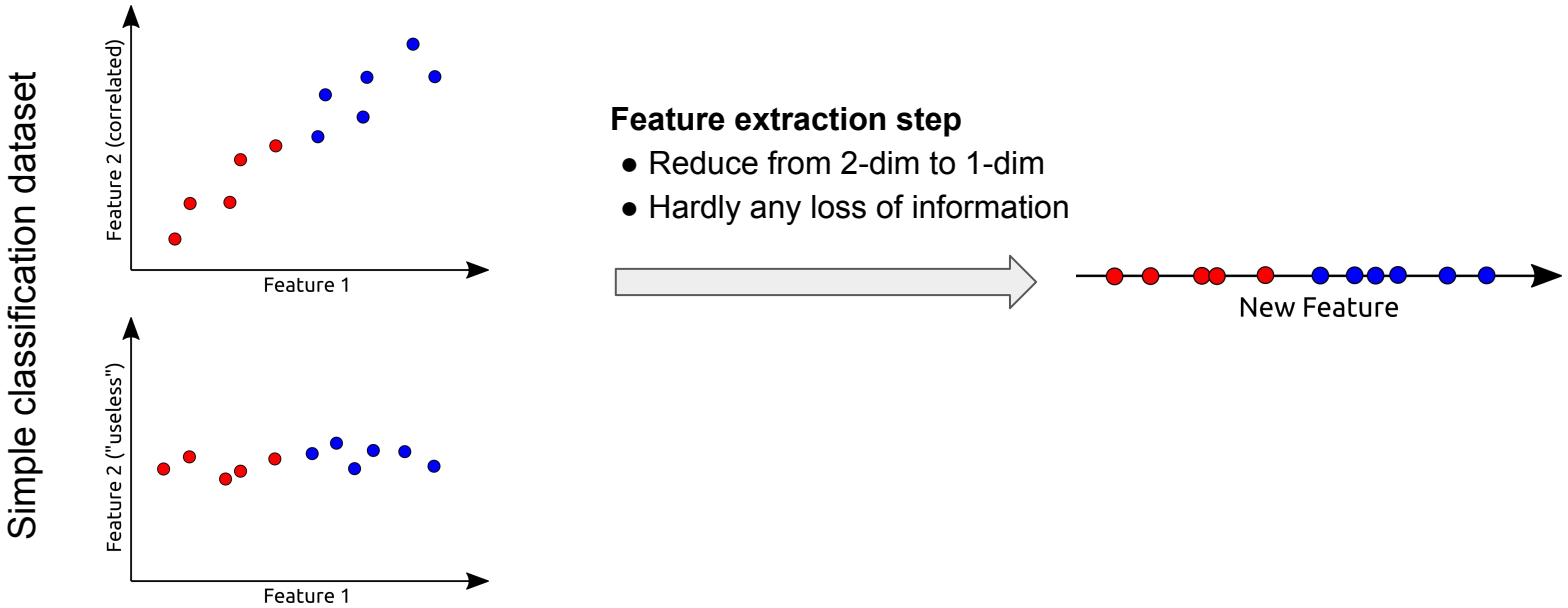
Feature Selection — Pros & Cons

- Pros
  - Relatively straightforward to implement
  - Does not change features themselves ➜ effects on analysis results are preserved

- Cons
  - Selecting important features based on expert knowledge often not trivial/obvious
  - Finding meaningful thresholds — e.g., min. variance or correlation — not obvious

Dimensionality Reduction — Feature Extraction

- Feature Extraction — basic idea
  - Generate new features as form of summary of original features
  - Feature extraction algorithms utilize discriminatory power and correlations among features



Outline

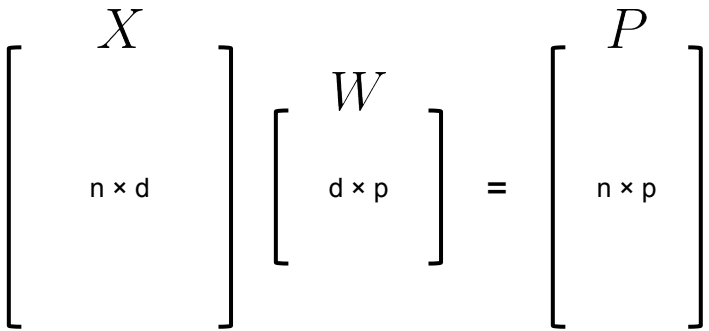
- Dimensionality Reduction
  - Motivation
  - Naive approaches

- **Dimensionality Reduction Techniques**
  - **PCA — Principal Component Analysis**
  - LDA — Linear Discriminant Analysis
  - t-SNE — t-distributed Stochastic Neighbor Embedding

Principal Component Analysis (PCA)

- PCA — Dimensionality reduction through linear transformation
  - New output features are a linear combination of original input features
  - Transforms data to a new coordinate systems
  - Unsupervised approach (independent from any kind of class labels)

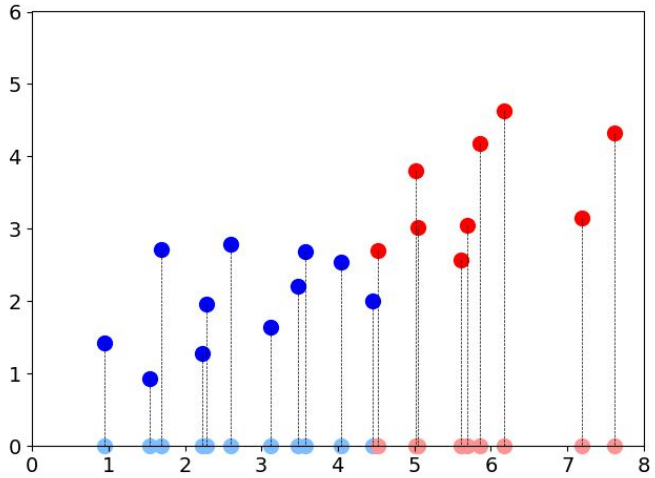
- Basic setup
  - Dataset *X* (*n* samples, *d* features)
  - Find matrix *W* to transform *X* into a *p*-dimensional dataset (*p* < *d*)



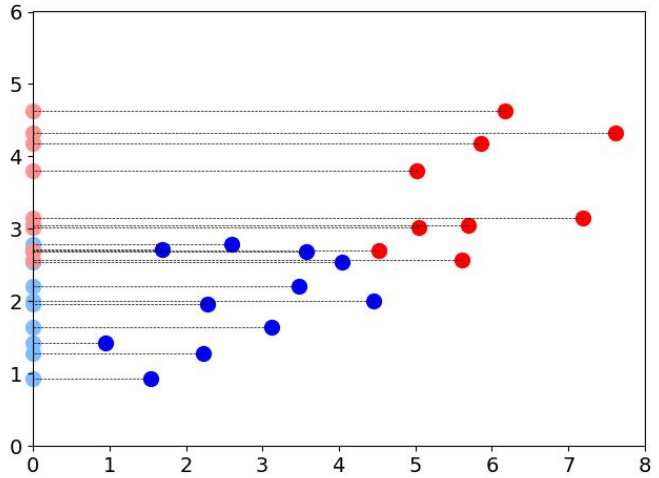
➜ What makes a good transformation matrix *W* and how to find it?

PCA — Intuition Using Naive Transformations

Mapping of data to x-axis: $W = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$



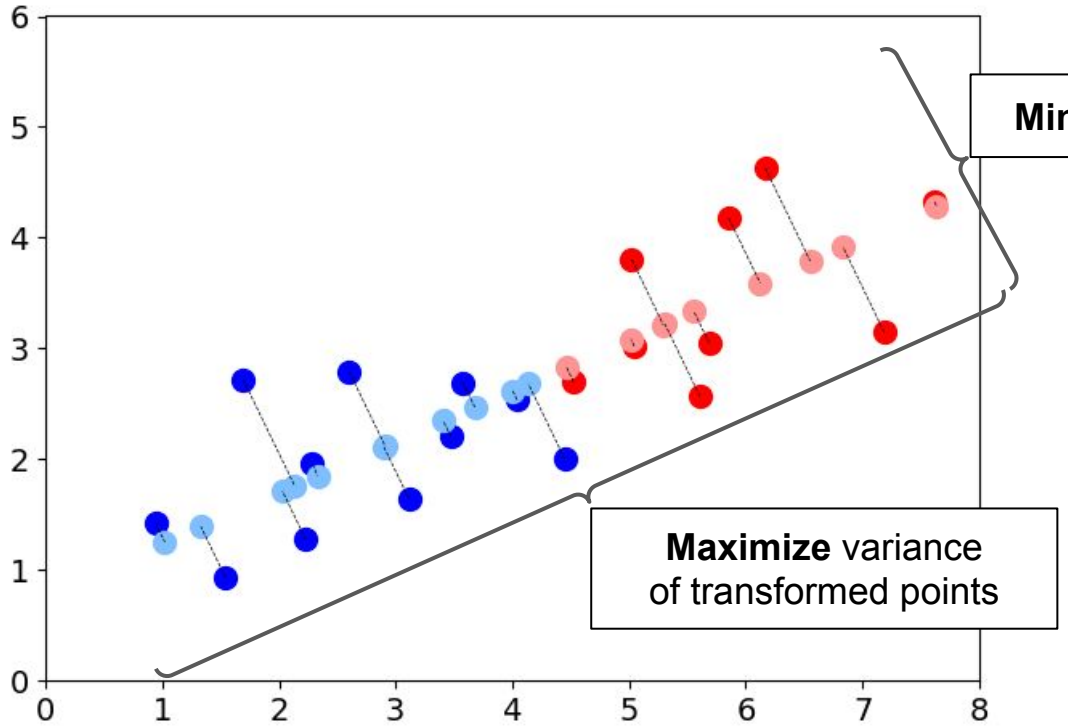
Mapping of data to x-axis: $W = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$



Which transformation is the better one? Why?

Is there an even better transformation?

PCA — Equivalent Objectives for Finding Transformation

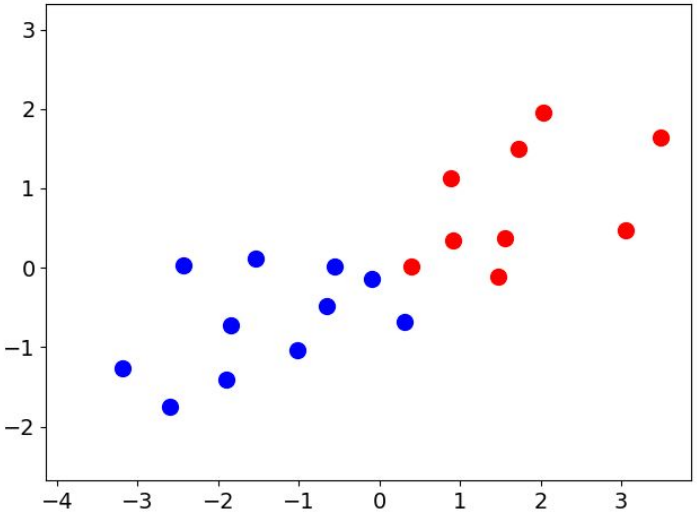


Note: Both optimization objectives are equivalent, with maximizing the variance being easier to handle

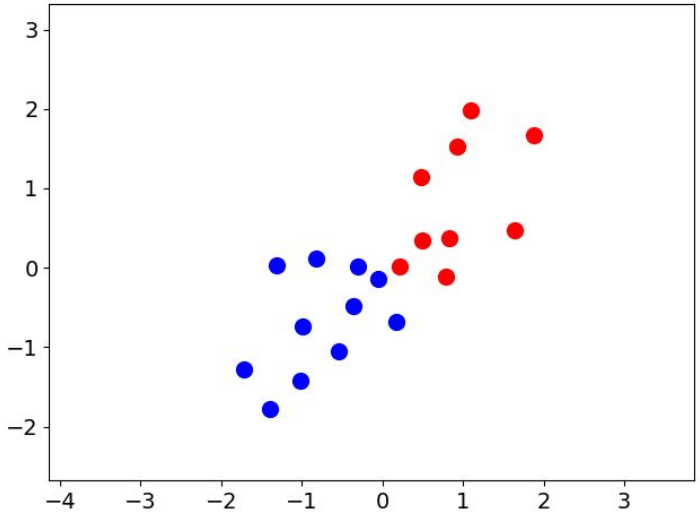
PCA — Data Normalization

- Data normalization steps
  - Mean-centering — does not affect results but makes the math much easier
  - Standardizing (divide by standard deviation) — optional; will affect the results

Mean-centered data

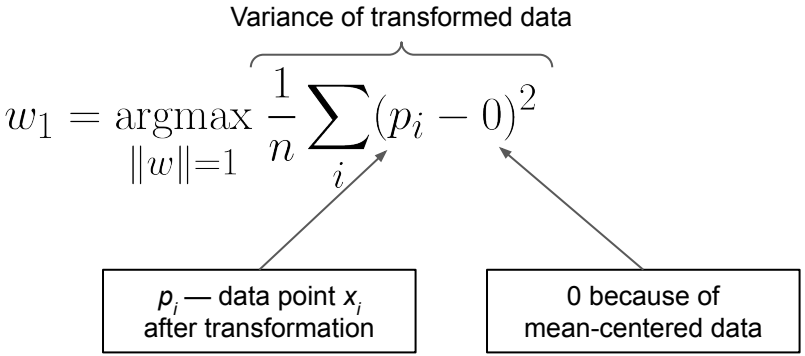


Mean-centered, standardized data



PCA — Finding the 1st Principal Component

- 1st Principal Component of data *X*
  - Unit vector w_1 that maximizes the variance of transformed data



PCA — Finding the 1st Principal Component

- 1st Principal Component of data *X*
  - Unit vector w_1 that maximizes the variance of transformed data

Variance of transformed data

$$w_1 = \underset{\|w\|=1}{\operatorname{argmax}} \frac{1}{n} \sum_i (p_i - 0)^2$$

p_i — data point x_i after transformation

0 because of mean-centered data

$$w_1 = \underset{\|w\|=1}{\operatorname{argmax}} \frac{1}{n} \sum_i (x_i \cdot w)^2$$

$$= \underset{\|w\|=1}{\operatorname{argmax}} \frac{1}{n} \|Xw\|^2$$

$$= \underset{\|w\|=1}{\operatorname{argmax}} \frac{1}{n} w^T X^T X w$$

$$= \underset{\|w\|=1}{\operatorname{argmax}} w^T \frac{X^T X}{n} w$$

$$= \underset{\|w\|=1}{\operatorname{argmax}} w^T C_X w$$

➜ C_X is the covariance matrix of X

Note: Sometimes $C_X = X^T X$ (instead of $C_X = X^T X/n$). In this case C_X is the unnormalized covariance matrix (scatter matrix). This only affects the magnitude of the eigenvalues but the eigenvectors for W.

PCA — Finding the 1st Principal Component

$$w_1 = \underset{\|w\|=1}{\operatorname{argmax}}\, w^T C_X w$$

$$= \underset{\|w\|=1}{\operatorname{argmax}} \frac{w^T C_X w}{w^T w}$$

Note that $w^T w = \|w\| = 1$

Rayleigh Quotient

➜ w_1 is the largest eigenvector of the covariance matrix C_X

PCA — Finding the k-th Principal Component

- Substract (k-1) principal components from X

$$X_k = X - \sum_{s=1}^{k-1} X w_s w_s^T$$

- k-th Principal Component of data X_k
  - Unit vector w_k that maximizes the variance of transformed data — after transforming X_k

$$w_k = \underset{\|w\|=1}{\operatorname{argmax}}\, w^T \frac{X_k^T X_k}{n} w = \underset{\|w\|=1}{\operatorname{argmax}}\, w^T C_X^{(k)} w = \underset{\|w\|=1}{\operatorname{argmax}} \frac{w^T C_X^{(k)} w}{w^T w}$$

➜ w_k is the largest eigenvector of the covariance matrix $C_X^{(k)}$

PCA — The Math

$$J(w) = \frac{w^T C w}{w^T w}$$

$$\frac{dJ(w)}{dw} = \frac{\frac{d(w^T C w)}{dw} \cdot w^T w - w^T C w \cdot \frac{d(w^T w)}{dw}}{(w^T w)^2}$$

Quotient Rule

$$= \frac{w^T (C + C^T)}{w^T w} - \frac{w^T C w \cdot 2w}{(w^T w)^2}$$

$$\frac{x^T A x}{dx} = x^T (A + A^T)$$

$$= \frac{2Cw}{w^T w} - \frac{w^T C w \cdot 2w}{(w^T w)^2}$$

C is symmetric ➜ $C + C^T = 2C$

$$= \frac{2}{w^T w} \left(Cw - \frac{w^T C w}{w^T w} w \right)$$

scalar value!

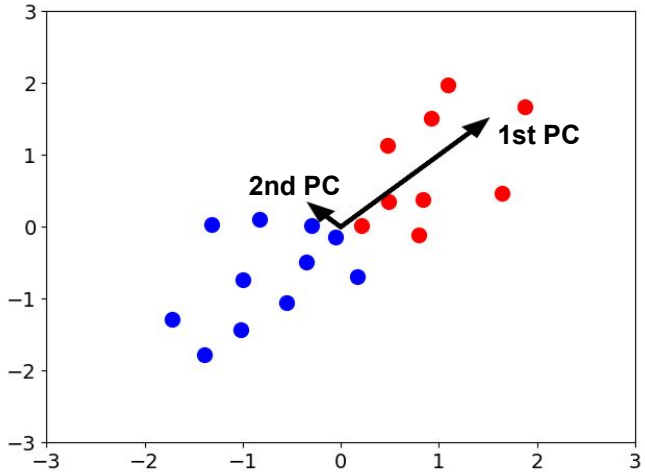
$$\frac{dJ(w)}{dw} = \frac{2}{w^T w}(Cw - J(w)w) \stackrel{!}{=} 0 \quad \Leftrightarrow \quad Cw = \overbrace{J(w)}w$$

➜ Eigenvalue problem!

PCA — Getting all Principal Components

- Mathematical convenience
  - Largest eigenvector of $C_X^{(k)}$ = k-largest eigenvector of C_X

➜ Principal Components of X = eigenvectors of covariance matrix C_X



Interpretation

- 1st PC points into the direction of maximum variance
- 2nd PC points into the direction of maximum variance after 1st PC removed from dataset X
- ...

PCA in Python Code (using numpy library)

```python
def normalize(X):
    X -= np.mean(X, 0) # Mean-center the data
    X /= np.std(X, 0)  # Standardize data (optional; in line with sklearn.decomposition.PCA)
    return X


def covariance(X):
    return np.dot(X.T, X) / X.shape[0] # Assumes mean-centered data matrix X


def pca(X, num_pc=None):
    # Prepare data
    X = normalize(X)
    C = covariance(X)
    # Calculate all eigenvectors and eigenvalues
    eigenval, eigenvec = np.linalg.eigh(C)
    # find the the num_pc largest eigenvalues
    top_idx = np.argsort(eigenval)[::-1][:num_pc]
    # Create transformation matrix W using eigenvectors with the largest eigenvalues
    W = eigenvec[:, top_idx]
    # Return the transformed data
    return np.dot(X, W)
```

PCA — Transforming Original Dataset X

- C_x is a ($d \times d$) matrix ➜ d eigenvectors and eigenvalues
  - How to choose $1 \leq p \leq d$ to get transformation matrix W of shape ($d \times p$)?

- **Explained Variance Ratio**
  - Percentage of variance that is attributed by each principal component

    ➜ normalized eigenvalues
  - Choose p such that p largest PCs explain a minimum amount of variance

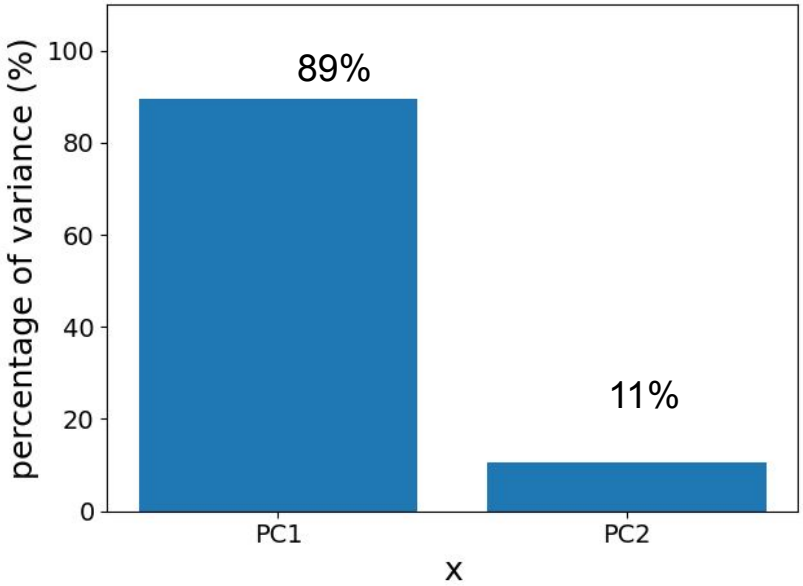
$$W = \begin{bmatrix} | \\ w_1 \\ | \end{bmatrix}$$

Using only PC1

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

Using PC1 and PC2

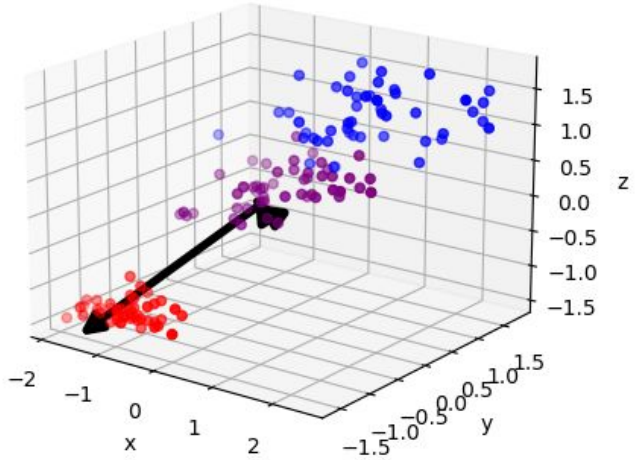
Eigenvalues for previous example:
[0.22, 1.88] ➜ [0.11, 0.89]



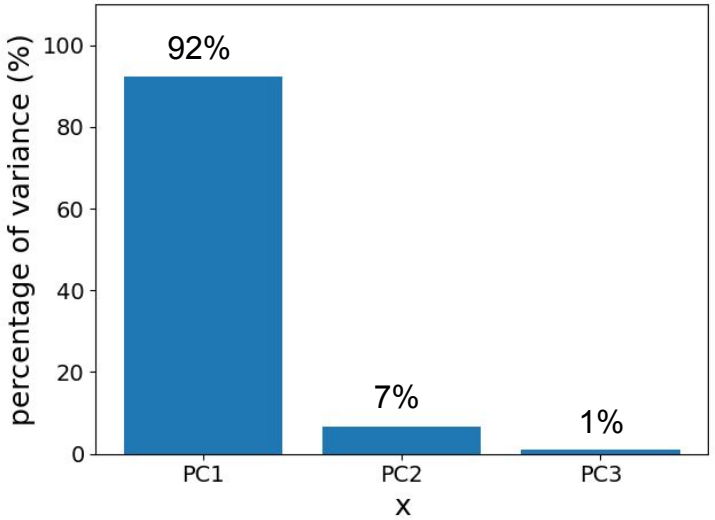
PCA — Full Example (IRIS dataset)

- IRIS dataset
  - Only 3 (out of 4) features considered — only to allow for easy visualization

Dataset with the 3 principal components



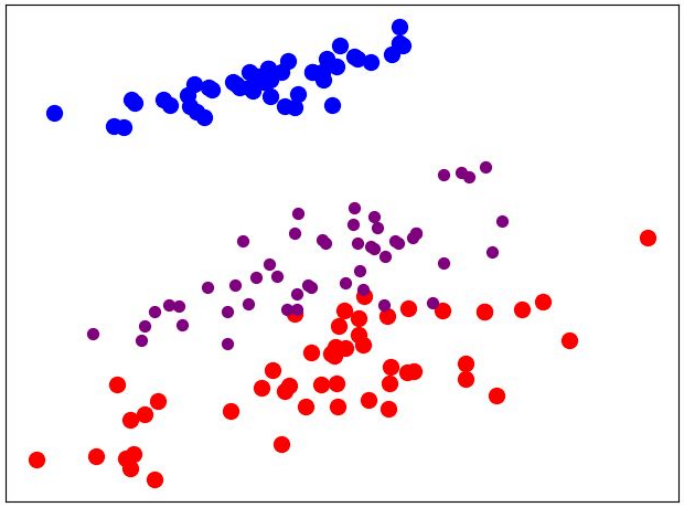
Explained variance of the 3 PCs



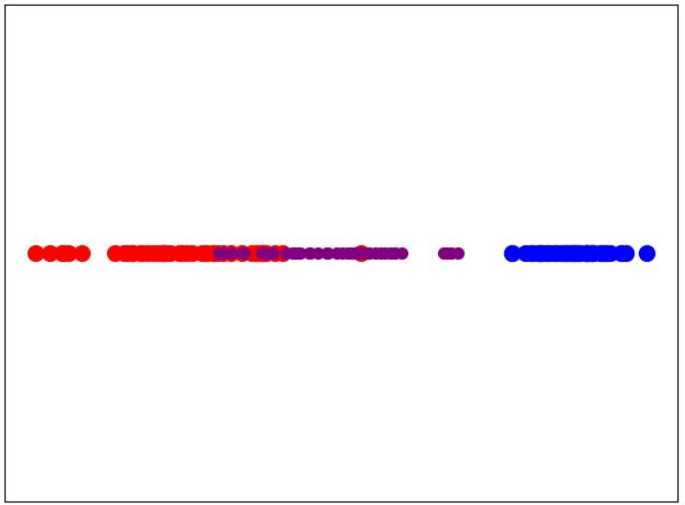
PCA — Full Example (IRIS dataset)

- Transformation of *X* using principal components

Using PC1 and PC2
(99% of variance explained)



Using only PC1
(92% of variance explained)



PCA — Pros & Cons

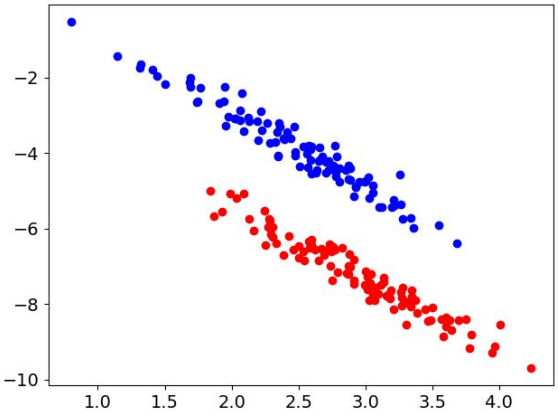
- Pros
  - Intuitive — exploit knowledge about correlated and low-variance features
  - Can significantly reduce the amount of data
  - Improves performance of algorithms and reduces risk of overfitting
  - Visualization of high-dimensional data (even just when applied during EDA)

- Cons
  - Most fundamentally: loss of information
  - Assumes linear correlations among features
  - Assumes that large variance equals high importance (does not always have to be the case)
  - Does not take class labels into account (in case of classification tasks)

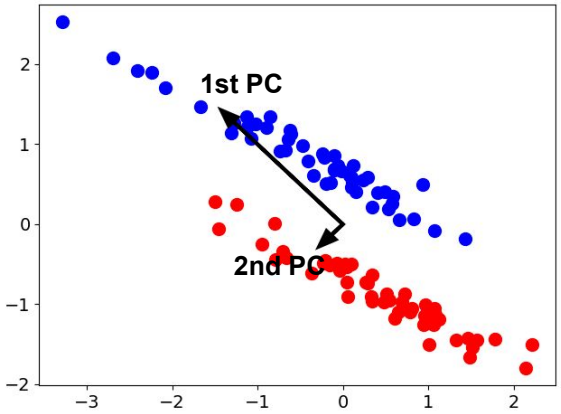
PCA — Limitation for Classification Datasets

- PCA applied to labelled datasets for classification (pathological example)
  - PCA maximizes w.r.t. the variance of the whole dataset
  - PCA ignores any information from the class labels

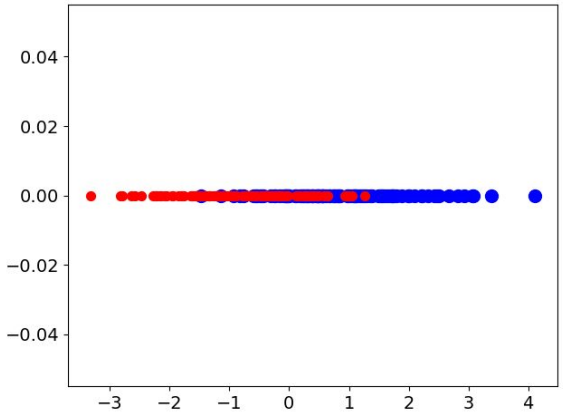
2-class dataset



principal components



transformation using 1st PC



Outline

- Dimensionality Reduction
  - Motivation
  - Naive approaches

- **Dimensionality Reduction Techniques**
  - PCA — Principal Component Analysis
  - **LDA — Linear Discriminant Analysis**
  - t-SNE — t-distributed Stochastic Neighbor Embedding

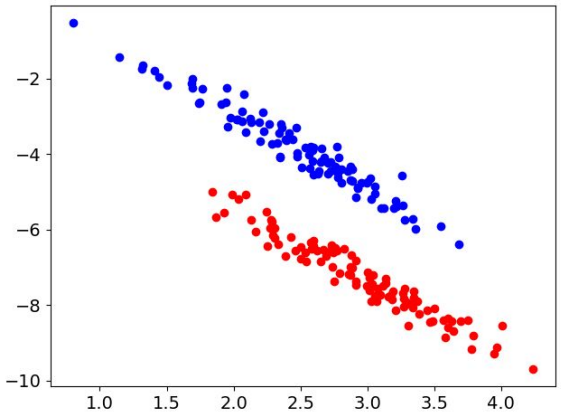
Linear Discriminant Analysis (LDA)

- Similarity to PCA
  - Linear transformation technique
  - Output: Matrix W to transform dataset X into a lower-dimensional space

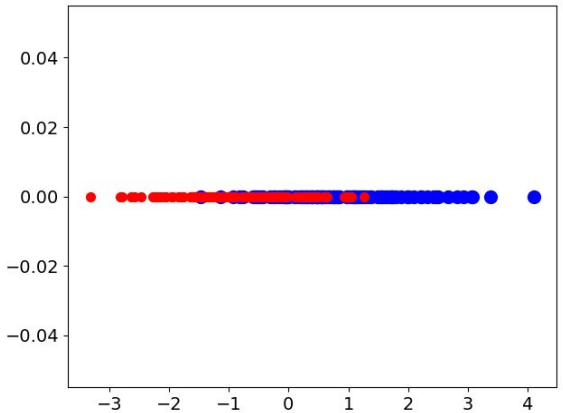
- Main difference: 2 optimization objectives
  - Minimize variance of transformed points with each class
    (recall that PCA maximizes the variance across the whole dataset)
  - Maximize separation between classes

LDA vs. PCA — Example

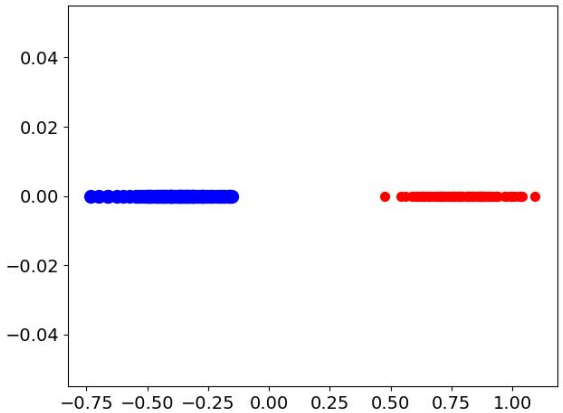
2-class dataset



PCA

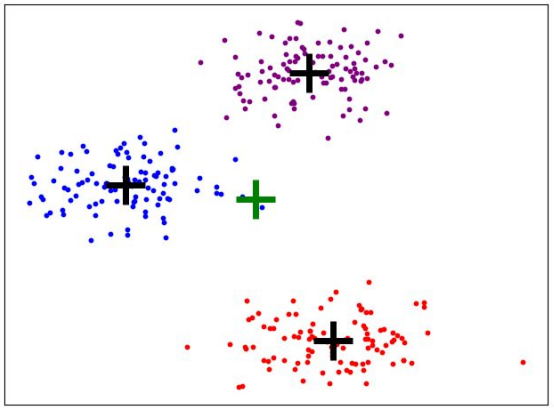


LDA



LDA Concepts — Between-Class Variance

- Variance of distances between class means and (overall) mean
  - C — number of classes
  - $\mu^{(i)}$ — class mean vector
    (mean of data points of class i)
  - μ — (overall) mean vector
    (mean of all data points)



$$\left[m^{(i)} - m\right]^2 = \left[w^T\mu^{(i)} - w^T\mu\right]^2$$

projected mean vectors

$$= w^T \underbrace{\left[\mu^{(i)} - \mu\right]\left[\mu^{(i)} - \mu\right]^T}_{\text{scatter of class means}} w$$

$$= w^T S_B^{(i)} w$$

over all classes

$$= w^T S_B w, \text{ with } S_B = \sum_{i=1}^{C} n^{(i)} S_B^{(i)}$$

LDA (S_B) in Python Code (using numpy library)

```python
def calc_S_b(X, y):
    C = np.unique(y)  # C = set of class labels
    d = X.shape[1]    # d = number of features

    mu = np.mean(X, axis=0)  # Calculate overall mean

    S_b = np.zeros((d, d))   # Initialize S_b matrix

    for c in C:
        # n_i number or samples labeled c
        n_i = X[y==c,:].shape[0]
        # mu_i mean of samples labeled c
        mu_i = np.mean(X[y==c], axis=0)
        # Calculate difference vector
        # (= mean-centering class means)
        diff = (mu_i - mu).reshape(-1, 1)
        # Add scatter of difference vector
        S_b += n_i * np.dot(diff, diff.T)

    return S_b
```

$$= w^T \underbrace{\left[\mu^{(i)} - \mu\right]\left[\mu^{(i)} - \mu\right]^T}_{\text{scatter of class means}} w$$

$$= w^T S_B^{(i)} w$$

over all classes

$$= w^T S_B w, \text{ with } S_B = \sum_{i=1}^{C} n^{(i)} S_B^{(i)}$$

LDA Concepts — Within-Class Variance

- Variance of data points of the same class

$$\sum_{j=1}^{n^{(i)}} \left[p_j^{(i)} - m^{(i)} \right]^2 = \sum_{j=1}^{n^{(i)}} \left[w^T x_j^{(i)} - w^T \mu^{(i)} \right]^2$$

$$= \sum_{j=1}^{n^{(i)}} w^T \underbrace{\left[x_j^{(i)} - \mu^{(i)} \right] \left[x_j^{(i)} - \mu^{(i)} \right]^T}_{\text{scatter of data points of class } i} w$$

$$= w^T S_W^{(i)} w$$

over all classes

$$= w^T S_W w, \text{ with } S_W = \sum_{i=1}^{C} S_W^{(i)}$$

LDA (S_W) in Python Code (using numpy library)

```python
def calc_S_w(X, y):
    C = np.unique(y)  # C = set of class labels
    d = X.shape[1]    # d = number of features

    S_w = np.zeros((d, d))  # Initialize S_w matrix

    for c in C:
        # Get all samples labeled c
        X_c = X[y==c]
        # Mean-center the data
        X_c -= np.mean(X_c, 0)
        # Add scatter matrix of X_c
        S_w += np.dot(X_c.T, X_c)

    return S_w
```

$$= \sum_{j=1}^{n^{(i)}} w^T \left[x_j^{(i)} - \mu^{(i)}\right] \left[x_j^{(i)} - \mu^{(i)}\right]^T w$$

$$= w^T S_W^{(i)} w$$

⇩ over all classes

$$= w^T S_W w, \text{ with } S_W = \sum_{i=1}^{C} S_W^{(i)}$$

LDA — Optimization Objective

Maximize: $J(w) = \dfrac{w^T S_B w}{w^T S_W w}$

- $w^T S_B w$: scatter of projected class means
- $w^T S_W w$: scatter of projected data points (per class)

Generalized Rayleigh Quotient

➜ Generalized eigenvalue problem: $S_W^{-1} S_B w = \underbrace{J(w)}_{\text{scalar value!}} w$

Optimal projection vectors =

eigenvectors of largest the eigenvalues of matrix $S_W^{-1} S_B$

LDA — The Math

$$J(w) = \frac{w^T A w}{w^T B w}$$

$$\frac{dJ(w)}{dw} = \frac{\frac{d(w^T A w)}{dw} \cdot w^T B w - w^T A w \cdot \frac{d(w^T B w)}{dw}}{(w^T B w)^2}$$

Quotient Rule

$$= \frac{w^T (A + A^T)(w^T B w) - w^T (B + B^T)(w^T A w)}{(w^T B w)^2}$$

$$\frac{x^T A x}{dx} = x^T (A + A^T)$$

$$= \frac{2Aw(w^T B w) - 2Bw(w^T A w)}{(w^T B w)^2}$$

X is symmetric ➜ $X + X^T = 2X$

scalar value!

$$\frac{dJ(w)}{dw} = \frac{2}{w^T B w}(Aw - J(w)Bw) \stackrel{!}{=} 0 \quad \Leftrightarrow \quad Aw = \overbrace{J(w)}Bw$$

$$B^{-1}Aw = J(w)w$$

➜ Generalized Eigenvalue problem!

Note on the Number of Eigenvectors

- Definition of S_B includes two constraints
  - S_B is the sum of C matrices of rank 1 or less
  - The mean μ is constraint by $\mu = \frac{1}{C}\sum_{i=1}^{C}\mu_i$

➜ S_B has rank of (C-1) or less

➜ only (C-1) eigenvectors are non-zero! (the respective eigenvalues are 0)

Note: In practice, these remaining d-c+1 eigenvalues are only very close to zero due to floating pointing imprecisions

LDA — Algorithm

1. Calculate mean vectors $\boldsymbol{\mu}$ and $\boldsymbol{\mu}^{(i)}$ for all C classes
2. Calculate scatter matrices $\boldsymbol{S}_W$ and $\boldsymbol{S}_B$
3. Calculate eigenvectors and eigenvalues of $\boldsymbol{S}_W^{-1}\boldsymbol{S}_B$
4. Select p eigenvectors $\boldsymbol{w}_p$ with the largest eigenvectors

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

with $p \leq C\text{-}1$

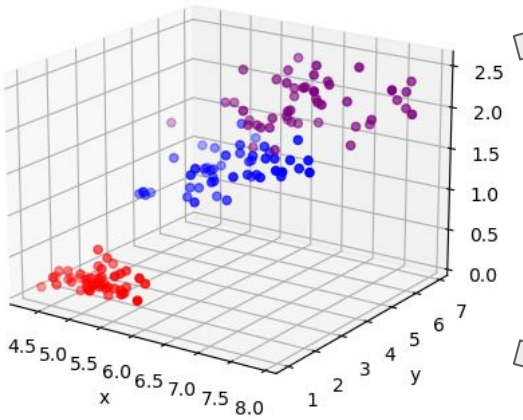
5. Project dataset X into new space via $\boldsymbol{XW}$

LDA in Python Code (using numpy library)

$$S_W^{-1} S_B$$

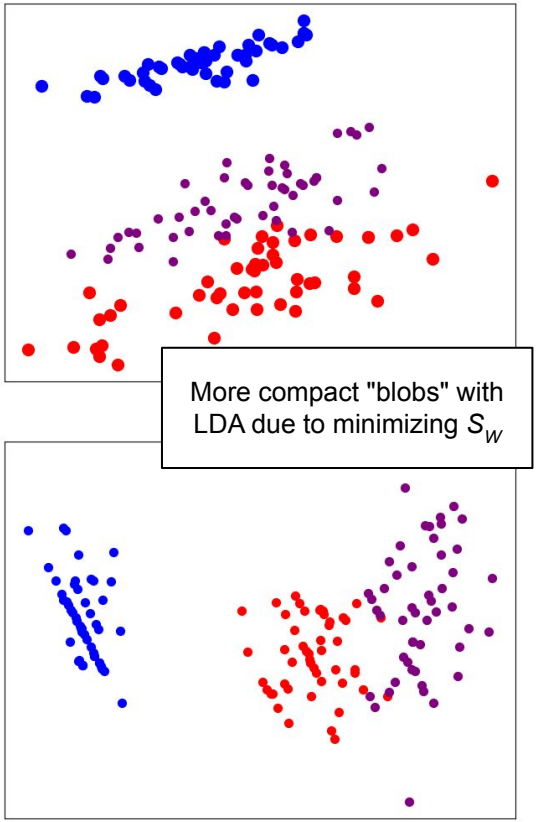
```python
def lda(X, y, num_pc=None):
    # Calculate scatter matrices
    S_w = calc_S_w(X, y)
    S_b = calc_S_b(X, y)
    # Calculate all eigenvectors and eigenvalues
    eigenval, eigenvec = np.linalg.eig(np.linalg.inv(S_w).dot(S_b))
    # find the the num_pc largest eigenvalues
    top_idx = np.argsort(eigenval)[::-1][:num_pc]
    # Create transformation matrix W using eigenvectors with the largest eigenvalues
    W = eigenvec[:, top_idx]
    # Return the transformed data
    return np.dot(X, W)
```

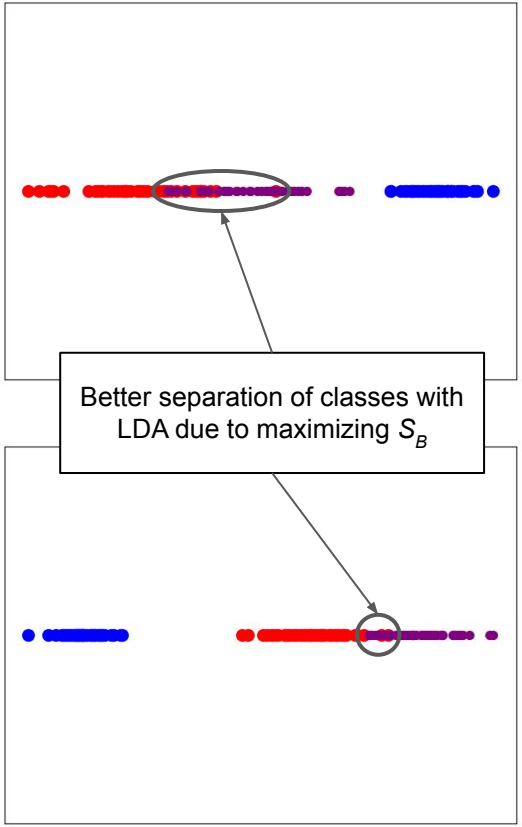
LDA — Full Example (IRIS dataset)



PCA

LDA





LDA — Full Example (IRIS dataset)

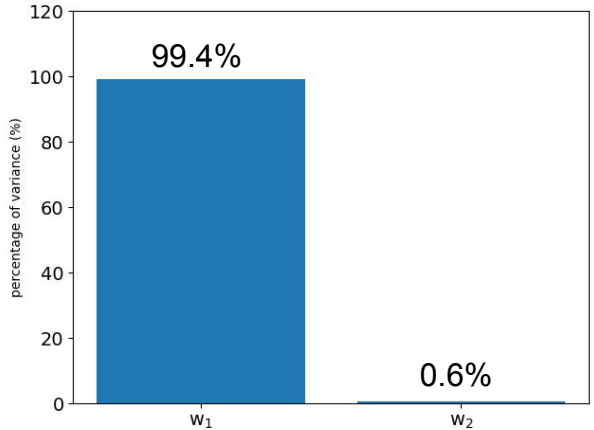
- Resulting *d* eigenvalues (sorted)

1.	26.9
2.	0.17
3.	$2.03e^{-15}$

C-1 = 3-1 = 2 non-zero eigenvalues

d-C+1 = 3-3+1 = 1 zero eigenvalues

- Choosing p ≤ C-1 using Explained Variance Ratio



➜ *p*=1 a good choice

LDA — Pros & Cons

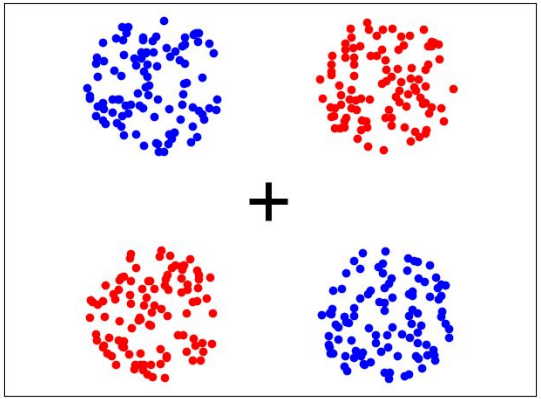
- Pros
  - Intuitive extension to PCA yielding similar benefits
    (reduction in amount of data, reduced risk of overfitting, visualization, etc.)
  - Consideration of class labels (generally more suitable than PCA for labelled dataset)

- Cons
  - Similar cons as PCA (loss of information, assumes linear correlations, etc.)
  - Assumes unimodal Gaussian distributions
  - Assumes that means are the most discriminant features

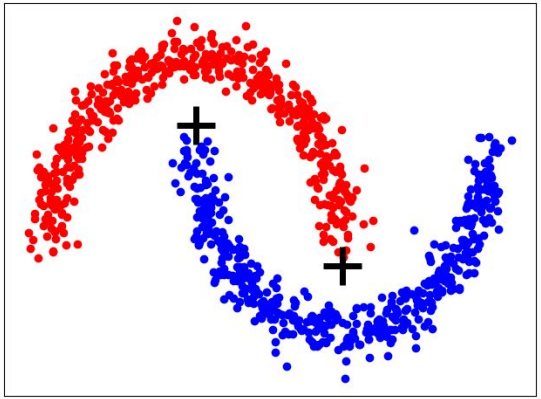
LDA — Problematic Cases

- Data distributions that are (significantly) non-unimodal Gaussian

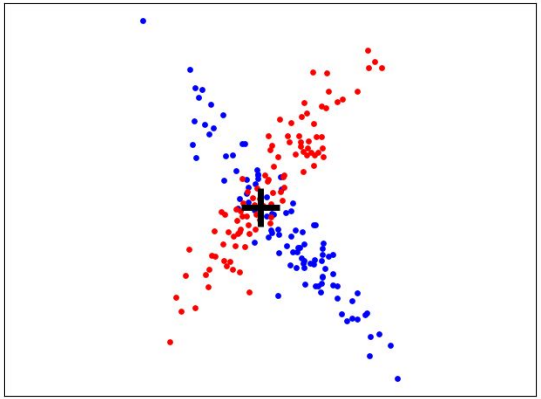
bimodal with $\mu = \mu_1 = \mu_2$



non-Gaussian



unimodal but $\mu = \mu_1 = \mu_2$



Outline

- Dimensionality Reduction
  - Motivation
  - Naive approaches

- **Dimensionality Reduction Techniques**
  - PCA — Principal Component Analysis
  - LDA — Linear Discriminant Analysis
  - **t-SNE — t-distributed Stochastic Neighbor Embedding**

t-Distributed Stochastic Neighbor Embedding (t-SNE)

- t-SNE — Non-linear dimensionality reduction technique
  - Unsupervised approach (independent from any kind of class labels)
  - Iterative algorithm:
    1) Start with random lower-dimensional representation Y
    2) Change Y until a loss function converges to a minimum

- Intuition behind t-SNE
  - Convert Euclidean distances in *X* and *Y* to conditional probabilities
    (e.g., if data points x_i and x_j are close ➜ conditional probability $p_{i|j}$ should be high)
  - Iteratively change Y such that both probability distributions become more similar

➜ Optimization objective: Points that are close in *X* are also close in *Y*

t-SNE — Convert Euclidean Distances to Cond. Probs

- Mathematical formulation
  - For data points in X

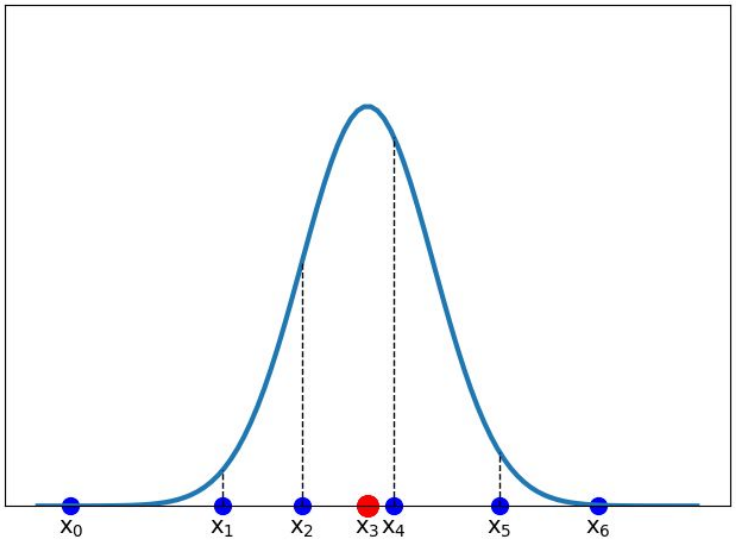
$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)}$$

- Visual explanation (for $i = 3$)
  - Assume d-dim Gaussian centered at x_3
  - Calculate $p_{j|3}$ **proportional** to Gaussian
    (proportional to heights of dashed lines from each point x_j)

$p_{4\|3}$	$p_{2\|3}$	$p_{5\|2}$	$p_{1\|3}$	$p_{6\|3}$	$p_{0\|3}$
0.84	0.16	~0.0	~0.0	~0.0	~0.0

Interpretation

$p_{j|i}$ is the probability that x_i would pick x_j as neighbor



t-SNE

- Calculate joint probabilities $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$

Joint probabilities $P = \{p_{ij}\}$ for previous example

$$\begin{bmatrix} 0 & 0.071 & 0 & 0 & 0 & 0 & 0 \\ 0.071 & 0 & 0.09 & 0 & 0 & 0 & 0 \\ 0 & 0.09 & 0 & 0.057 & 0.09 & 0 & 0 \\ 0 & 0 & 0.057 & 0 & 0.129 & 0.001 & 0 \\ 0 & 0 & 0.009 & 0.129 & 0 & 0.025 & 0 \\ 0 & 0 & 0 & 0.001 & 0.025 & 0 & 0.117 \\ 0 & 0 & 0 & 0 & 0 & 0.117 & 0 \end{bmatrix}$$

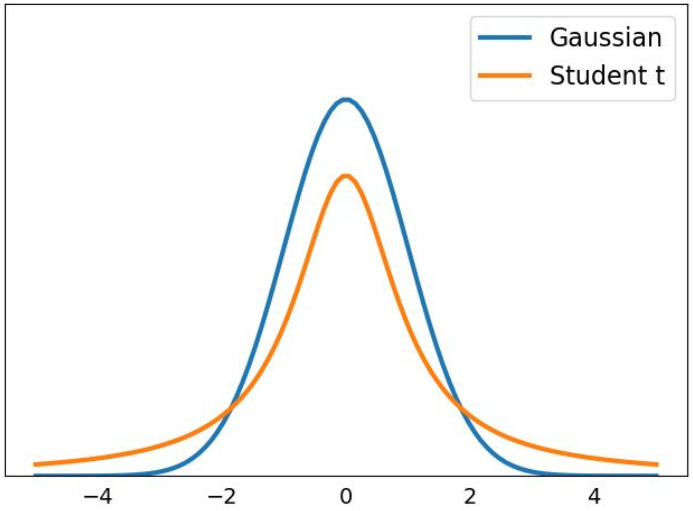
Assume $Q = \{q_{ij}\}$ being the joint probabilities of data points y_i in Y

➜ How to modify all y_i such that $P \approx Q$?

t-SNE

- **t**-SNE uses a Student **t** distribution
  - Heavier tails yield better results in practice
  - 1 degree of freedom

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



- Loss function: Kullback-Leibler divergence between P and Q

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

Note: The KL divergence is a measure of how one probability distribution is different from another probability distribution

t-SNE — Minimizing Loss

- Minimize *L* using Gradient Descent

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

$$\frac{\partial L}{\partial y_i} = \cdots = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1}$$

Details in paper: Visualizing Data using t-SNE (Maaten and Hinton, 2008)

t-SNE — Basic Algorithm

Input : Dataset X, number of iterations T, learning rate η

Initialization : Sample $Y^{(0)} = \{y_1, y_2, \ldots, y_n\}$ from $\mathcal{N}(0, 10^{-4}\mathbf{I})$

Calculate $p_{j|i}$ and p_{ij} for all (x_i, x_j)-pairs

for $t = 1$ **to** T **do**

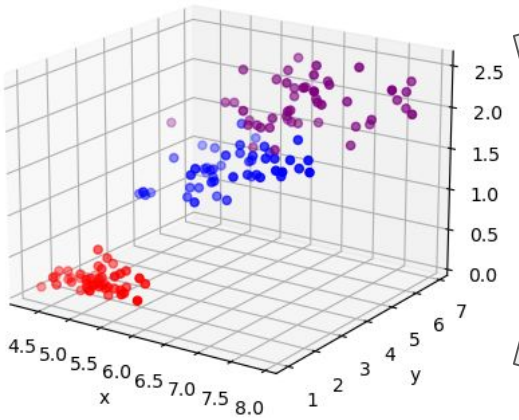
Calculate q_{ij} for all (y_i, y_j)-pairs

Calculate gradients $\dfrac{\partial L}{\partial y_i}$

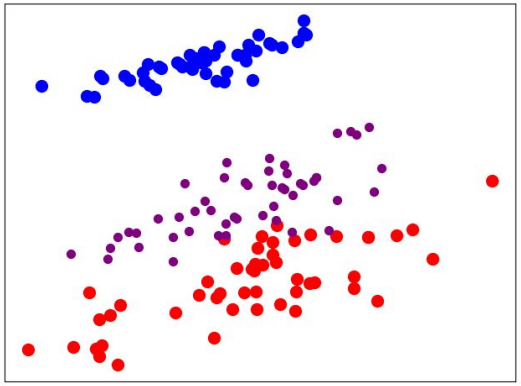
Update $y_i^t \leftarrow y_i^{t-1} - \eta \dfrac{\partial L}{\partial y_i}$

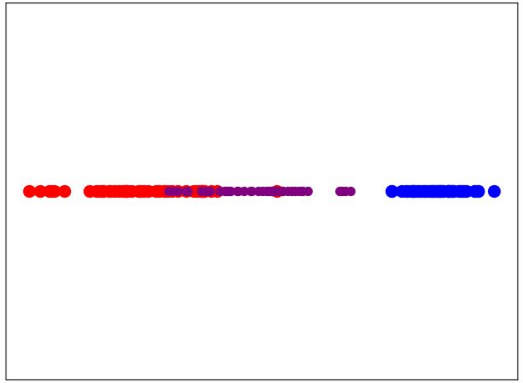
return $Y^{(T)}$

t-SNE — Full Example (IRIS dataset)

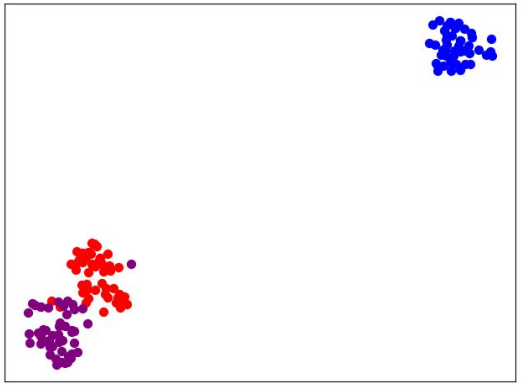


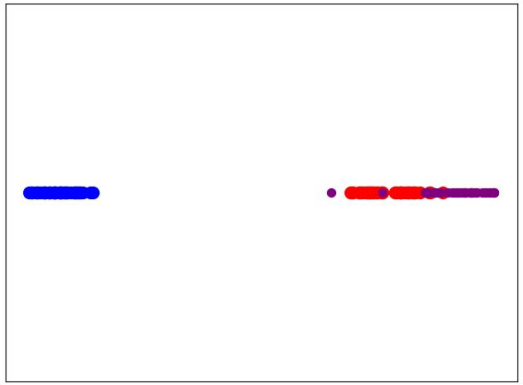
PCA





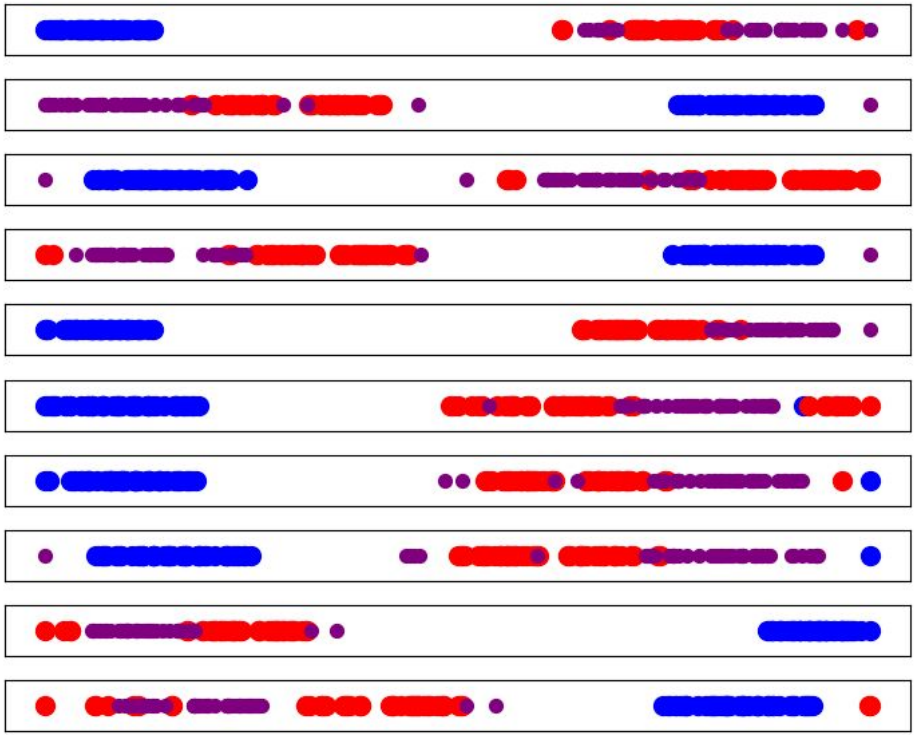
t-SNE





t-SNE — Non-Determinism

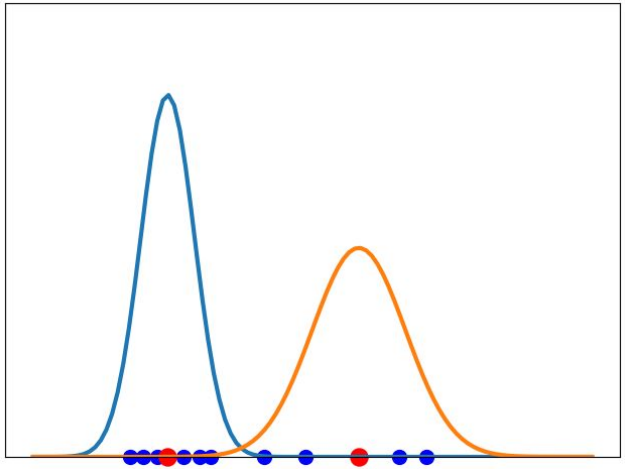
- t-SNE is non-deterministic
  - $Y^{(0)}$ is randomly sampled
  - Different runs will generally yield different projections
  - In practice, perform multiple runs to get an understanding of the data



t-SNE — Calculating $p_{j|i}$ (implementation detail)

- How to pick the values for σ_i?

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

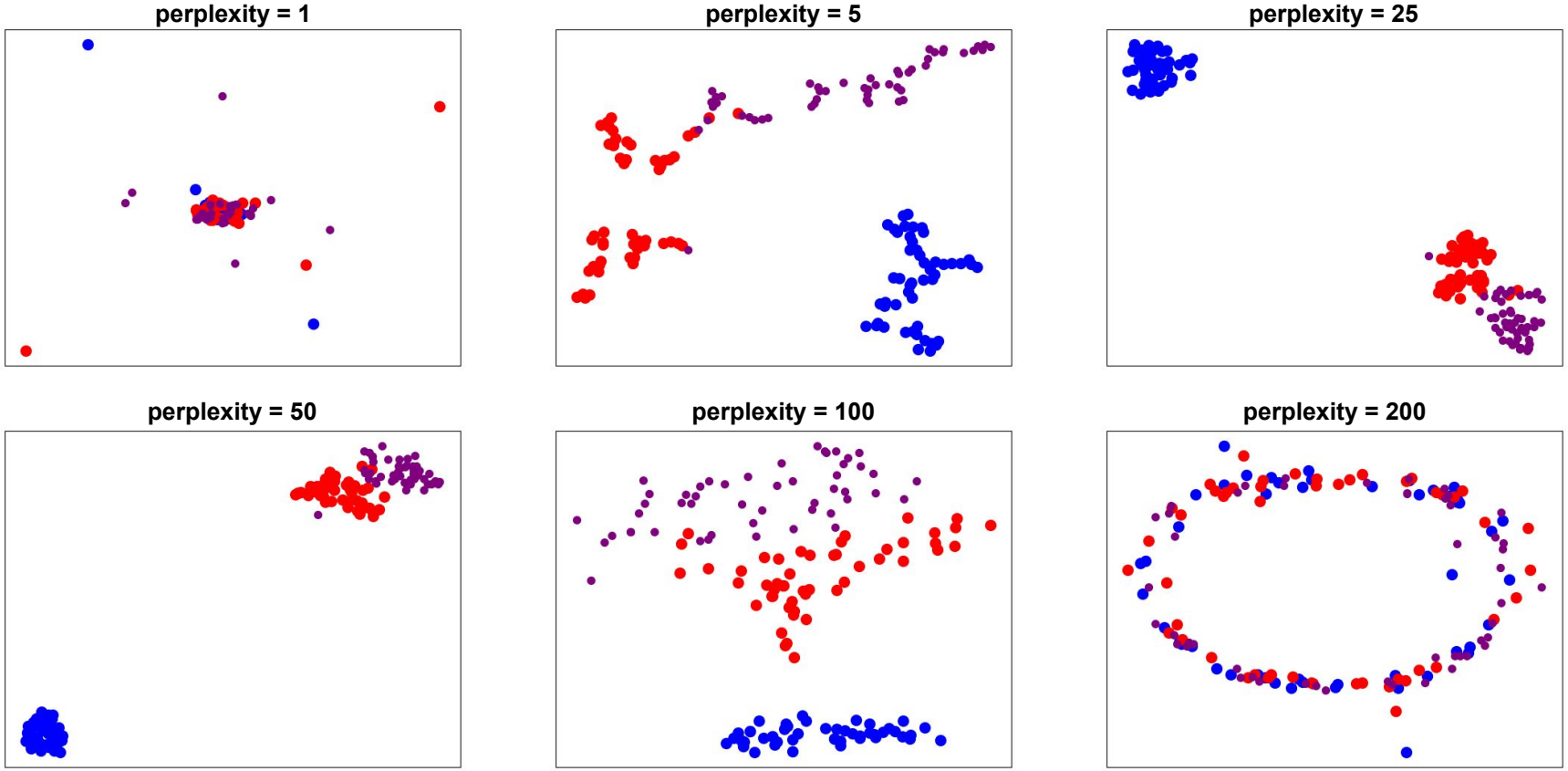


- Intuition: Set σ_i based on density around x_i
  - High density ➜ smaller σ_i / Low density ➜ larger σ_i
  - Controls how many x_j with effective $p_{j|i}$

- Calculate best based σ_i on hyperparameter **perplexity**
  - Larger perplexity: more neighbors have effective $p_{j|i}$
  - Common perplexity values in practice: 5..50

Details in paper: Visualizing Data using t-SNE (Maaten and Hinton, 2008)

t-SNE — Effects of Perplexity Parameter

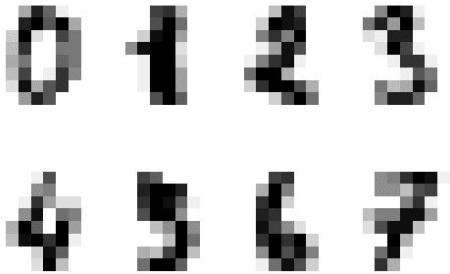


t-SNE — Pros & Cons

- Pros
  - Can handle non-linear data
  - Works very well for data visualization

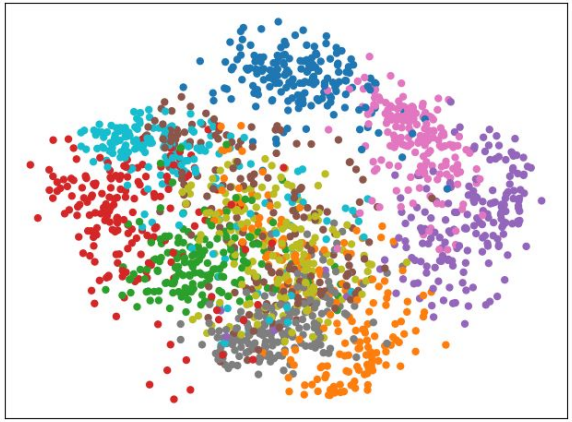
- Cons
  - Computational very expensive on very high-dimensional data (compared to, e.g., PCA)
  - Non-deterministic behavior; might need multiple runs
  - Several hyperparameters affecting the output (perplexity, learning rate, number of iterations, initialization)

Example — Digits Dataset

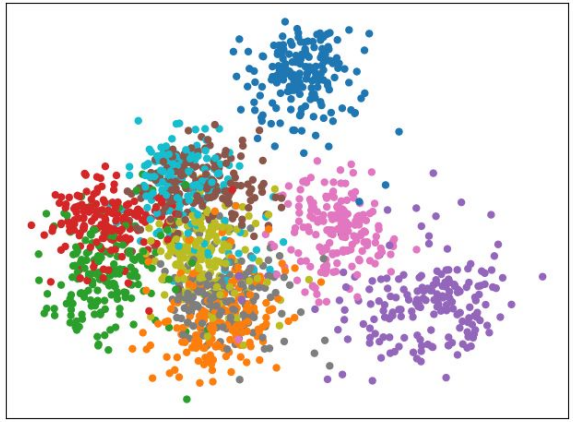


- Digits Dataset
  - 1,797 handwritten digits 0, 1, 2, …, 9
    (~180 samples for class)
  - 8×8 pixels ➜ 64 features
    (integer grayscale value 0..16)

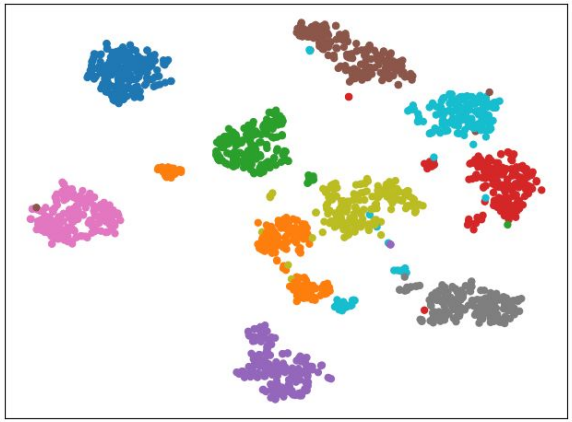
PCA



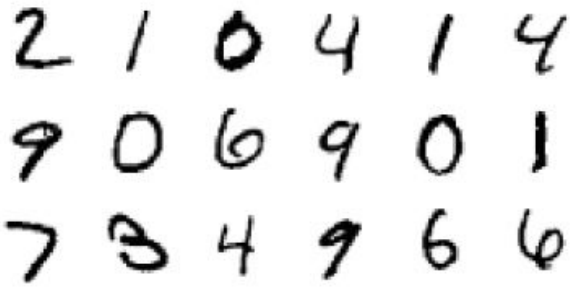
LDA



t-SNE



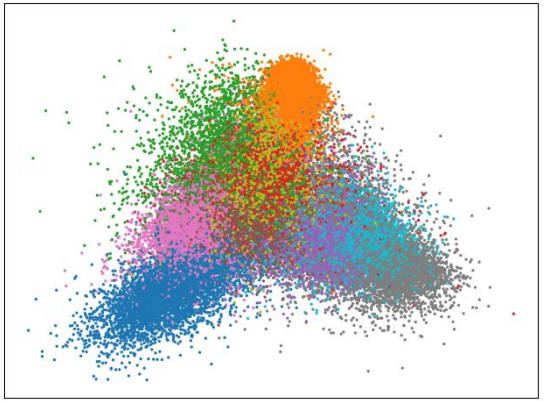
Example — MNIST



- Digits Dataset
  - 60k handwritten digits 0, 1, 2, …, 9
    (~6k samples for class)
  - 28×28 pixels ➜ 784 features
    (integer grayscale valueS 0..255)

PCA (4.9 sec)

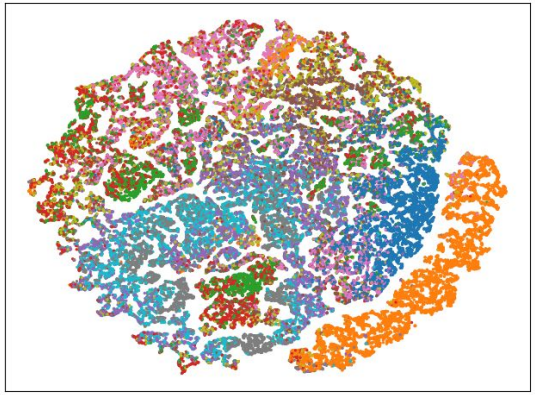
LDA (5.1 sec)



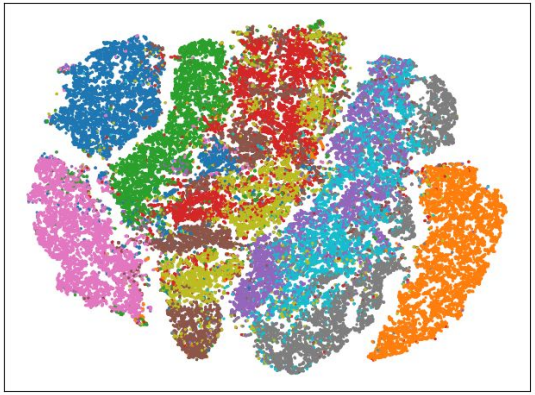
t-SNE (**~58 min!**)

PCA + t-SNE

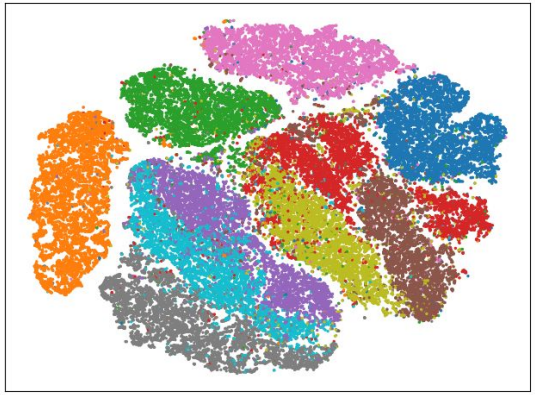
PCA/3 + t-SNE (2:04 min)



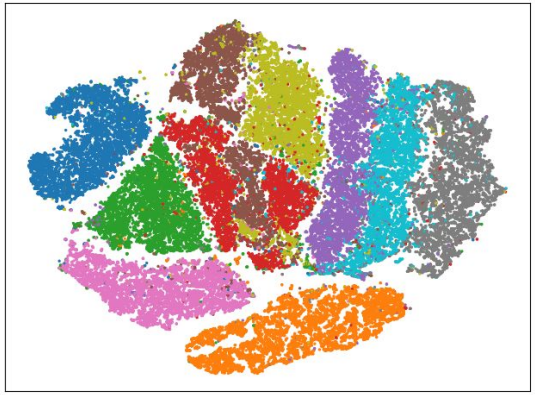
PCA/7 + t-SNE (2:30 min)



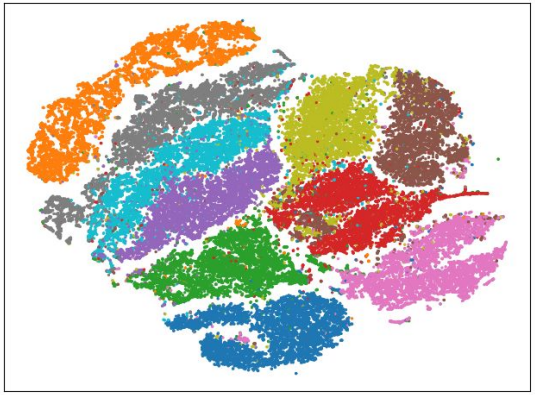
PCA/14 + t-SNE (2:56 min)



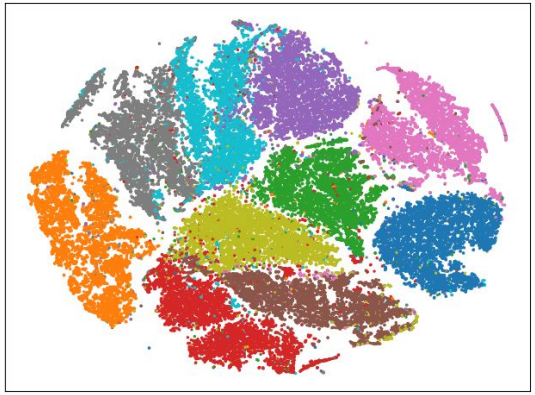
PCA/28 + t-SNE (4:31 min)



PCA/56+ t-SNE (8:16 min)



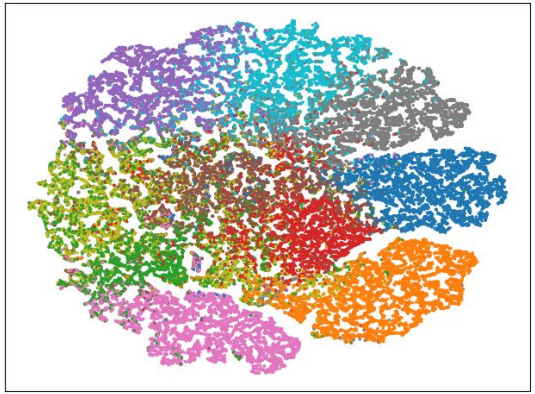
PCA/128 + t-SNE (14:00 min)



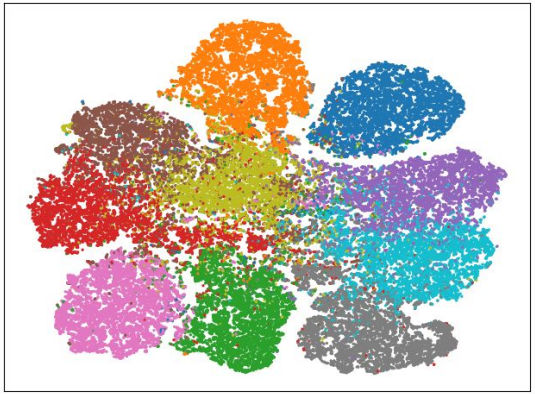
LDA + t-SNE

- Recall restriction of LDA (compared to PCA)
  - Only up to (C-1) non-zero eigenvalues
  - Range of dimensionality reduction by: $1 \leq p \leq 9$ for MNIST

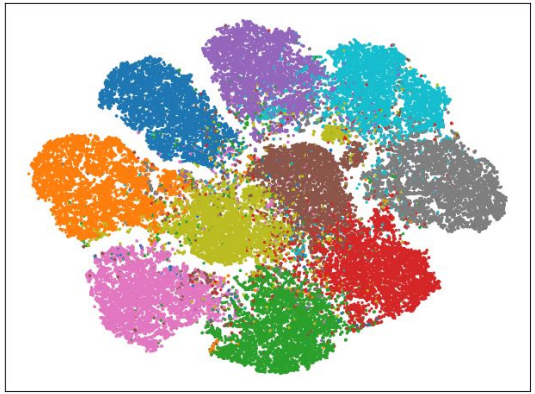
LDA/3 + t-SNE (2:30 min)



LDA/6+ t-SNE (2:41 min)



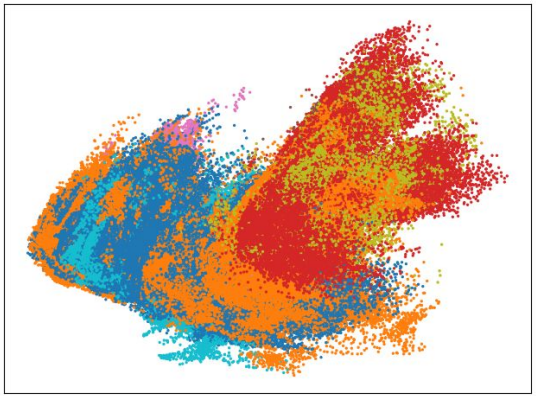
LDA/9 + t-SNE (2:55 min)



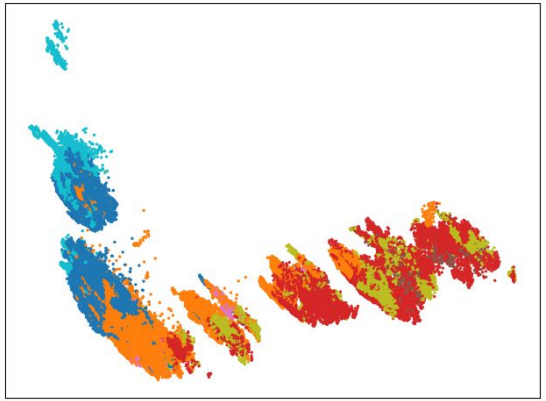
Example — Forest Cover Type

- Forest cover type classification dataset
  - 581k samples of 30×30m cells with 1 of 7 forest types
    (Spruce/Fir, Lodgepole Pine, Ponderosa Pine, Cottonwood/Willow, Aspen, Douglas-fir, Krummholz)
  - 54 features (elevation, aspect, slope, etc.)

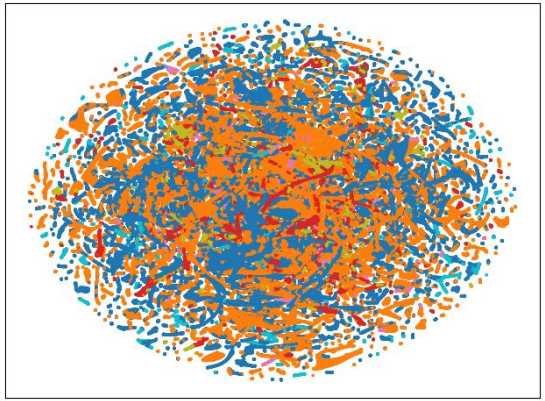
PCA (1.2sec)



LDA (3.2sec)



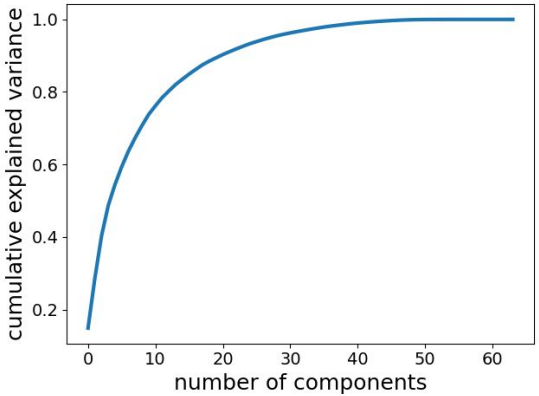
t-SNE (**~30min**)



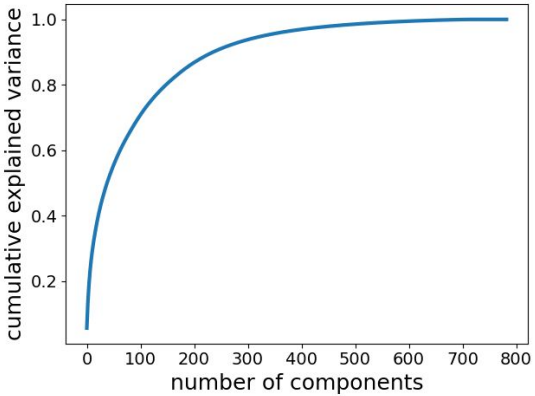
Example — Explained Variance (PCA)

- Cumulative explained variance
  - Data distribution more homogeneous for Digits and MNIST
  - Cover type dataset with many features have only 0 values (low/no variance)

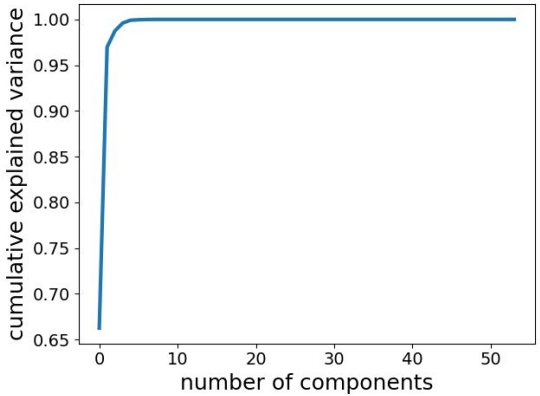
Digits



MNIST



Cover Type



Summary

- Problem: high-dimensional dataset (i.e., large number of features)
  - Higher risk of overfitting
  - Higher computational costs
- Two basic types of countermeasures
  - Feature selection — "manually" remove subset of features
  - Feature extraction — create new features based on original features
- Dimensionality reduction techniques for feature extraction
  - Linear techniques: PCA (unsupervised) and LDA (supervised)
  - Probabilistic techniques: t-SNE