Topics:

① Vector & Matrix derivatives → ML ← Intuition ← post-read

High dim data viz

② PCA - for visualization

③ t-SNE (2008)

④ UMAP (2018)

- Math ✓ ; intuition
- Research papers
- Practical considerations
- Code

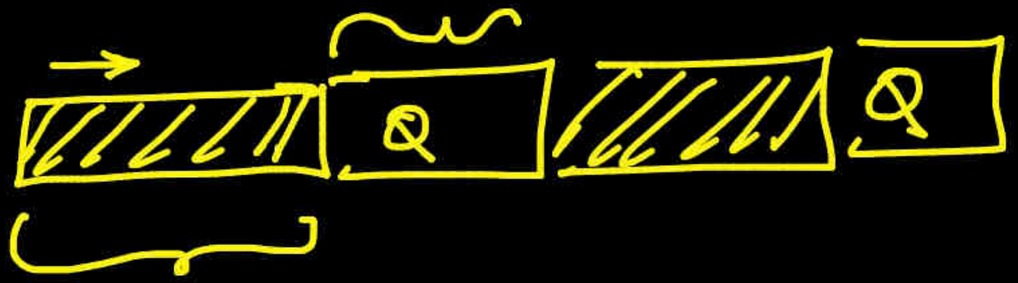
Toy

d-dim, d=100 → 2D or 3D → interactivity ✓

Chrome File Edit View History Bookmarks Profiles Tab Window Help

ops:

Question - Tab



"END:----."

2 / 2

Vector & Matrix derivatives

post-mul (trick)

(1) $y_{m\times 1} = A_{m\times n}\, x_{n\times 1}$

$$\frac{\partial y}{\partial x} = A \quad \Rightarrow \quad \frac{d(Ax)}{dx} = A$$

(2)

$$\underset{1\times 1}{\alpha} = \underbrace{\underset{1\times m}{y^T}\underset{m\times n}{A}}_{1\times n}\underset{n\times 1}{x}$$

post-mul

$(y_{m\times 1})$

scalar

$$\frac{\partial \alpha}{\partial x} = y^T A$$

(3)

pre → $\alpha = y^T A x$

α: 1×1 ; y^T: $1\times M$; A: $M\times N$; x: $N\times1$

$(A\cdot B)^T = B^T A^T$

$\frac{\partial \alpha}{\partial y} = (Ax)^T = x^T A^T$ (earlier)

$\frac{\partial \text{Loss}}{\partial w}$ → scalar

↳ vec $\vec{w}$

$\nabla_w \mathcal{L} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial w_1} \\ \frac{\partial \mathcal{L}}{\partial w_2} \\ \vdots \end{bmatrix}$

Typical proof:

To prove: $\frac{\partial \alpha}{\partial y} = x^T A^T$; $\alpha = \underbrace{y^T A x}_{w^T}$

$(A \cdot B)^T = B^T A^T$

Let $w^T = y^T A$ $\Rightarrow (w^T)^T = (y^T A)^T$

$\Rightarrow w = A^T y$

$\Rightarrow \alpha = w^T \cdot x$

$\alpha^T = (w^T \cdot x)^T$

$\Rightarrow \alpha^T = x^T w$

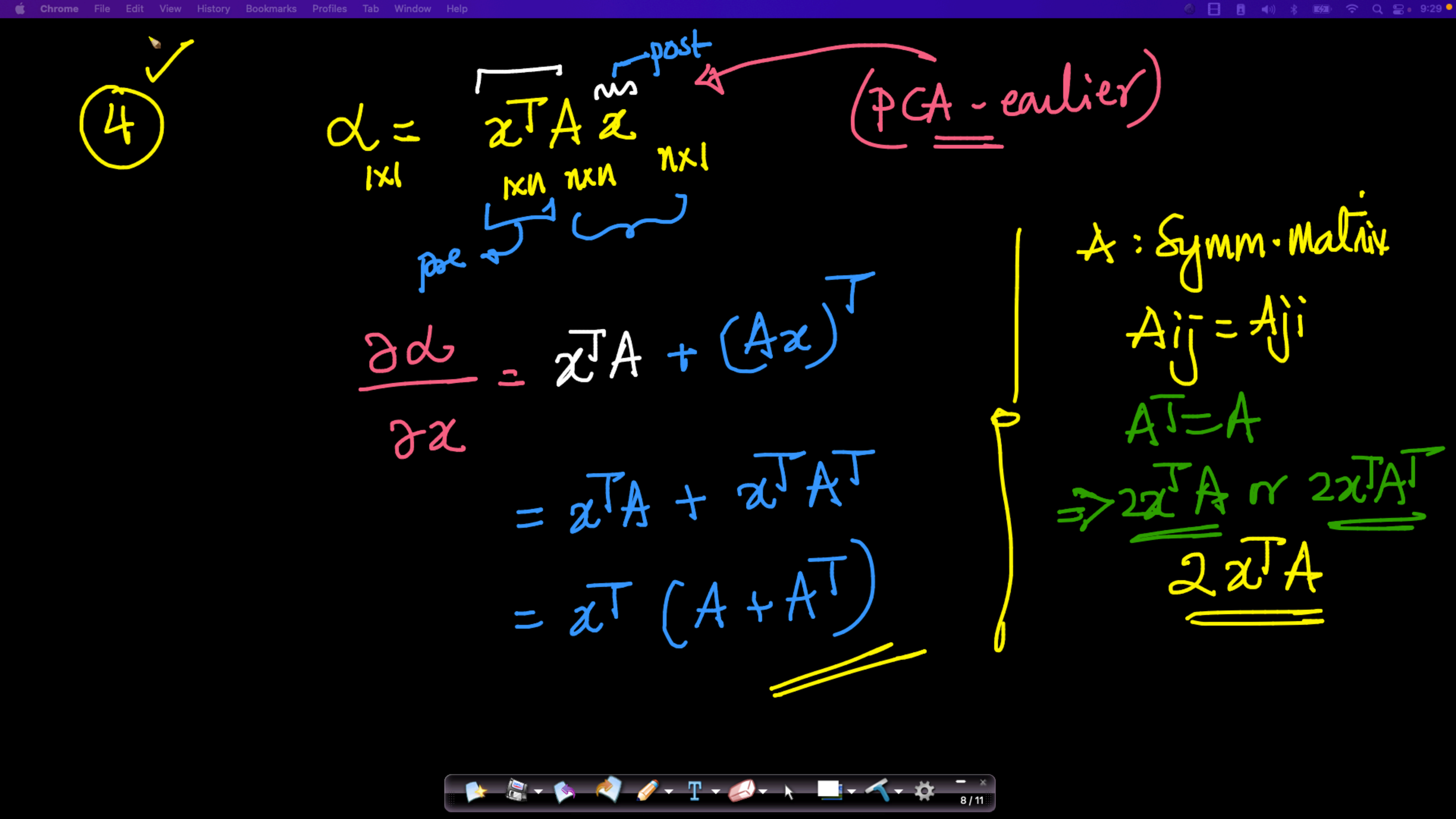
(*) $\alpha_{1\times 1}$: scalar

✓ $\alpha^T = \alpha$

$\alpha = \alpha^T = x^T W$

$\Rightarrow \alpha = x^T A^T y$ ← past.

$\frac{\partial \alpha}{\partial y} = x^T A^T$

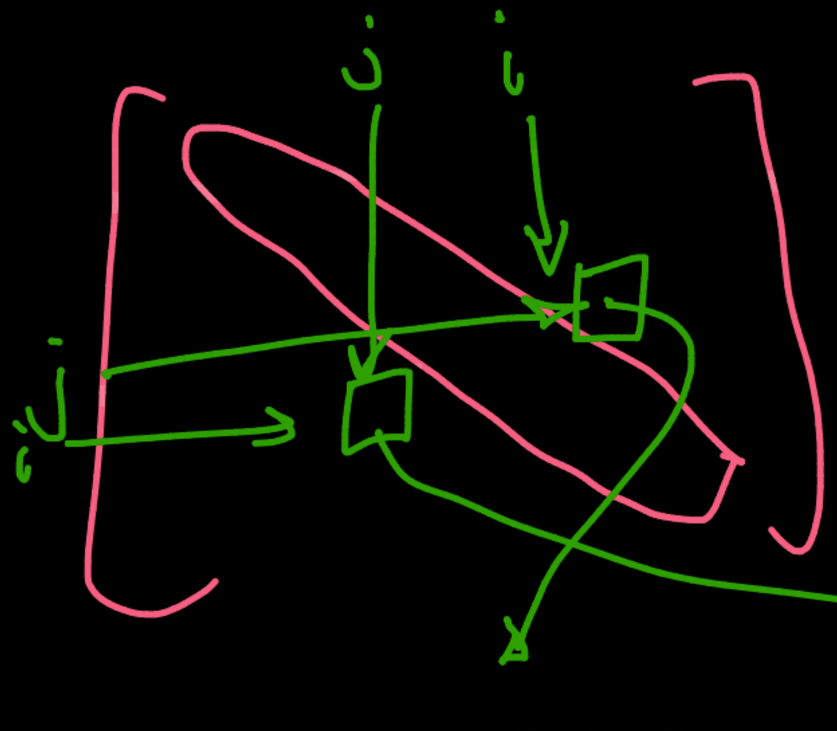

4
$\alpha = x^T A x$
1x1
1xn nxn nxn 1x1
post
pre
(PCA - earlier)
$\frac{\partial \alpha}{\partial x} = x^T A + (Ax)^T$
$= x^T A + x^T A^T$
$= x^T (A + A^T)$
A : Symm. Matrix
$A_{ij} = A_{ji}$
$A^T = A$
$\Rightarrow 2x^T A$ or $2x^T A^T$
$2x^T A$

Cov-matrix

$A = A^T$ } Symm. matrix

$A_{ij} = A_{ji}$

Cov. Matrix



$$\{\ \widetilde{Cov}(f_i, f_j) = Cov(f_j, f_i)$$

$$\underset{1\times 1}{\alpha} = \underset{1\times n}{x^T} \; \underset{n\times 1}{x}$$

num → past

pre

$$\frac{\partial \alpha}{\partial x} = x^T + x^T$$

$$= 2x^T$$

$$\|w\|_2^2 = 1$$

$$\Downarrow$$

$$w^T w = 1$$

$$\min_w \; Loss + \lambda(w^T w)$$

[Most-widely used vector & Matrix derivative]

Q

$$\underset{1\times 1}{\alpha} = \overbrace{A x}^{w}\ \overbrace{y^T}^{m}{}_{m\times 1}$$

doesn't occur much in ML

$$\frac{\partial \alpha}{\partial y^T} = Ax$$

$$\frac{\partial \alpha}{\partial y} = x^T A^T$$

$$\nabla_y \alpha = \begin{bmatrix} \frac{\partial \alpha}{\partial y_1} \\ \frac{\partial \alpha}{\partial y_2} \\ \vdots \\ \frac{\partial \alpha}{\partial y_d} \end{bmatrix}$$

$$\nabla_{y^T} \alpha = \begin{bmatrix} \frac{\partial \alpha}{\partial y_1} & \frac{\partial \alpha}{\partial y_2} & \cdots \end{bmatrix}$$

$$\left(\nabla_y \alpha\right)^T = \nabla_{y^T} \alpha$$

Statement of theorem [edit]

Bayes' theorem is stated mathematically as the following equation:[4]

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

posterior likelihood prior

where A and B are events and $P(B) \neq 0$.

- $P(A \mid B)$ is a conditional probability: the probability of event A occurring given that B is true. It is also called the posterior probability of A given B.
- $P(B \mid A)$ is also a conditional probability: the probability of event B occurring given that A is true. It can also be interpreted as the likelihood of A given a fixed B because $P(B \mid A) = L(A \mid B)$.
- $P(A)$ and $P(B)$ are the probabilities of observing A and B respectively without any given conditions; they are known as the marginal probability or prior probability.

Proof [edit]

For events [edit]

Bayes' theorem may be derived from the definition of conditional probability:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}, \text{ if } P(B) \neq 0,$$

where $P(A \cap B)$ is the probability of both A and B being true. Similarly,

$$P(B \mid A) = \frac{P(A \cap B)}{P(A)}, \text{ if } P(A) \neq 0,$$

Solving for $P(A \cap B)$ and substituting into the above expression for $P(A \mid B)$ yields Bayes' theorem:

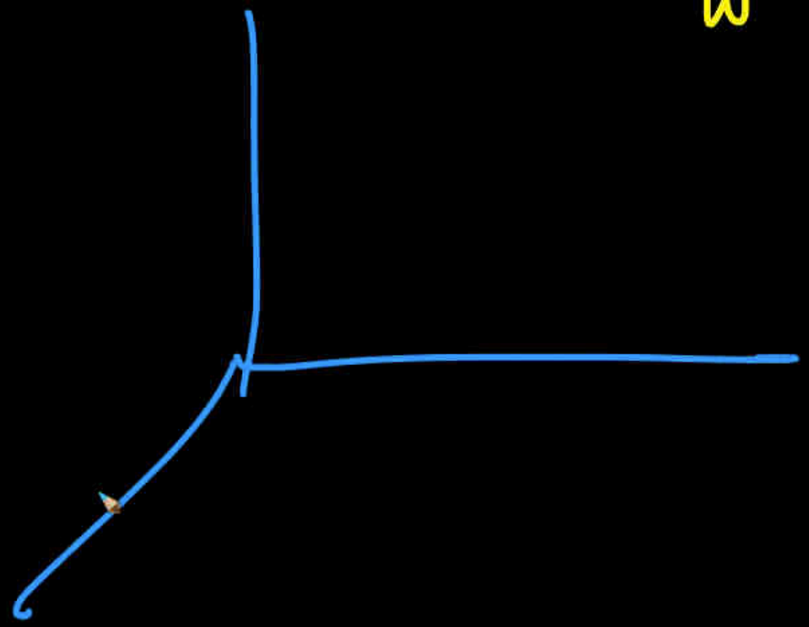
$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}, \text{ if } P(B) \neq 0.$$

For continuous random vari

For two continuous random variables X and Y, Bayes' theorem may be analogously derived from the definition of conditional density:

ML: vec$\tilde{w}$ -der

$$\nabla_{\tilde{w}} \widetilde{Loss} = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \\ \frac{\partial L}{\partial w_j} \end{bmatrix}$$



Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:40

google.com/search?q=plot(x%5E2%2By%5E2&rlz=1C5CHFA_enIN958IN958&oq=plot(x%5E2%2By%5E2&aqs=chrome..69i57j0i30l9.3644j0j4&sourceid=chrome&ie=UTF-8

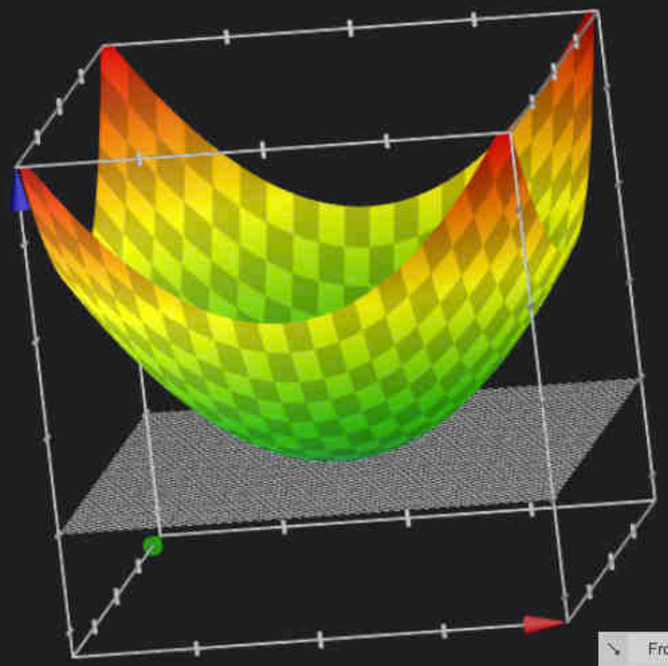
Google

plot(x^2+y^2

All Images News Maps Videos More Tools

About 73,60,00,000 results (0.66 seconds)

Graph for x^2+y^2



	From	To
x	-10.0000	10.0000
y	-10.0000	10.0000
z	-63.1513	189.482

17 / 17

Chrome File Edit View History Bookmarks Profiles Tab Window Help

google.com/search?q=plot(x%5E2%2By%5E2&rlz=1C5CHFA_enIN958IN958&oq=plot(x%5E2%2By%5E2&aqs=chrome..69i57j0i30l9.3644j0j4&sourceid=chrome&ie=UTF-8

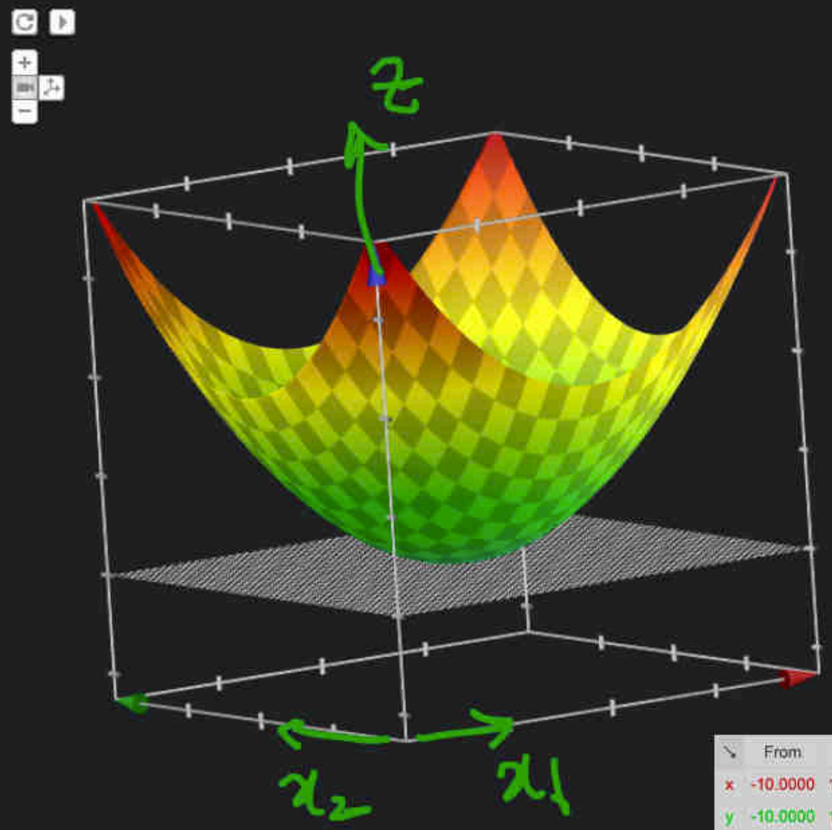
Google

plot(x^2+y^2

All Images News Maps Videos More Tools

About 73,60,00,000 results (0.66 seconds)

Graph for x^2+y^2



	From	To
x	-10.0000	10.0000
y	-10.0000	10.0000
z	-63.1513	189.482

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

scalar

$$z = x^2 + y^2$$

$$z = x_1^2 + x_2^2$$

$$z = x^T x$$

$$\nabla_x z = \begin{bmatrix} \frac{\partial z}{\partial x_1} \\ \frac{\partial z}{\partial x_2} \end{bmatrix}$$

18 / 18

Chrome File Edit View History Bookmarks Profiles Tab Window Help

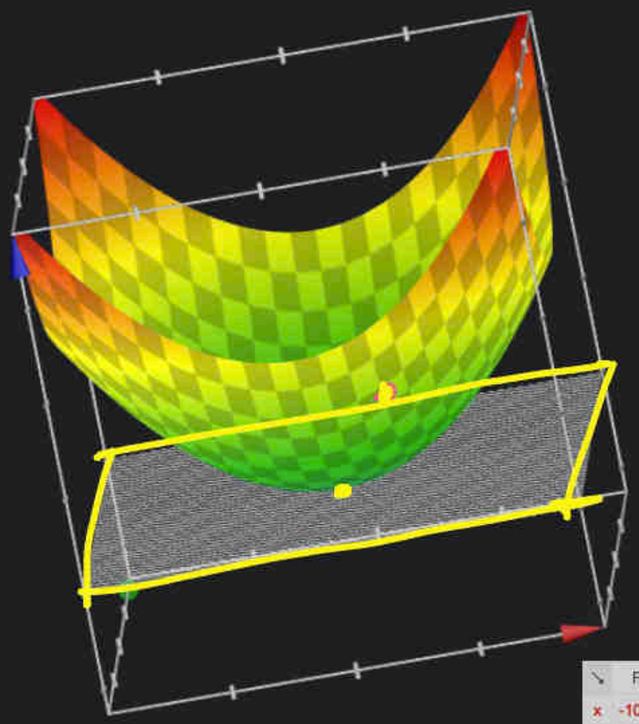
google.com/search?q=plot(x%5E2%2By%5E2&rlz=1C5CHFA_enIN958IN958&oq=plot(x%5E2%2By%5E2&aqs=chrome..69i57j0i30l9.3644j0j4&sourceid=chrome&ie=UTF-8

Google plot(x^2+y^2

All Images News Maps Videos More Tools

About 73,60,00,000 results (0.66 seconds)

Graph for x^2+y^2



	From	To
x	-10.0000	10.0000
y	-10.0000	10.0000
z	-63.1513	189.482

More info

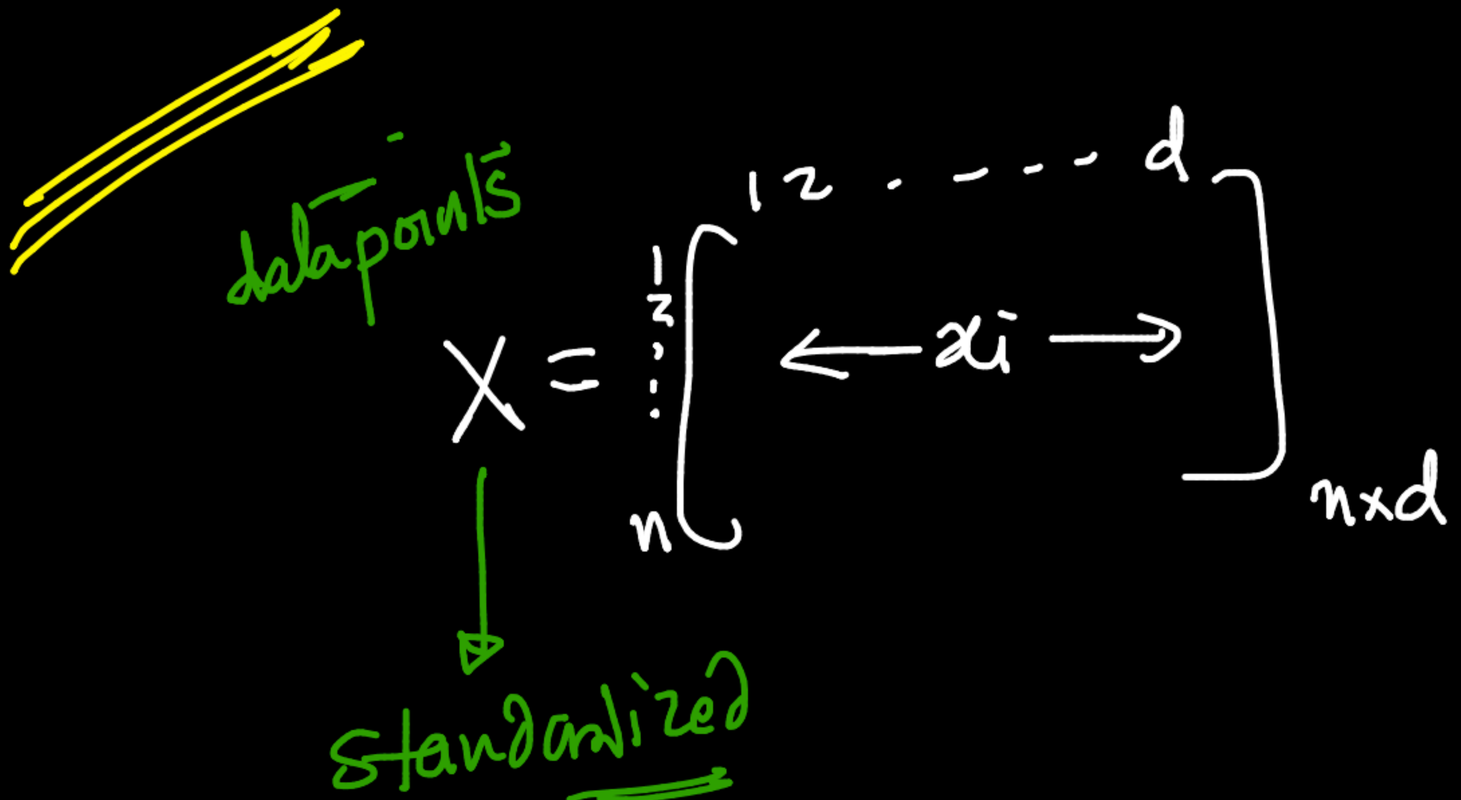
Slope of the tangent plane

Rate of change of z w.r.t x_1 as x_1 changes

$$\nabla_x z = \begin{bmatrix} \frac{\partial z}{\partial x_1} \\ \frac{\partial z}{\partial x_2} \end{bmatrix}$$

19 / 19

PCA - recap



$$S_{d \times d} : \text{cov}(X) = X^T X$$

↳ symmetric

optimization

$$\min_{u} \sum_{i=1}^{n} (u^T x_i)^2$$

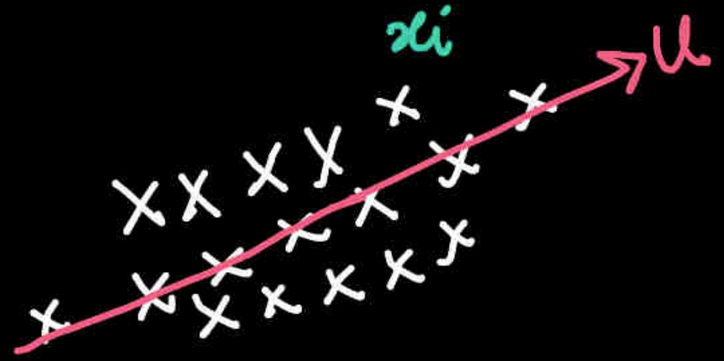
$$\text{s.t } u^T u = 1$$

$$\Downarrow$$

$$\min_{u} \underset{1\times d}{u^T} \underset{d\times d}{S} \underset{d\times 1}{u}$$

$$\text{s.t } u^T u = 1$$

geom



$$\bar{x} = \text{mean}(x_i) = \vec{0}$$

$$u^T \bar{x} = \vec{0}$$

$$\min_u \quad \underbrace{u^T S u + \lambda u^T u}_{\mathcal{L}} \longrightarrow \text{One minima}$$

$$@\ u^* \qquad \frac{\partial \mathcal{L}}{\partial u} = 0$$

$$= 2u^T S + 2\lambda u^T = 0$$

$$u^T S = -\lambda u^T$$

$$S^T u = u\lambda'$$

$$S u = \lambda' u$$

eigenval & eige vec

best u → eigvec corresponding to largest eig-val

$\forall i := 1 \to n$

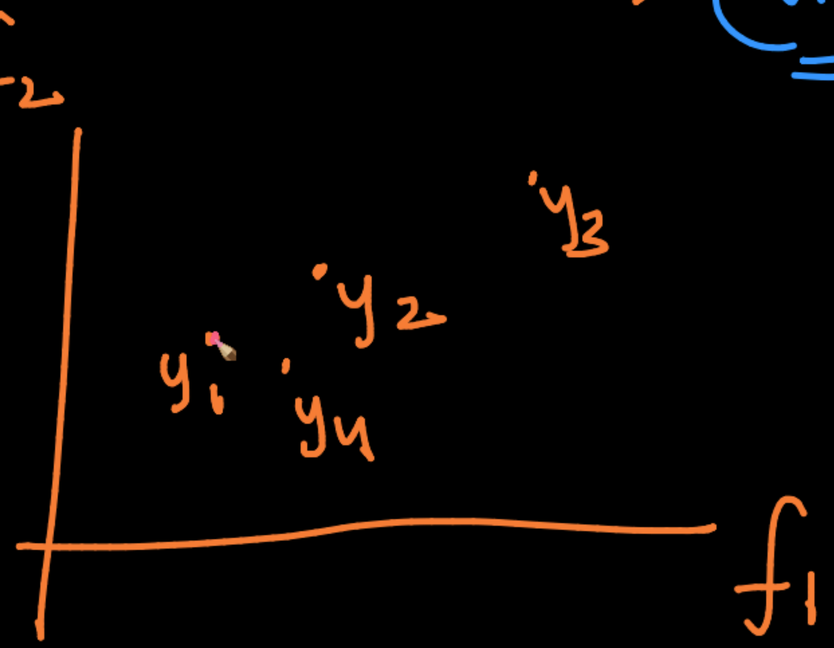
$x_i \in \mathbb{R}^d$

Project → top 2 eig vec → NOT labels

$y_i \in \mathbb{R}^2$

PCA for dim-red (viz)

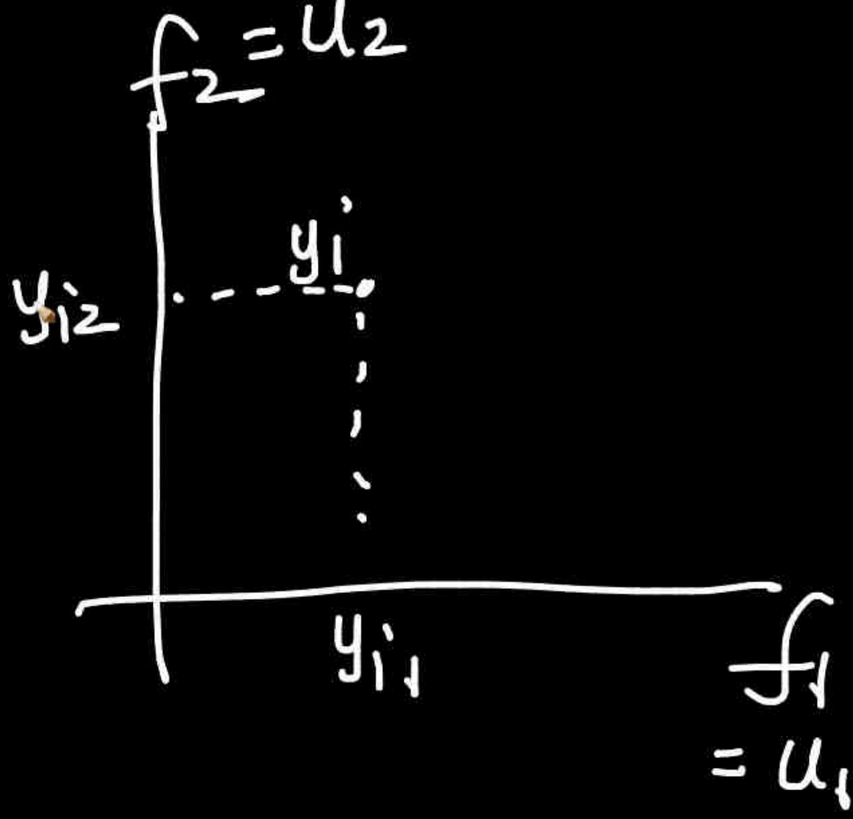
u_1 & u_2 : 2 largest eig-vec



$$\left[u_1^T x_i \quad u_2^T x_i \right]^T = y_i$$

y_{i1}

y_{i2}



1

PCA : Linear mapping

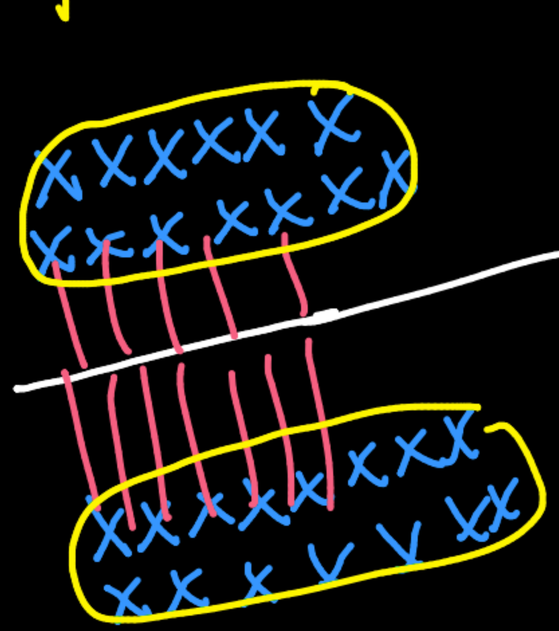
$$y_i_{2\times 1} = \begin{bmatrix} \leftarrow u_1^T \rightarrow \\ \leftarrow u_2^T \rightarrow \end{bmatrix}_{2\times d} x_i_{d\times 1}$$

$$\tilde{y}_i = U \cdot \tilde{x}_i \longrightarrow \text{Linear-mapping}$$

$\tilde{x}_i$: d-dim

$\tilde{y}_i$: 2-dim rep. of x_i

(2) does Not often preserve the local/neighborhood info



messing-up

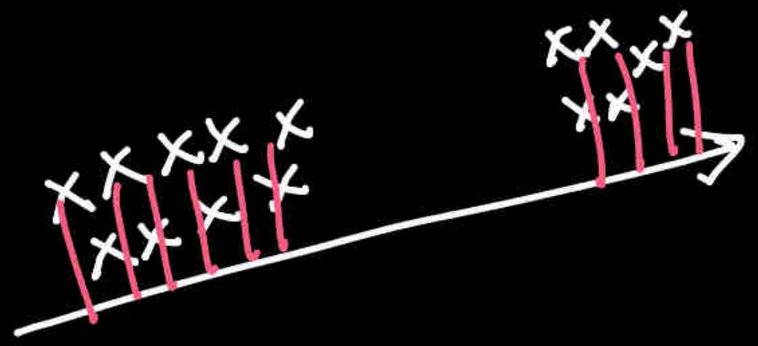
$f_1 = u_1$

2D
d-dim

PCA

1D
d'-dim

presenving Neighbors



2D

1D

high dim viz using clustering

$x_i \in \mathbb{R}^d$

2D

- polar plots
- parallel coord plots
- hierarchical clustering (tree)

Q

PCA for regression ?

$x_i \in R^d$

d : V · V · large

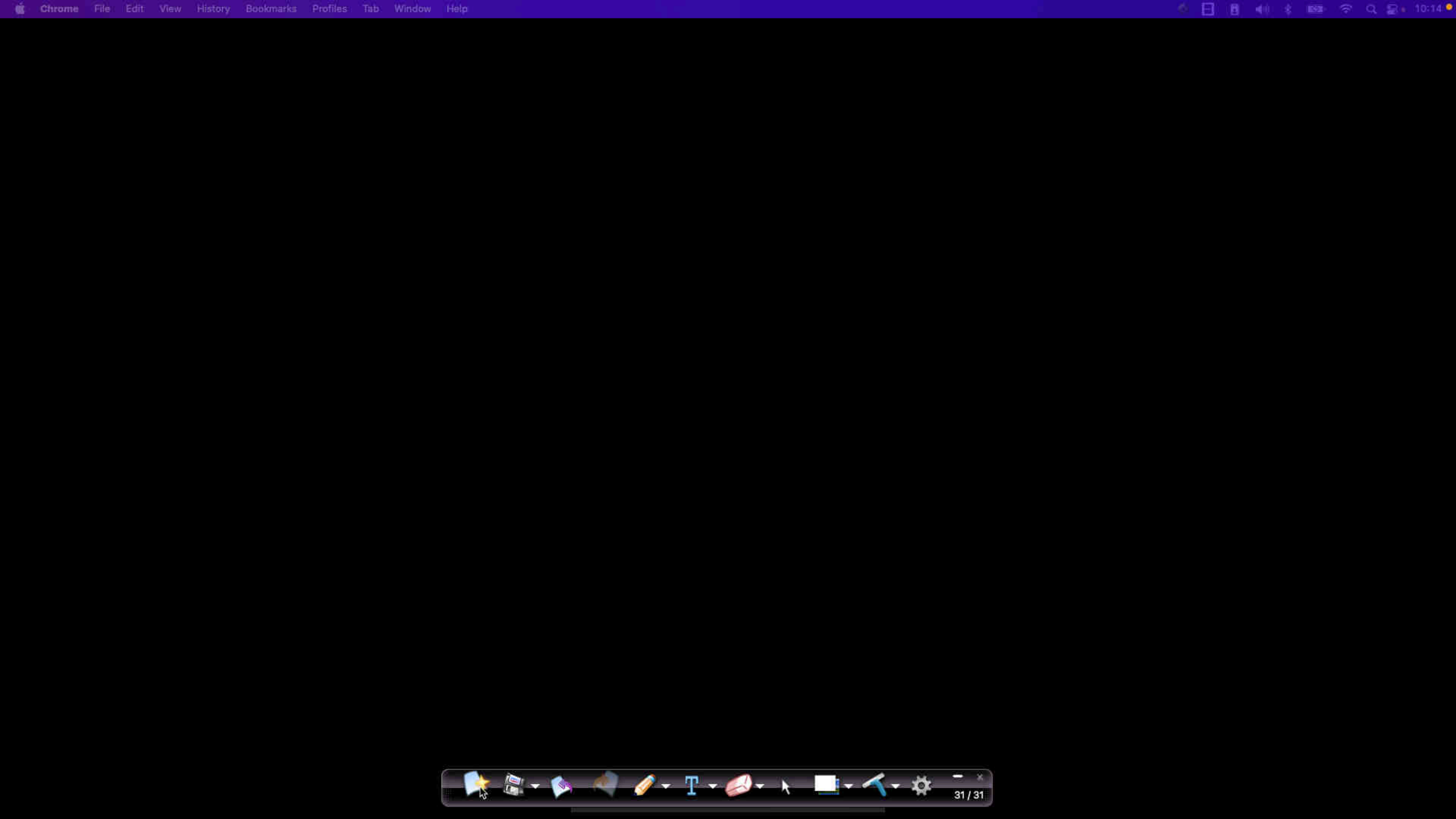
PCA → loss of info → Can be avoided

d' → build model

Can be done

→ better soln :- L1-Regularization

Heavy (large λ)

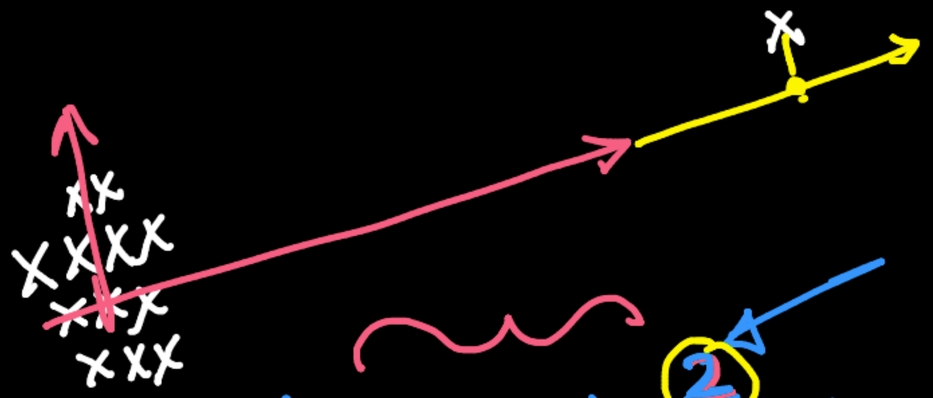

Chrome File Edit View History Bookmarks Profiles Tab Window Help
10:14
31 / 31

3

PCA: v. low noise tolerance

Why?

geom:



Math:

$$\min_u \sum_{i=1}^{n} \left(u^T x_i \right)^2 + \lambda u^T u$$

How to solve it?

- ✓ → remove outliers using stats method
- ✓ → RANSAC-like
- L1-PCA →

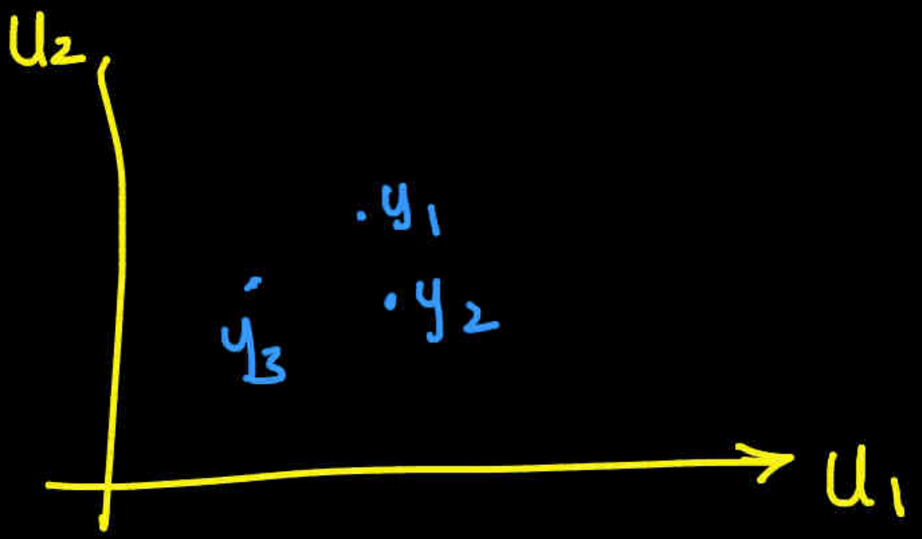
$$\min_u \sum_{i=1}^{n} \underbrace{|u^T x_i|}_{\text{abs-value}} + \lambda u^T u$$

$\lambda u^T u$ → L_2

Q

$x_i \in \mathbb{R}^d \xrightarrow{\text{PCA}} y_i \in \mathbb{R}^2 \rightsquigarrow$ 2 or 3

Viz

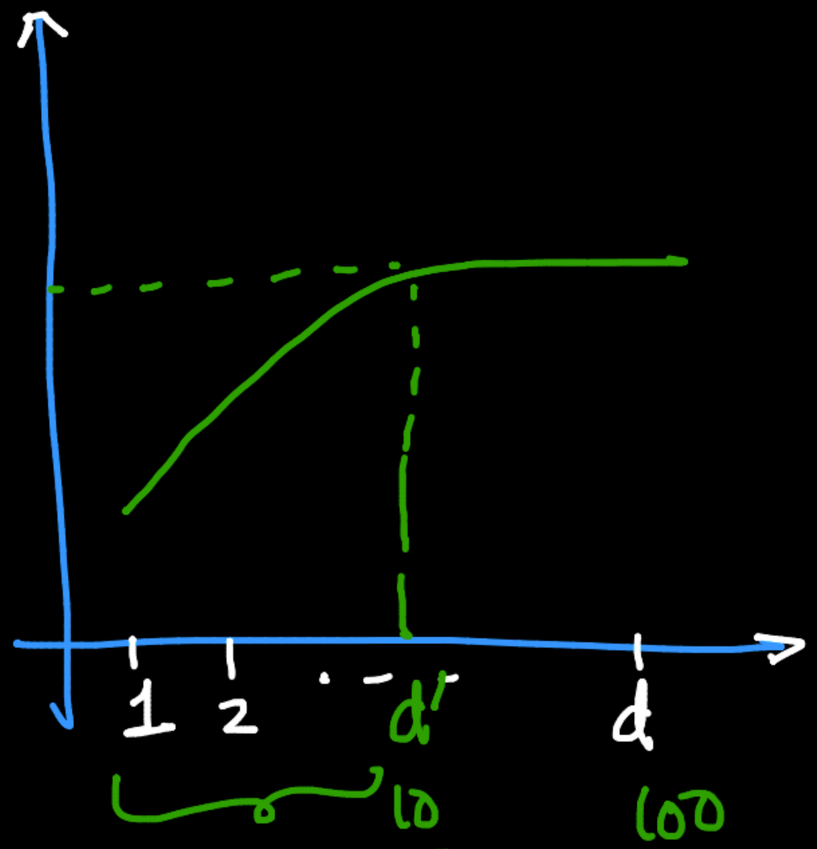


2D

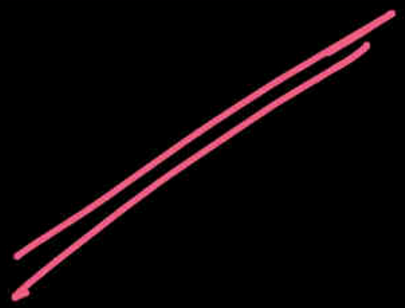
Dim-red

100 $d \rightarrow d'$ 10

$d' < d$

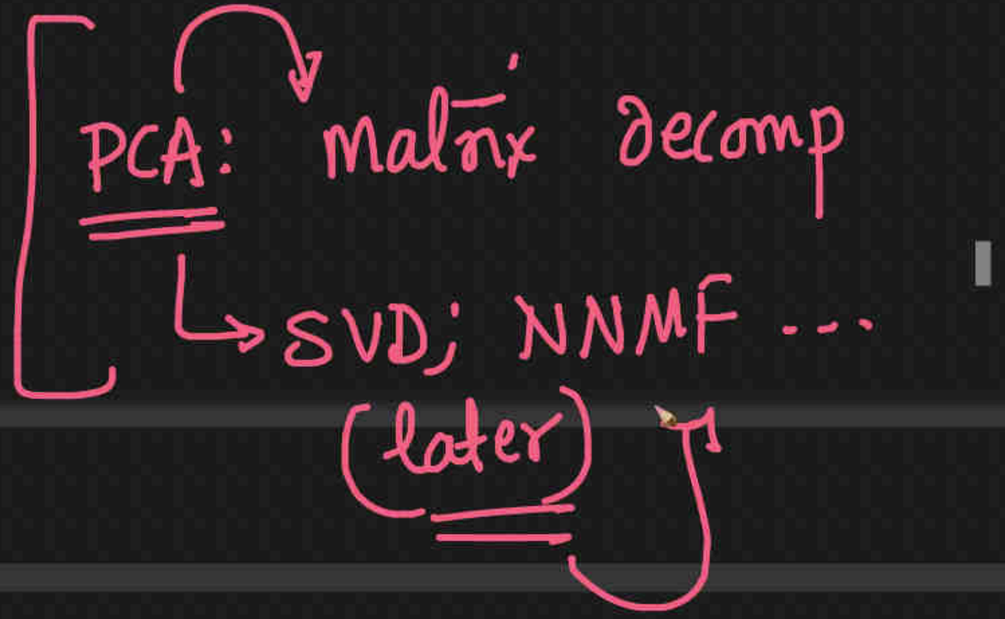


95% Avar → doesn't guarantee Neighb preservation



```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import cm
import seaborn as sns

from openTSNE import TSNE
from umap import UMAP
from sklearn import decomposition
```




```
[ ] from sklearn.datasets import load_digits
    digits = load_digits()
```

```
[ ] digits.keys()
```

```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DESCR'])
```

```
[ ] X = digits.data
    Y = digits.target
```

```
[ ] X.shape
```

```
from sklearn import decomposition
```

```
from sklearn.datasets import load_digits
digits = load_digits()
```

0 1 2 3 4 5 6 7 8 9

```
digits.keys()
```

```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DESCR'])
```


```
X = digits.data
Y = digits.target
```

```
X.shape
```

```
(1797, 64)
```

```
digits.images.shape
```

```
(1797, 8, 8)
```

```
plt.gray()
plt.imshow(digits.images[10
```

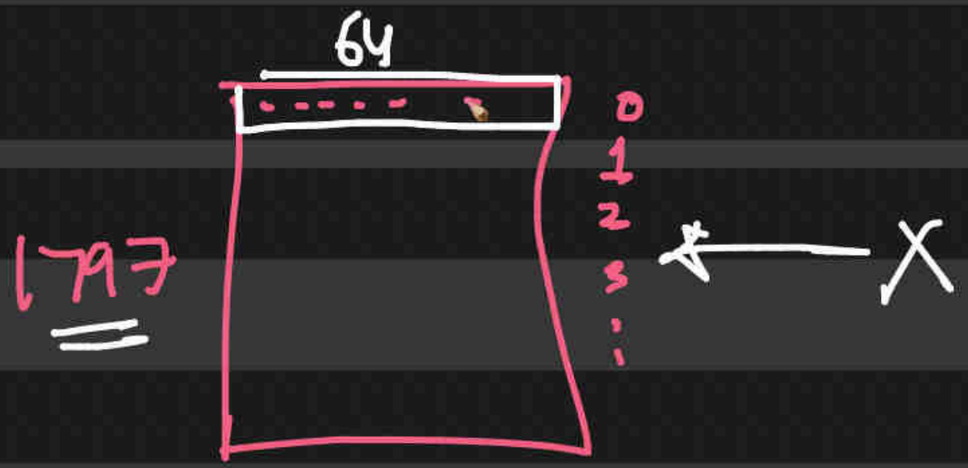
```
from sklearn import decomposition
```

```
from sklearn.datasets import load_digits
digits = load_digits()
```

```
digits.keys()
```

```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DESCR'])
```

```
X = digits.data
Y = digits.target
```




```
X.shape
```

```
(1797, 64)
```

```
digits.images.shape
```

```
(1797, 8, 8)
```

```
plt.gray()
plt.imshow(digits.images[10
```

+ Code + Text Reconnect

```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DESCR'])
```

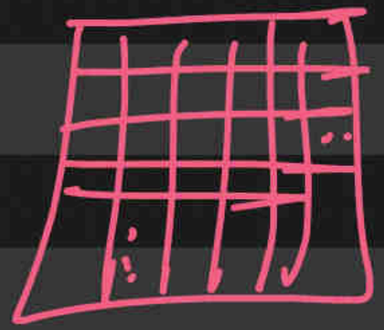
```
[ ] X = digits.data
    Y = digits.target
```

```
[ ] X.shape
```

```
(1797, 64)
```

```
[ ] digits.images.shape
```

```
(1797, 8, 8)
```



0 — 255
B W

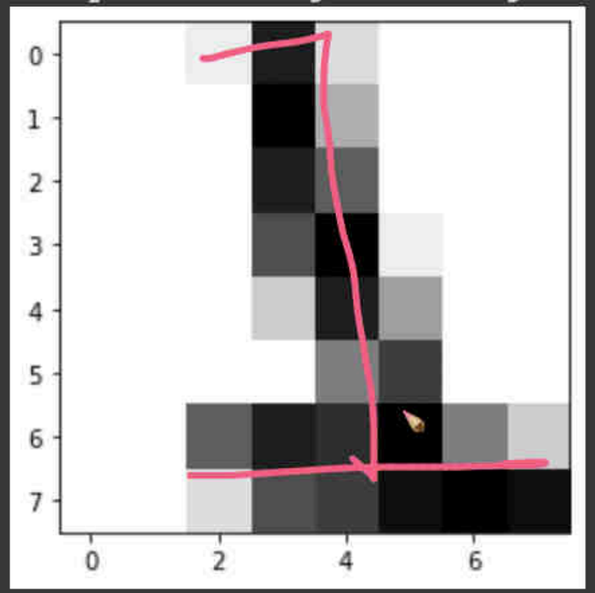
```
plt.gray()
plt.imshow(digits.images[1000], cmap=plt.cm.gray_r)
```

```
<matplotlib.figure.AxesImage at 0x7f5b003e5110>
```

39 / 39

```
[ ]  plt.gray()
     plt.imshow(digits.images[1000], cmap=plt.cm.gray_r)
```

```
<matplotlib.image.AxesImage at 0x7f5b003e5110>
```




```
[ ]  #PCA
     %%time
     pca_2D = decomposition.PCA(n_components=2)
     pca_2D.fit(X)
     Z1 = pca_2D.transform(X)
```

+ Code + Text Reconnect

```
[ ] Y = digits.target
```

```
[ ] X.shape
```

```
(1797, 64)
```

64x64

```
[ ] digits.images.shape
```

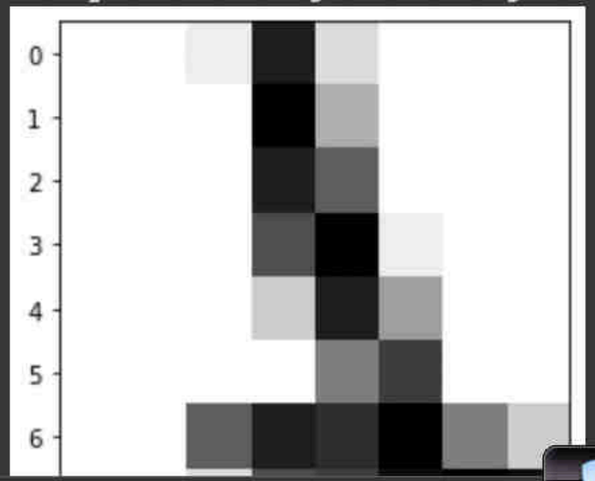
```
(1797, 8, 8)
```

1920

1080

```
[ ] plt.gray()
    plt.imshow(digits.images[1000], cmap=plt.cm.gray_r)
```

```
<matplotlib.image.AxesImage at 0x7f5b003e5110>
```



```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DESCR'])
```

```
X = digits.data
Y = digits.target
```

```
X.shape
```

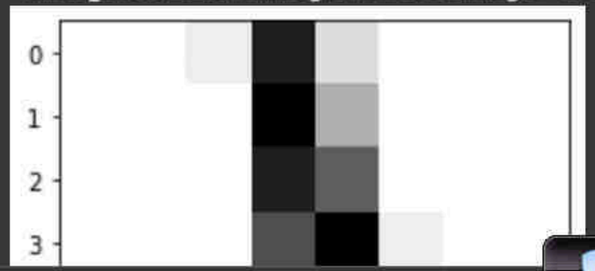
```
(1797, 64)
```

```
digits.images.shape
```

```
(1797, 8, 8)
```

```
plt.gray()
plt.imshow(digits.images[1000], cmap=plt.cm.gray_r)
```

```
<matplotlib.image.AxesImage at 0x7f5b003e5110>
```



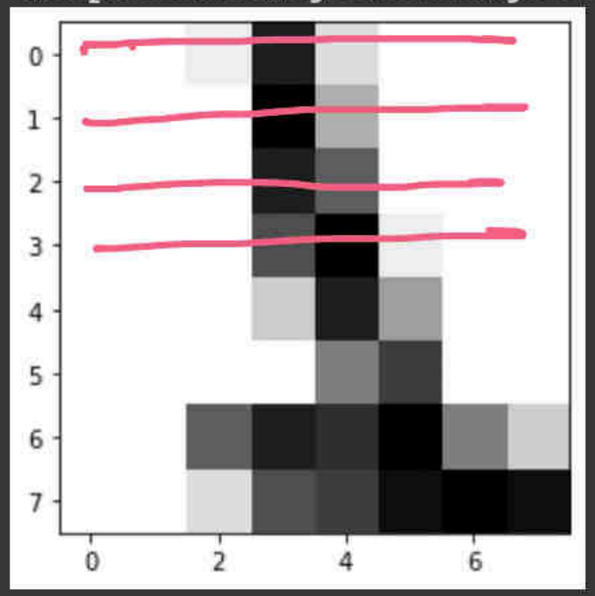
+ Code + Text Reconnect

```
[ ] digits.images.shape
```

```
(1797, 8, 8)
```

```
[ ] plt.gray()
    plt.imshow(digits.images[1000], cmap=plt.cm.gray_r)
```

```
<matplotlib.image.AxesImage at 0x7f5b003e5110>
```

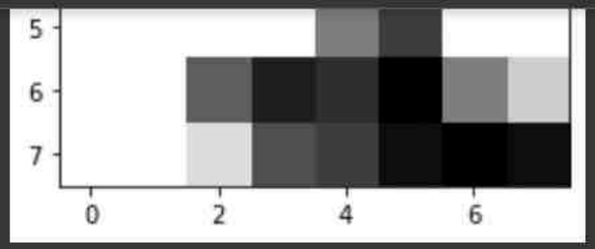


255 255

8X8 → 64

```
[ ] #PCA
    %%time
    pca_2D = decomposition.PCA(
```

[ ]



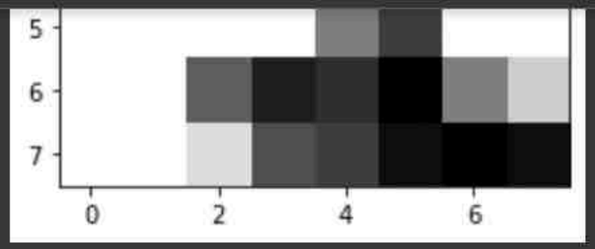
64-dim → 2-dim

```
[ ] #PCA
    %%time
    pca_2D = decomposition.PCA(n_components=2)
    pca_2D.fit(X)
    Z1 = pca_2D.transform(X)
```

```
CPU times: user 18.2 ms, sys: 8.02 ms, total: 26.2 ms
Wall time: 18.4 ms
```

```
[ ] from matplotlib.colors import ListedColormap
    cmap = ListedColormap(sns.husl_palette(len(np.unique(Y))))
```

```
fig, ax = plt.subplots()
im = ax.scatter(Z1[:,0], Z1[:,1], s=10, c=Y, cmap=cmap, edgecolor='none')
cbar = fig.colorbar(im, ax=ax,label='Digit')
```



X : 1797x64

```
[ ] #PCA
    %%time
    pca_2D = decomposition.PCA(n_components=2)
    pca_2D.fit(X)
    Z1 = pca_2D.transform(X)
```

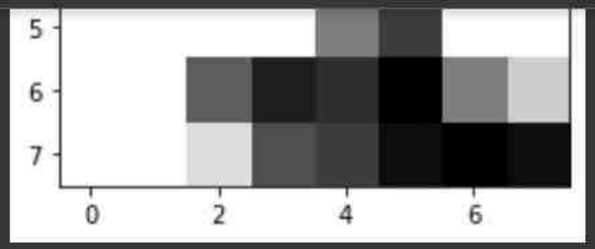
64.81M → 29.1M

```
    CPU times: user 18.2 ms, sys: 8.02 ms, total: 26.2 ms
    Wall time: 18.4 ms
```

```
[ ] from matplotlib.colors import ListedColormap
    cmap = ListedColormap(sns.husl_palette(len(np.unique(Y))))
```

```
    fig, ax = plt.subplots()
    im = ax.scatter(Z1[:,0], Z1[:,1], s=10, c=Y, cmap=cmap, edgecolor='none')
    cbar = fig.colorbar(im, ax=ax,label='Digit')
```

[ ]



[ ]
```
#PCA
%%time
pca_2D = decomposition.PCA(n_components=2)
pca_2D.fit(X)
Z1 = pca_2D.transform(X)
```

u1 & u2

$Z1_i = \begin{bmatrix} -u_1- \\ -u_2- \end{bmatrix} x_i$

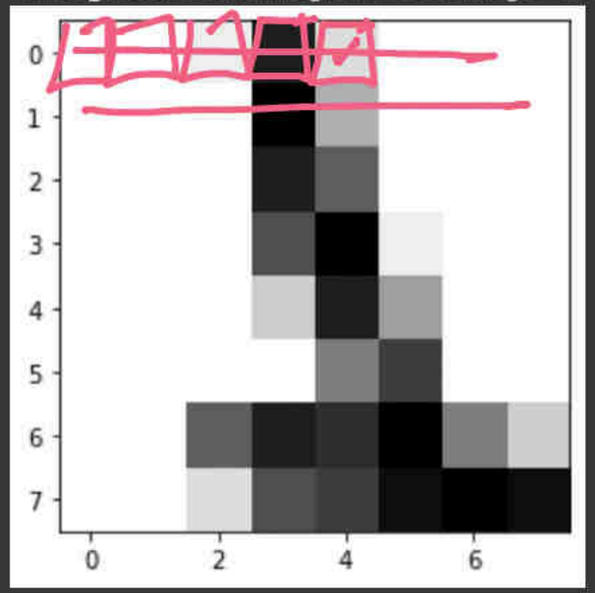
```
CPU times: user 18.2 ms, sys: 8.02 ms, total: 26.2 ms
Wall time: 18.4 ms
```

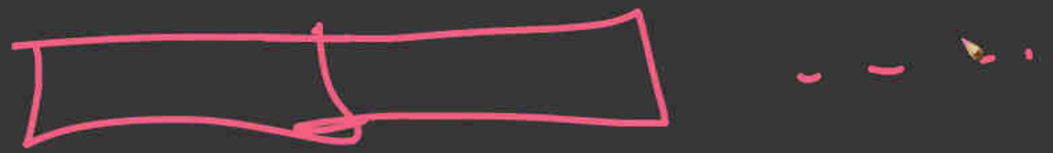
[ ]
```
from matplotlib.colors import ListedColormap
cmap = ListedColormap(sns.husl_palette(len(np.unique(Y))))
```

```
fig, ax = plt.subplots()
im = ax.scatter(Z1[:,0], Z1[:,1], s=10, c=Y, cmap=cmap, edgecolor='none')
cbar = fig.colorbar(im, ax=ax,label='Digit')
```

```
plt.gray()
plt.imshow(digits.images[1000], cmap=plt.cm.gray_r)
```

```
<matplotlib.image.AxesImage at 0x7f5b003e5110>
```





```
#PCA
%%time
pca_2D = decomposition.PCA(n_components=2)
pca_2D.fit(X)
Z1 = pca_2D.transform(X)
```

```
CPU times: user 18.2 ms, sys: 8.02 ms, total: 26.2 ms
```

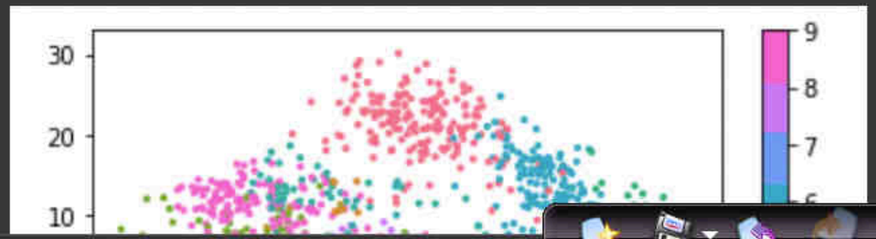
[ ]

[ ]
```
#PCA
%%time
pca_2D = decomposition.PCA(n_components=2)
pca_2D.fit(X)
Z1 = pca_2D.transform(X)
```

```
CPU times: user 18.2 ms, sys: 8.02 ms, total: 26.2 ms
Wall time: 18.4 ms
```

[ ]
```
from matplotlib.colors import ListedColormap
cmap = ListedColormap(sns.husl_palette(len(np.unique(Y))))
```

```
fig, ax = plt.subplots()
im = ax.scatter(Z1[:,0], Z1[:,1], s=10, c=Y, cmap=cmap, edgecolor='none')
cbar = fig.colorbar(im, ax=ax,label='Digit')
```



[ ]

[ ]
```
#PCA
%%time
pca_2D = decomposition.PCA(n_components=2)
pca_2D.fit(X)
Z1 = pca_2D.transform(X)
```

64次元 → 2次元

x_i

```
CPU times: user 18.2 ms, sys: 8.02 ms, total: 26.2 ms
Wall time: 18.4 ms
```

[ ]
```
from matplotlib.colors import ListedColormap
cmap = ListedColormap(sns.husl_palette(len(np.unique(Y))))
```

```
fig, ax = plt.subplots()
im = ax.scatter(Z1[:,0], Z1[:,1], s=10, c=Y, cmap=cmap, edgecolor='none')
cbar = fig.colorbar(im, ax=ax,label='Digit')
```

+ Code + Text Reconnect

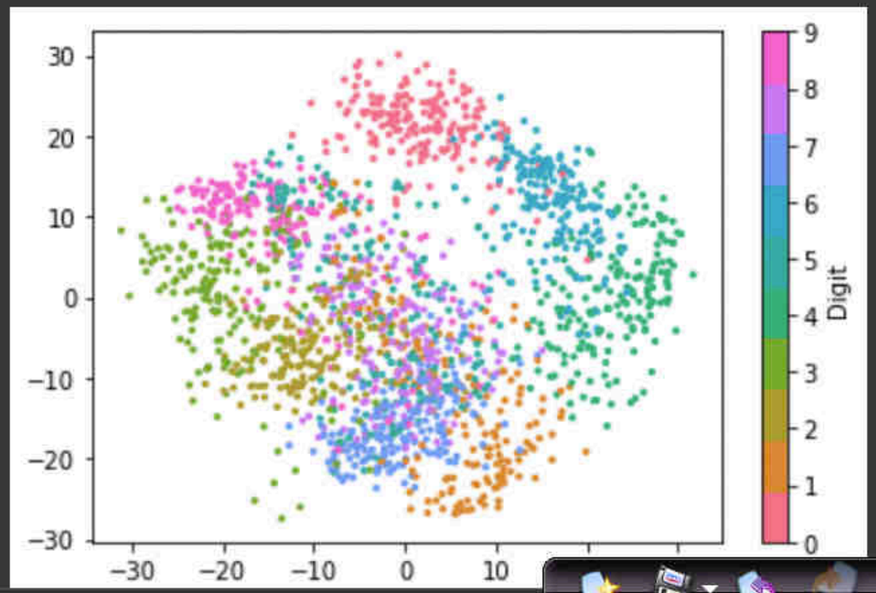
```
[ ]  Z1 = pca_2D.transform(X)
```

```
CPU times: user 18.2 ms, sys: 8.02 ms, total: 26.2 ms
Wall time: 18.4 ms
```

0, 1, 2 ... 9

```
[ ]  from matplotlib.colors import ListedColormap
     cmap = ListedColormap(sns.husl_palette(len(np.unique(Y))))
```

```
fig, ax = plt.subplots()
im = ax.scatter(Z1[:,0], Z1[:,1], s=10, c=Y, cmap=cmap, edgecolor='none')
cbar = fig.colorbar(im, ax=ax,label='Digit')
```

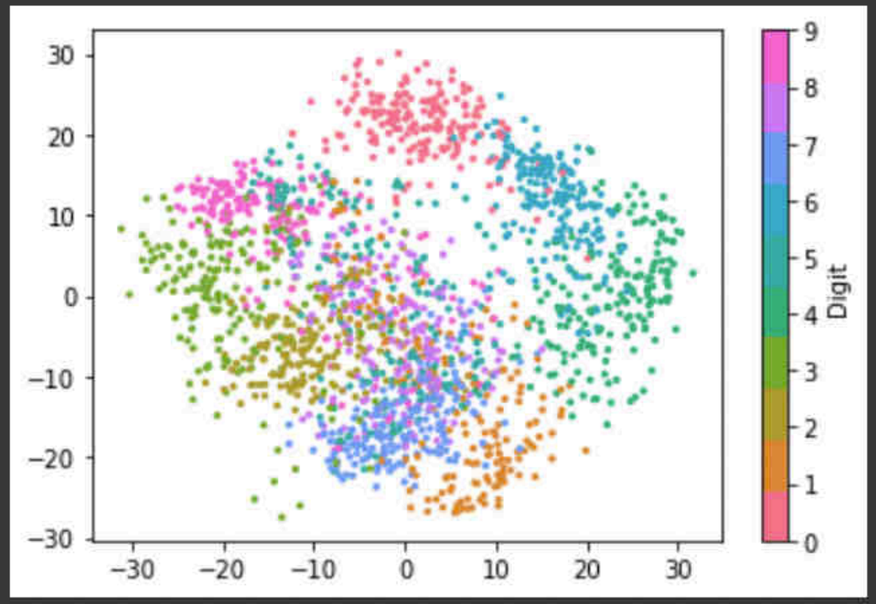


50 / 50

```
Wall time: 18.4 ms
```

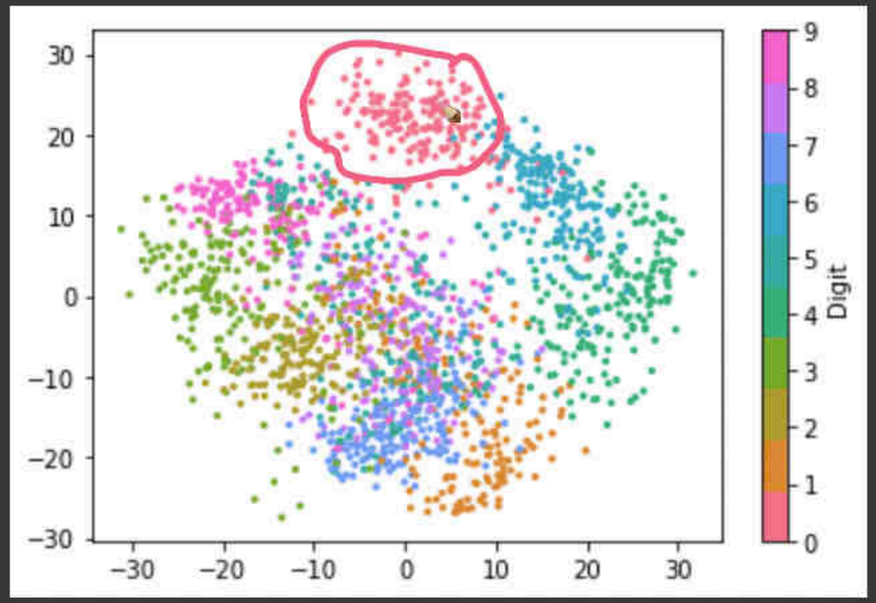
```
[ ] from matplotlib.colors import ListedColormap
    cmap = ListedColormap(sns.husl_palette(len(np.unique(Y))))
```

```
[ ] fig, ax = plt.subplots()
    im = ax.scatter(Z1[:,0], Z1[:,1], s=10, c=Y, cmap=cmap, edgecolor='none')
    cbar = fig.colorbar(im, ax=ax,label='Digit')
```




```
[ ] # tSNE
```

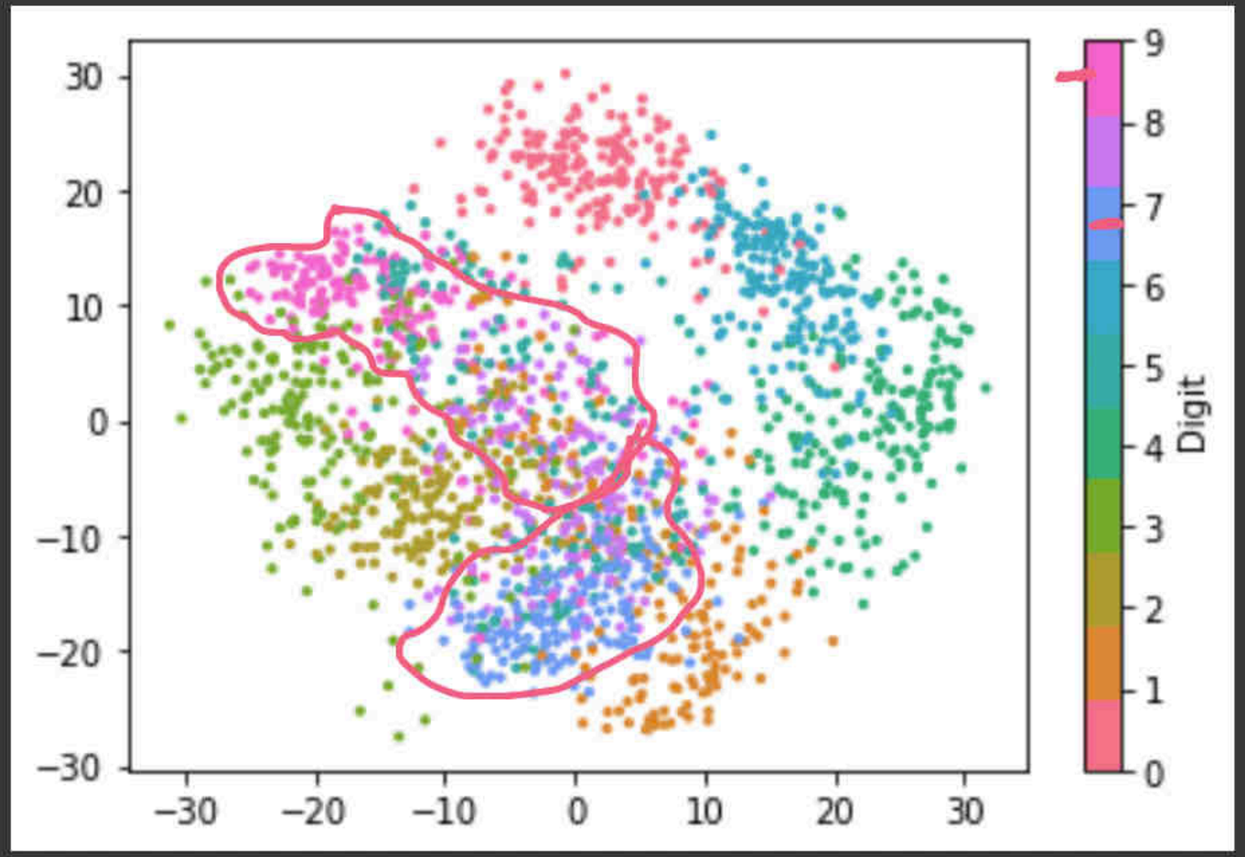
```
[ ] fig, ax = plt.subplots()
    im = ax.scatter(Z1[:,0], Z1[:,1], s=10, c=Y, cmap=cmap, edgecolor='none')
    cbar = fig.colorbar(im, ax=ax,label='Digit')
```




```
[ ] # tSNE
    %%time
    Z2 = TSNE(n_jobs=-1, initialization='random', random_state=42, negative_gradient_method='bh').fit(X)
```

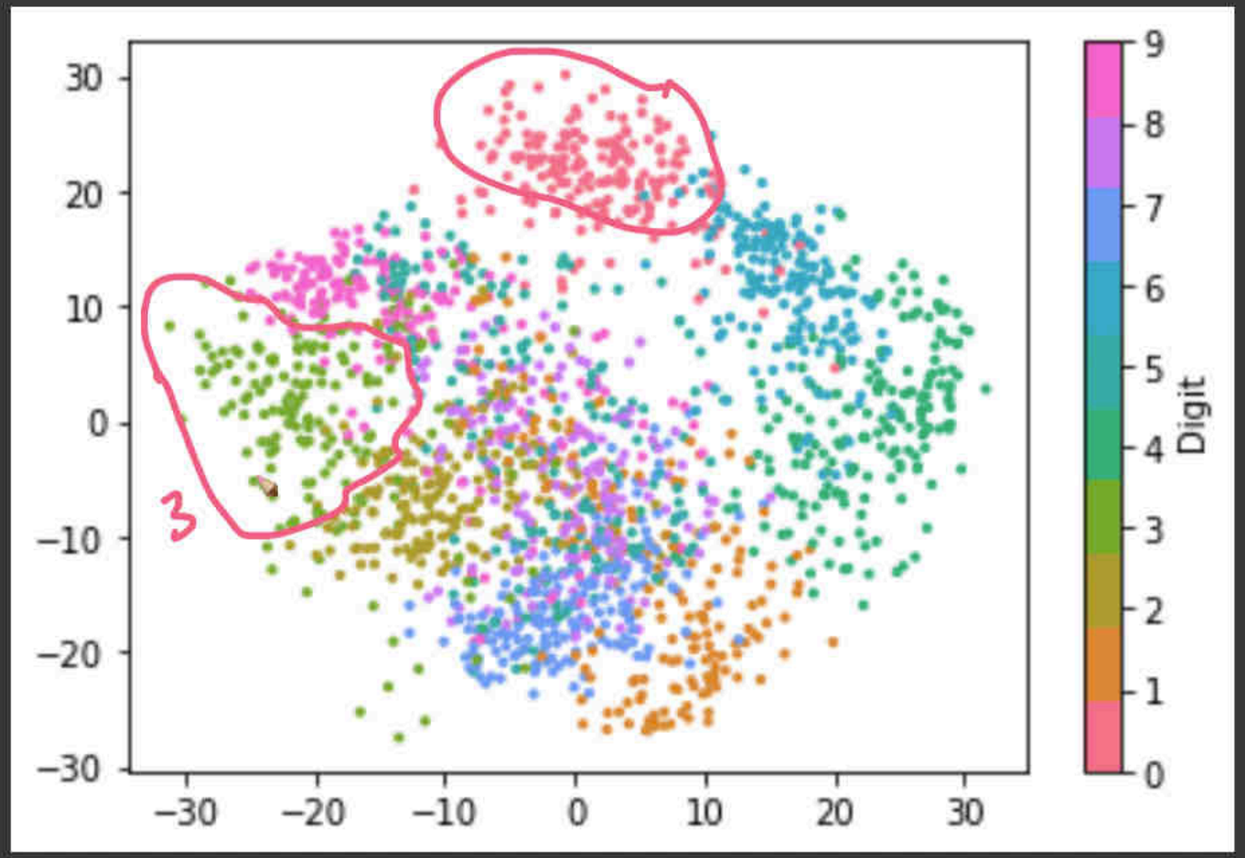
```
CPU times: user 14.6 s, sys: 179 ms, total: 14.8 s
Wall time: 8.32 s
```

[ ]

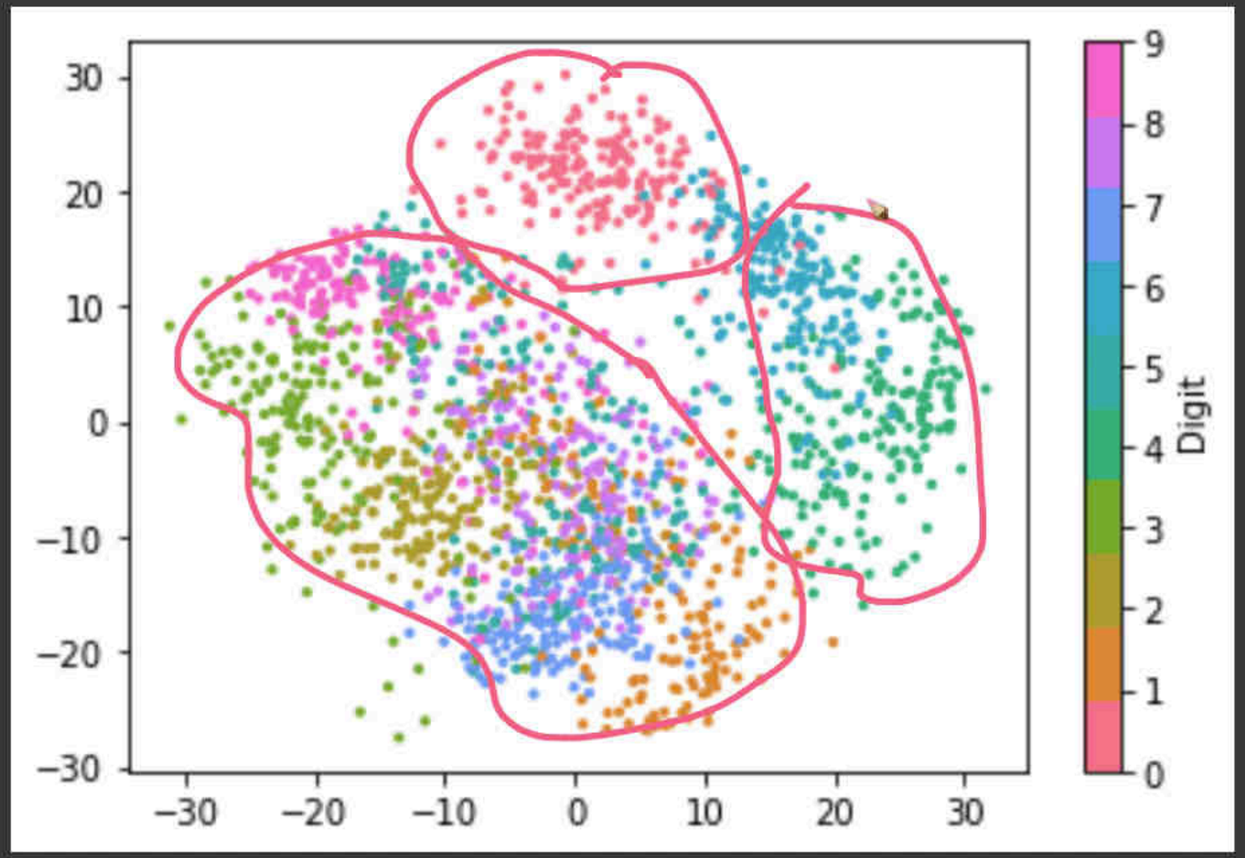


7 9

[ ] # tSNE




```
[ ]  # tSNE
```

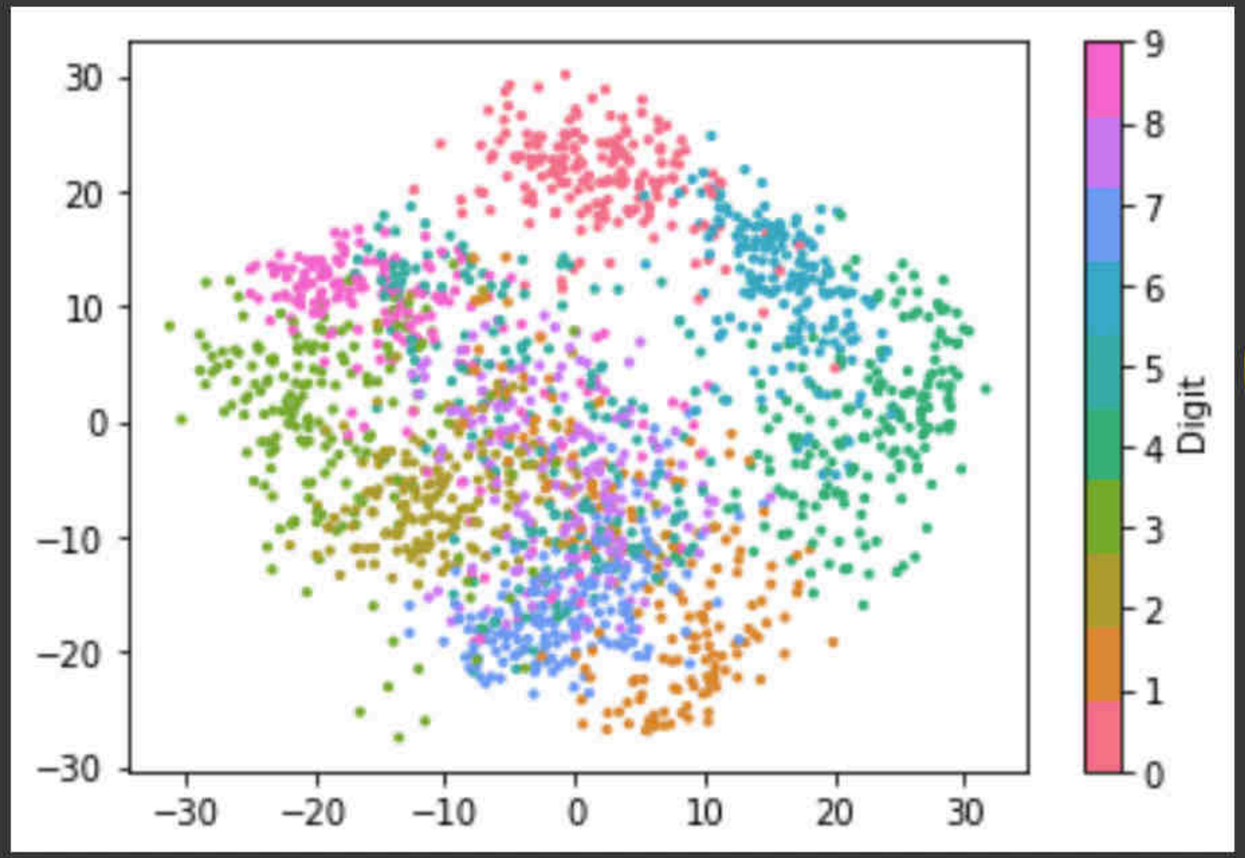


xi's 64 dim

↓

2 dim

```
[ ]  # tSNE
```

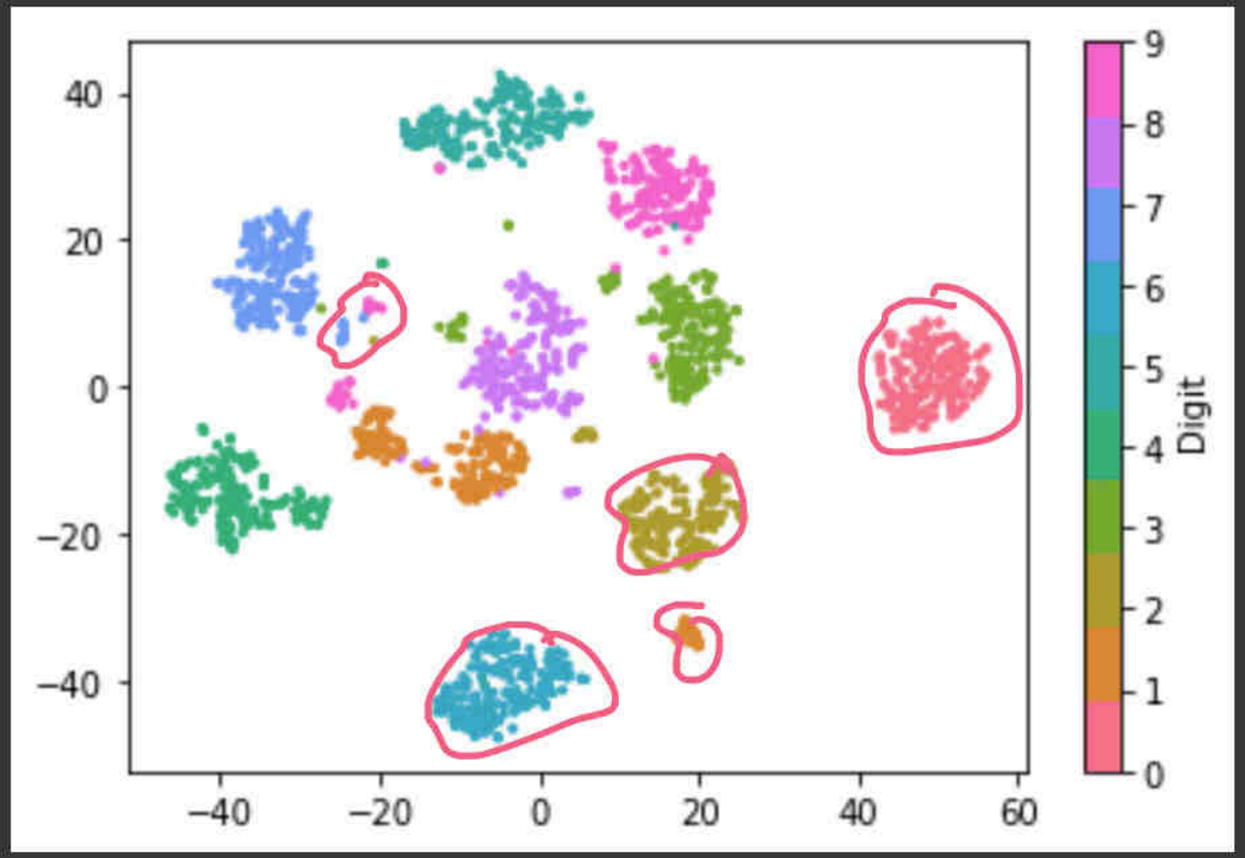


PCA

64 → 28ish

no-Neigh poss

[ ] # tSNE



tSNE

Visualizing Data using t-SNE

Laurens van der Maaten LVDMAATEN@GMAIL.COM
TiCC
Tilburg University
P.O. Box 90153, 5000 LE Tilburg, The Netherlands

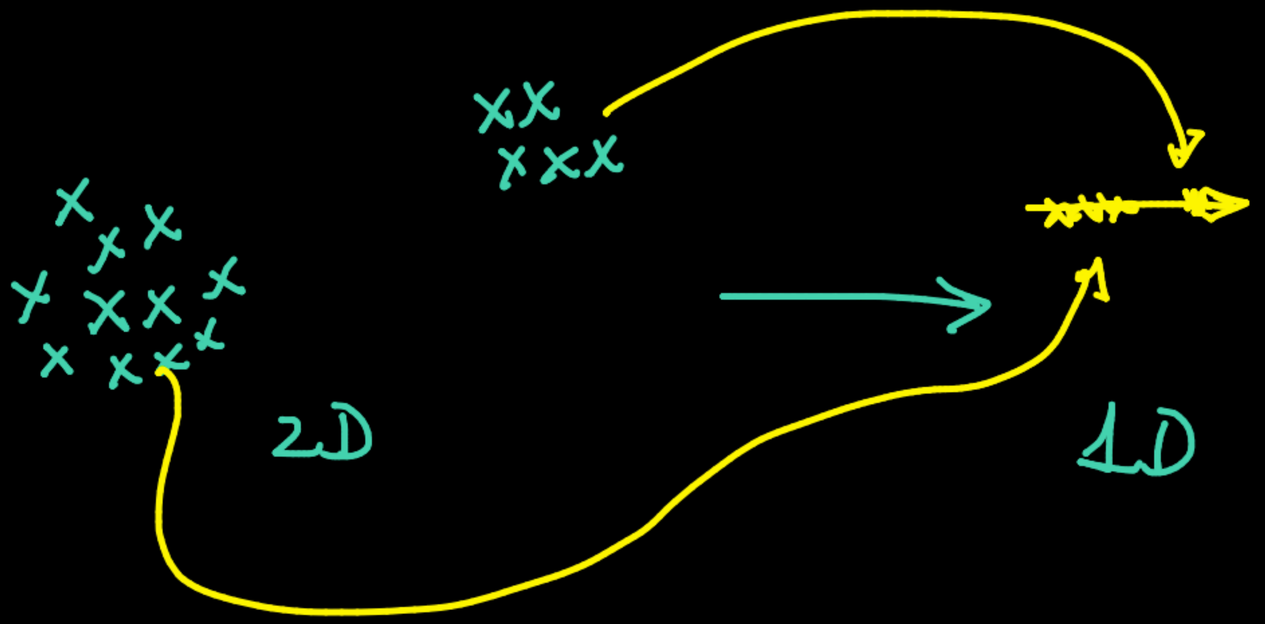
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
6 King's College Road, M5S 3G4 Toronto, ON, Canada

Editor: Yoshua Bengio

t-SNE:

t-distributed Stochastic Neighbor embedding

probabilistic



Core-idea!

Preserve all pairwise distances in a Neighborhood as best as possible

$x_i - x_j$

$x_j \in N(x_i)$

$d \longrightarrow d' = 2$ (typically)

$x_i \in \mathbb{R}^d$

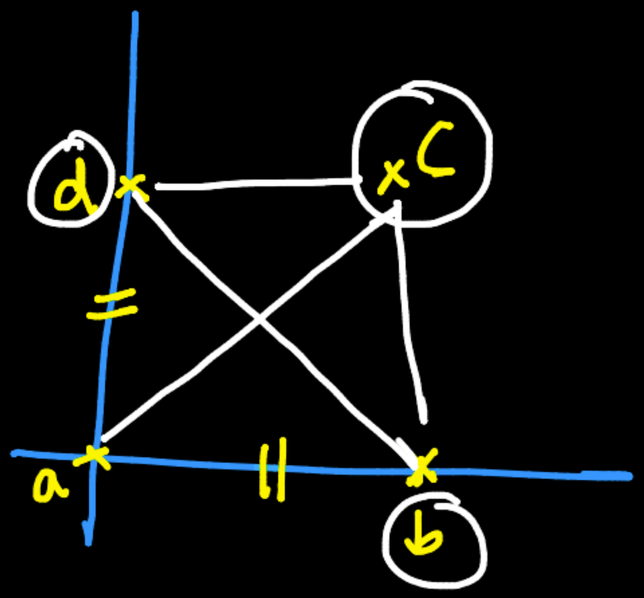
$y_i \in \mathbb{R}^{d'}$

$x_i \; x_j$

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:56

Crowding

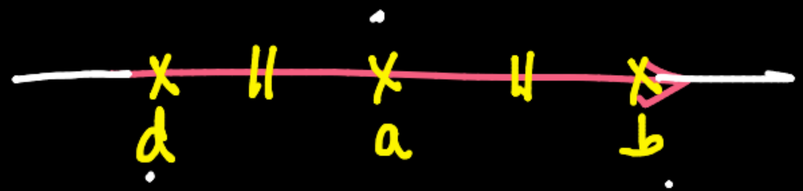
$aC > ad = ab$



2D

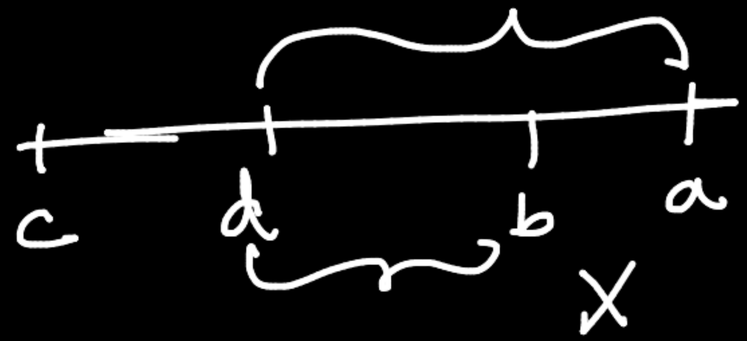
N·E

[No-way]

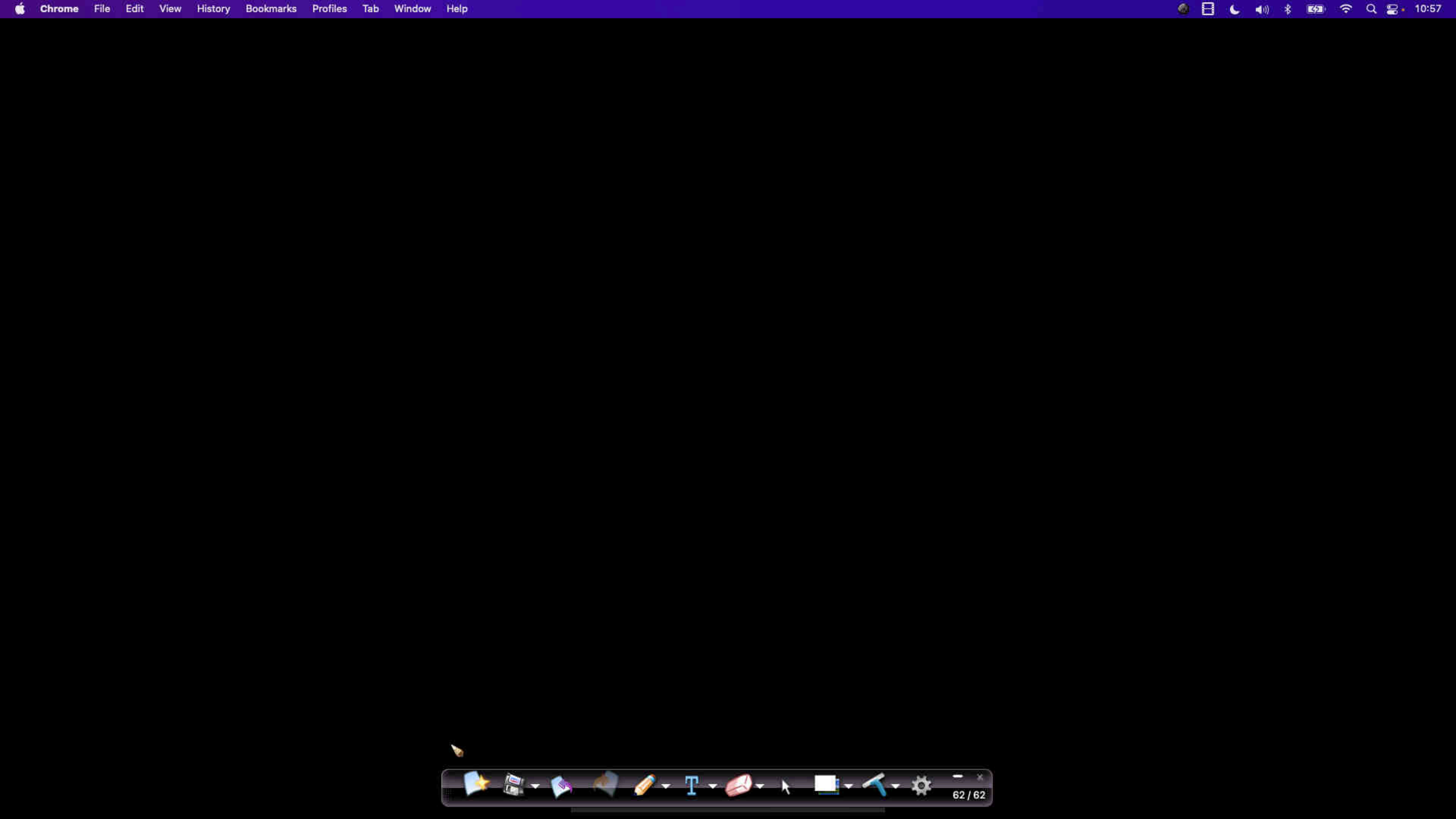


1D

Let $N(A) = \{b, c, d\}$



61 / 61


Chrome File Edit View History Bookmarks Profiles Tab Window Help
10:57
62 / 62

Hallu

$x_i \in \mathbb{R}^d \xrightarrow{\text{tSNE}} y_i \in \mathbb{R}^{2=d'}$

↗ NOT-labels

the low-dimensional map q_{ij} are given by

$$q_{ij} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k \neq l} \exp\left(-\|y_k - y_l\|^2\right)}, \tag{3}$$

The obvious way to define the pairwise similarities in the high-dimensional space is

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

but this causes problems when a high-dimensional datapoint x_i is an outlier (i.e., all pairwise distances $\|x_i - x_j\|^2$ are large for x_i). For such an outlier, the values of p_{ij} are extremely small for all j, so the location of its low-dimensional map point y_i has very little effect on the cost function. As a result, the position of the map point is not well determined by the positions of the other map points. We circumvent this problem by defining the joint probabilities p_{ij} in the high-dimensional space to be the symmetrized conditional probabilities, that is, we set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$. This ensures that $\sum_j p_{ij} > \frac{1}{2n}$ for all x_i makes a significant contri-

the low-dimensional map q_{ij} are given by

$$q_{ij} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k \neq l} \exp\left(-\|y_k - y_l\|^2\right)}, \tag{3}$$

The obvious way to define the pairwise similarities in the high-dimensional space p_{ij} is

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

but this causes problems when a high-dimensional datapoint x_i is an outlier (i.e., all pairwise distances $\|x_i - x_j\|^2$ are large for x_i). For such an outlier, the values of p_{ij} are extremely small for all j, so the location of its low-dimensional map point y_i has very little effect on the cost function. As a result, the position of the map point is not well determined by the positions of the other map points. We circumvent this problem by defining the joint probabilities p_{ij} in the high-dimensional space to be the symmetrized conditional probabilities, that is, we set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$. This ensures that $\sum_j p_{ij} > \frac{1}{2n}$ for all x_i makes a significant contri-

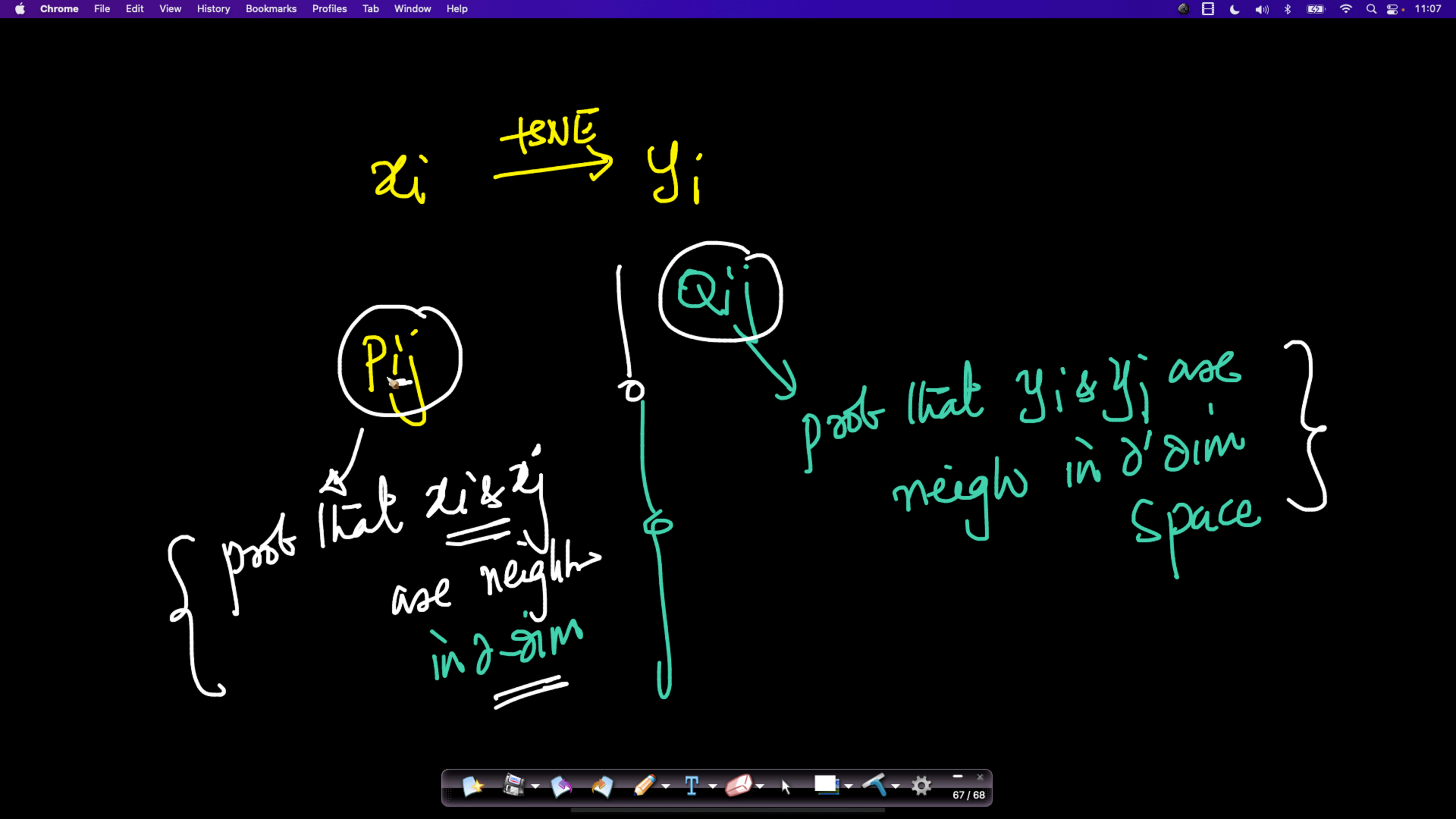
the low-dimensional map q_{ij} are given by

$$q_{ij} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k \neq l} \exp\left(-\|y_k - y_l\|^2\right)}, \tag{3}$$

The obvious way to define the pairwise similarities in the high-dimensional space p_{ij} is

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

but this causes problems when a high-dimensional datapoint x_i is an outlier (i.e., all pairwise distances $\|x_i - x_j\|^2$ are large for x_i). For such an outlier, the values of p_{ij} are extremely small for all j, so the location of its low-dimensional map point y_i has very little effect on the cost function. As a result, the position of the map point is not well determined by the positions of the other map points. We circumvent this problem by defining the joint probabilities p_{ij} in the high-dimensional space to be the symmetrized conditional probabilities, that is, we set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$. This ensures that $\sum_j p_{ij} > \frac{1}{2n}$ for all x_i makes a significant contri-


Chrome File Edit View History Bookmarks Profiles Tab Window Help
11:07
x_i tsNE y_i
P_{ij}
Q_{ij}
prob that x_i & x_j are neighs in d-dim
prob that y_i & y_j are neigh in d' dim space
67 / 68

$\forall i,j \quad P_{ij} \approx Q_{ij}$

P, Q

KL_div

→ differentiable

→ measure similarity b/w dists

P_{ij} Q_{ij}

KL-div (P, Q) ⟶ relative Entropy ⟶ comm. theory

$$= \sum_i \sum_j P_{ij} \log\left(\frac{P_{ij}}{Q_{ij}}\right)$$

(1) ✓ if $P_{ij} \approx Q_{ij}$ $\quad \frac{P_{ij}}{Q_{ij}} \approx 1 \Rightarrow P_{ij} \log \frac{P_{ij}}{Q_{ij}} \to 0$

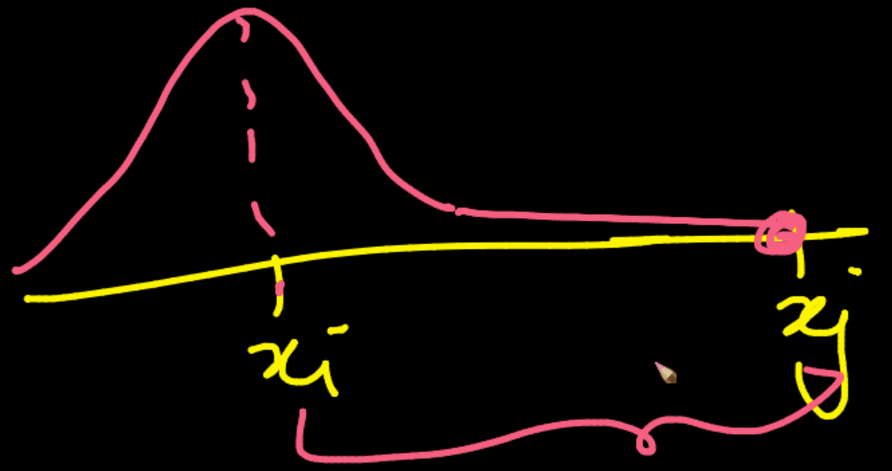
$P \approx Q$ then KL-div$(P, Q) \to 0$

(2) p_{ij} is V-small & $\frac{p_{ij}}{\theta_{ij}} \not\approx 1$

Weightage

$p_{ij} \log \frac{p_{ij}}{\theta_{ij}}$ Small

Prob that x_i & x_j are Neighs in ∂-dim



min y_i's

KL(p||q)

p_{ij}

q_{ij} → y_i's

z_i's

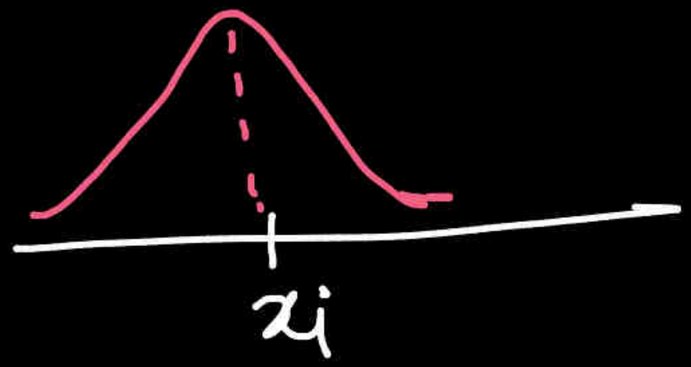
easy to Prove:

KL(p||q) ≥ 0

t-SNE

P_{ij} : Gaussian - fnction

P_{ij} : t-dist with dof = 1



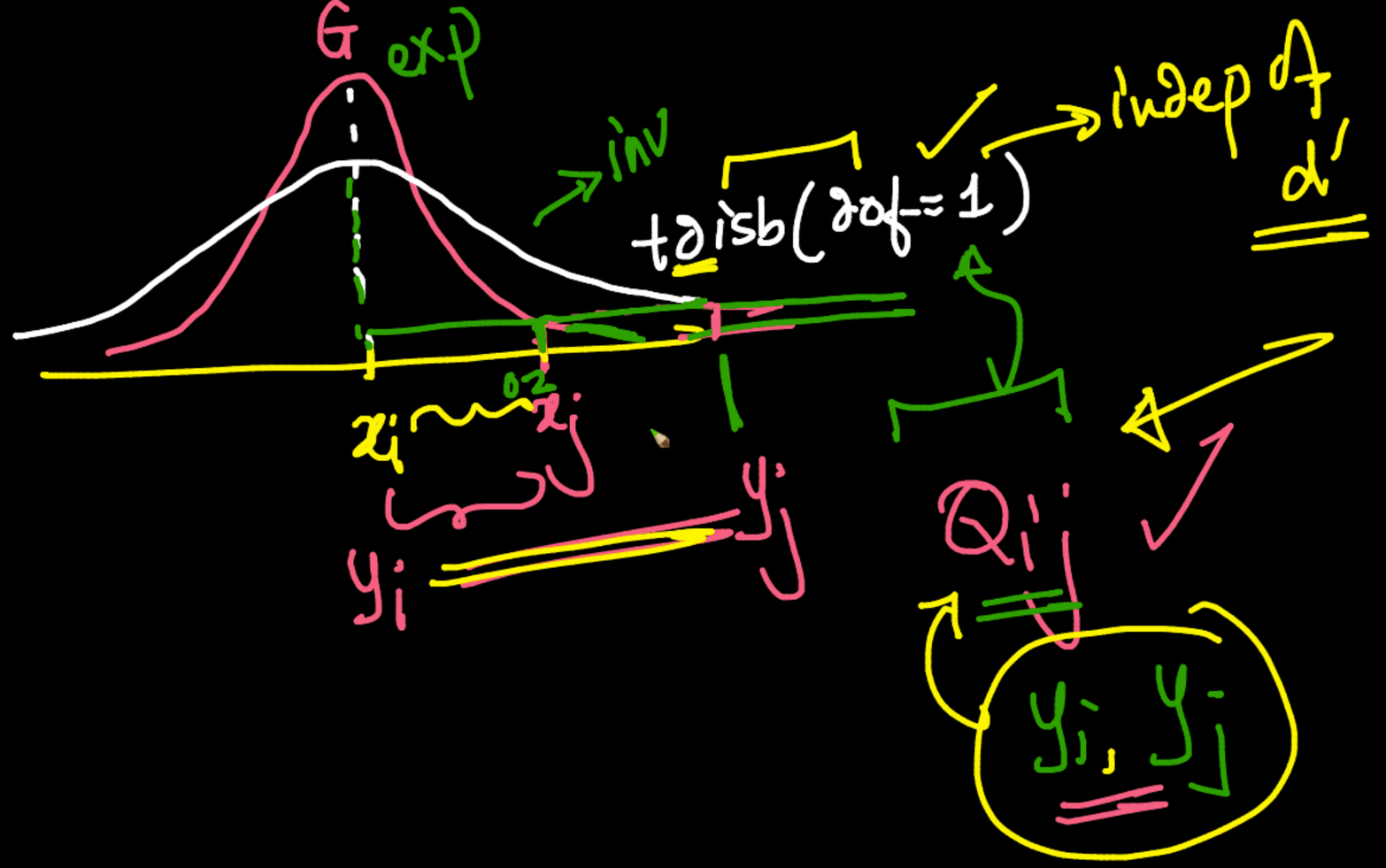
x_i, x_j

Crowding

Pij

x_i x_j

Gaussian



convert distances into probabilities using a Gaussian distribution. In the low-dimensional map, we can use a probability distribution that has much heavier tails than a Gaussian to convert distances into probabilities. This allows a moderate distance in the high-dimensional space to be faithfully modeled by a much larger distance in the map and, as a result, it eliminates the unwanted attractive forces between map points that represent moderately dissimilar datapoints.

In t-SNE, we employ a Student t-distribution with one degree of freedom (which is the same as a Cauchy distribution) as the heavy-tailed distribution in the low-dimensional map. Using this distribution, the joint probabilities q_{ij} are defined as

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}. \tag{4}$$

We use a Student t-distribution with a single degree of freedom, because it has the particularly nice property that $\left(1 + \|y_i - y_j\|^2\right)^{-1}$ approaches an inverse square law for large pairwise distances $\|y_i - y_j\|$ in the low-dimensional map. This makes the map's representation of joint probabilities (almost) invariant to changes in the scale of the map for map points that are far apart. It also means that large clusters of points that are far apart interact in just the same way as individual points, so the optimization operates in the same way at all but the finest scales. A theoretical justification for our

convert distances into probabilities using a Gaussian distribution. In the low-dimensional map, we can use a probability distribution that has much heavier tails than a Gaussian to convert distances into probabilities. This allows a moderate distance in the high-dimensional space to be faithfully modeled by a much larger distance in the map and, as a result, it eliminates the unwanted attractive forces between map points that represent moderately dissimilar datapoints.

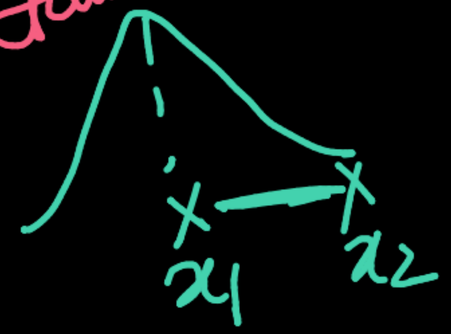
In t-SNE, we employ a Student t-distribution with one degree of freedom (which is the same as a Cauchy distribution) as the heavy-tailed distribution in the low-dimensional map. Using this distribution, the joint probabilities q_{ij} are defined as

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}. \tag{4}$$

We use a Student t-distribution with a single degree of freedom, because it has the particularly nice property that $\left(1 + \|y_i - y_j\|^2\right)^{-1}$ approaches an inverse square law for large pairwise distances $\|y_i - y_j\|$ in the low-dimensional map. This makes the map's representation of joint probabilities (almost) invariant to changes in the scale of the map for map points that are far apart. It also means that large clusters of points that are far apart interact in just the same way as individual points, so the optimization operates in the same way at all but the finest scales. A theoretical justification for our

0-dim

Gaussian

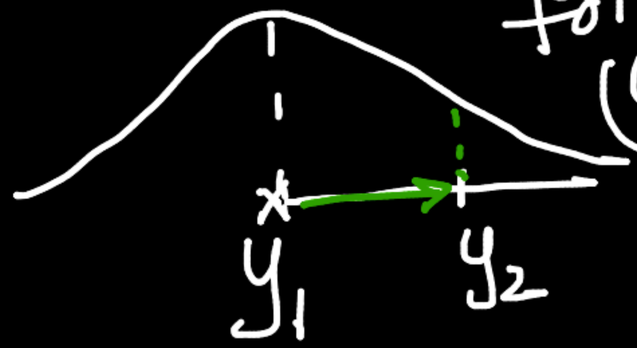


$\gamma_{12} \approx 0.2$

mitigalás

Cauchy

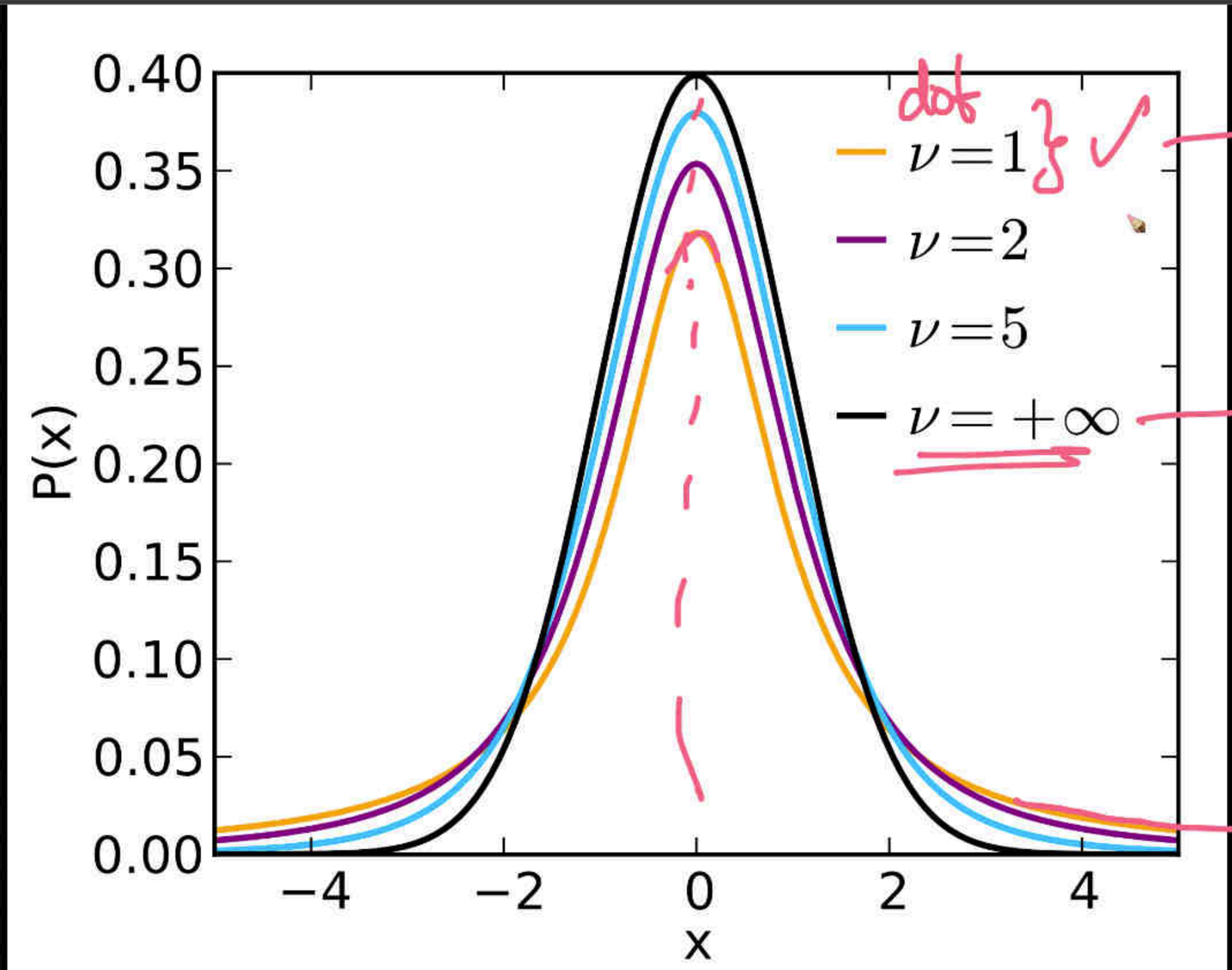
1-dim



for isb (df = 1)
(Cauchy-dist)

$Q_{12} \approx 0.2$

(help me somewhat
overcome finding
Gmse "wiggle" room



Plot of the density function for several members of the Student t family.

convert distances into probabilities using a Gaussian distribution. In the low-dimensional map, we can use a probability distribution that has much heavier tails than a Gaussian to convert distances into probabilities. This allows a moderate distance in the high-dimensional space to be faithfully modeled by a much larger distance in the map and, as a result, it eliminates the unwanted attractive forces between map points that represent moderately dissimilar datapoints.

In t-SNE, we employ a Student t-distribution with one degree of freedom (which is the same as a Cauchy distribution) as the heavy-tailed distribution in the low-dimensional map. Using this distribution, the joint probabilities q_{ij} are defined as

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}. \tag{4}$$

We use a Student t-distribution with a single degree of freedom, because it has the particularly nice property that $\left(1 + \|y_i - y_j\|^2\right)^{-1}$ approaches an inverse square law for large pairwise distances $\|y_i - y_j\|$ in the low-dimensional map. This makes the map's representation of joint probabilities (almost) invariant to changes in the scale of the map for map points that are far apart. It also means that large clusters of points that are far apart interact in just the same way as individual points, so the optimization operates in the same way at all but the finest scales. A theoretical justification for our

✓ Perplexity ≈ (intuitively) effective # neighbors whose distances you are preserving

typically: 5-50

optmzn-problem → no single optima

↳ very time taking

slow

(burnes-hut approx)

$$\mathcal{D} = \{x_i\}_{i=1}^{n} \xrightarrow{\text{tSNE}} \mathcal{D}' = \{y_i\}_{i=1}^{n}$$

$x_{new} \rightarrow$ redo $\rightarrow y_{new}$

sklearn.manifold.TSNE

class `sklearn.manifold.`**`TSNE`**(*n_components=2, *, perplexity=30.0, early_exaggeration=12.0, learning_rate='warn', n_iter=1000, n_iter_without_progress=300, min_grad_norm=1e-07, metric='euclidean', metric_params=None, init='warn', verbose=0, random_state=None, method='barnes_hut', angle=0.5, n_jobs=None, square_distances='deprecated'*) [source]

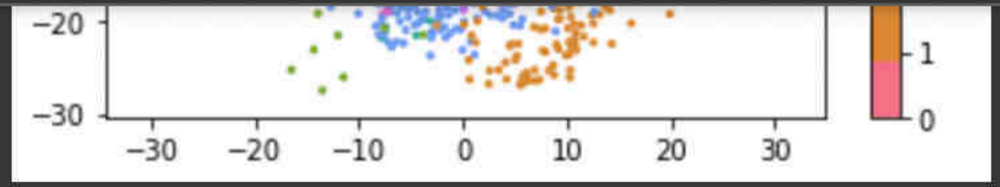
T-distributed Stochastic Neighbor Embedding.

t-SNE [1] is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. t-SNE has a cost function that is not convex, i.e. with different initializations we can get different results.

It is highly recommended to use another dimensionality reduction method (e.g. PCA for dense data or TruncatedSVD for sparse data) to reduce the number of dimensions to a reasonable amount (e.g. 50) if the number of features is very high. This will suppress some noise and speed up the computation of pairwise distances between samples. For more tips see Laurens van der Maaten's FAQ [2].

Read more in the User Guide.

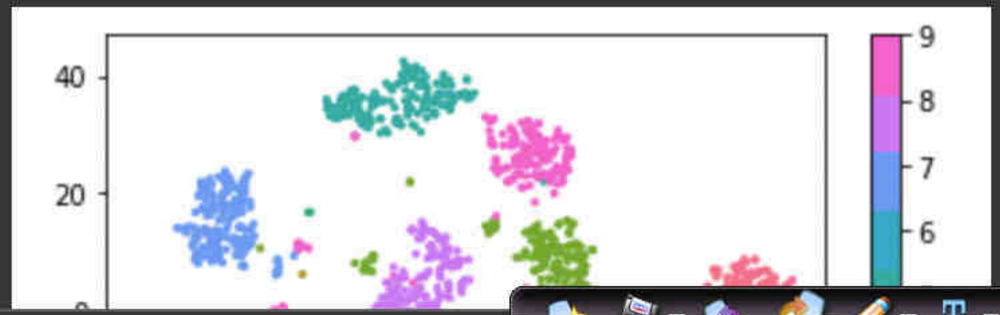
+ Code + Text Reconnect




```
# tSNE
%%time
Z2 = TSNE(n_jobs=-1, initialization='random', random_state=42, negative_gradient_method='bh').fit(X
```

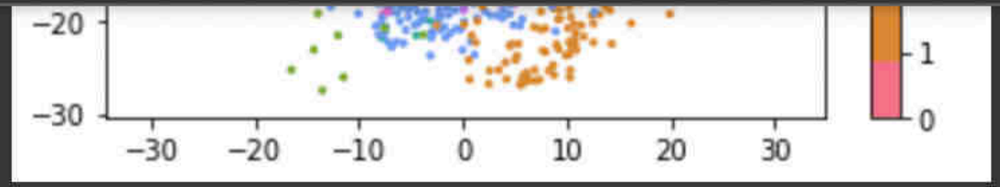
```
CPU times: user 14.6 s, sys: 179 ms, total: 14.8 s
Wall time: 8.32 s
```

```
fig, ax = plt.subplots()
im = ax.scatter(Z2[:,0], Z2[:,1], s=10, c=Y, cmap=cmap, edgecolor='none')
cbar = fig.colorbar(im, ax=ax,label='Digit')
```



82 / 83

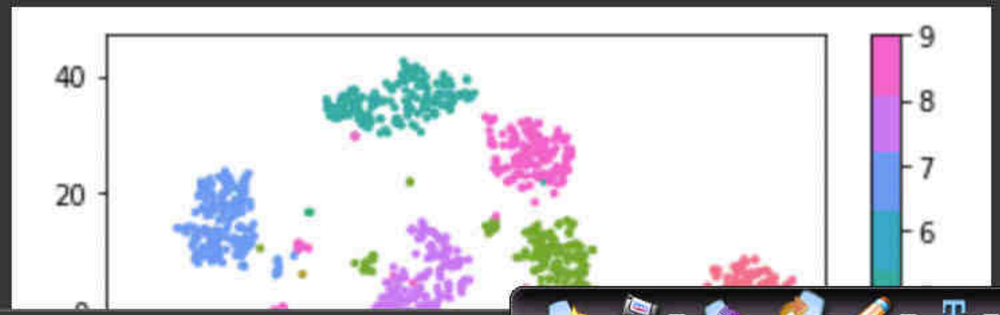
[ ]



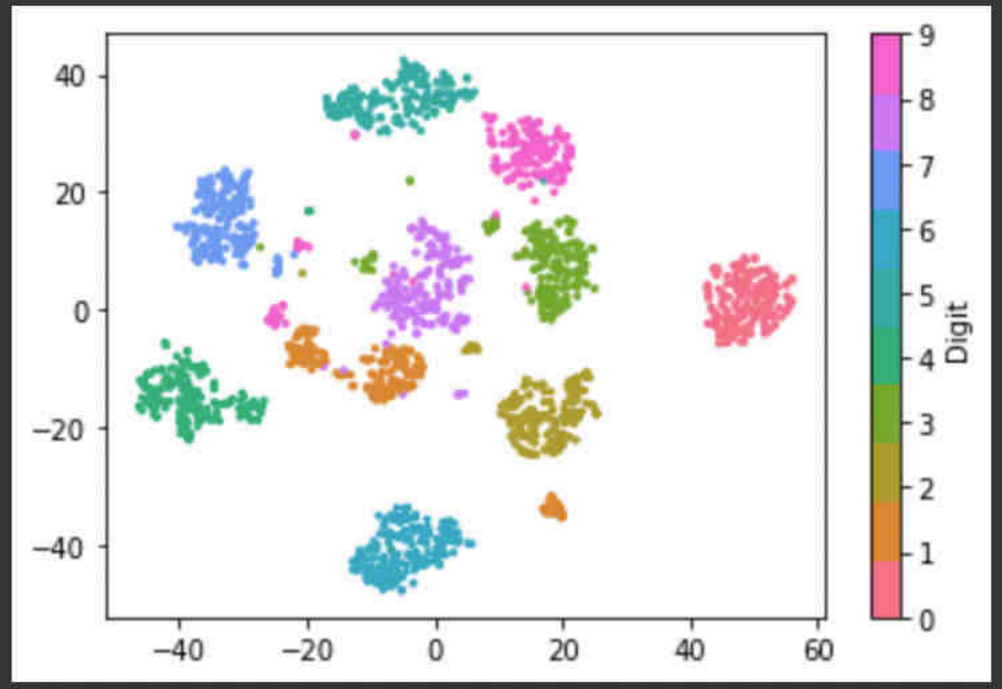

```
# tSNE
%%time
Z2 = TSNE(n_jobs=-1, initialization='random', random_state=42, negative_gradient_method='bh').fit(X
```

```
CPU times: user 14.6 s, sys: 179 ms, total: 14.8 s
Wall time: 8.32 s
```

```
fig, ax = plt.subplots()
im = ax.scatter(Z2[:,0], Z2[:,1], s=10, c=Y, cmap=cmap, edgecolor='none')
cbar = fig.colorbar(im, ax=ax,label='Digit')
```



```
[ ] im = ax.scatter(Z2[:,0], Z2[:,1], s=10, c=Y, cmap=cmap, edgecolor='none')
    cbar = fig.colorbar(im, ax=ax,label='Digit')
```




```
[ ] %%time
    Z3 = UMAP(init='random', random_state=42).fit_transform(X)
```

```
CPU times: user 8.39 s, sys: 30 ms, total: 8.42 s
```

Q

✓ Standardized - data

$w \sim$ Normal - distb

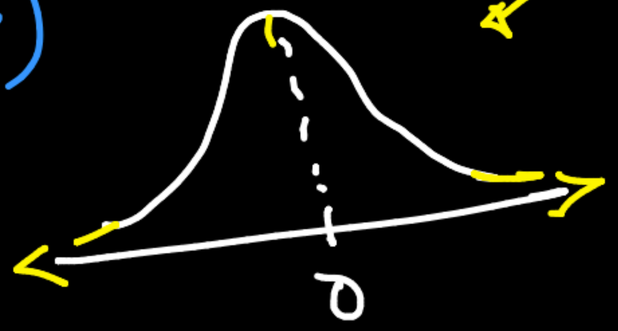
L_2 - reg

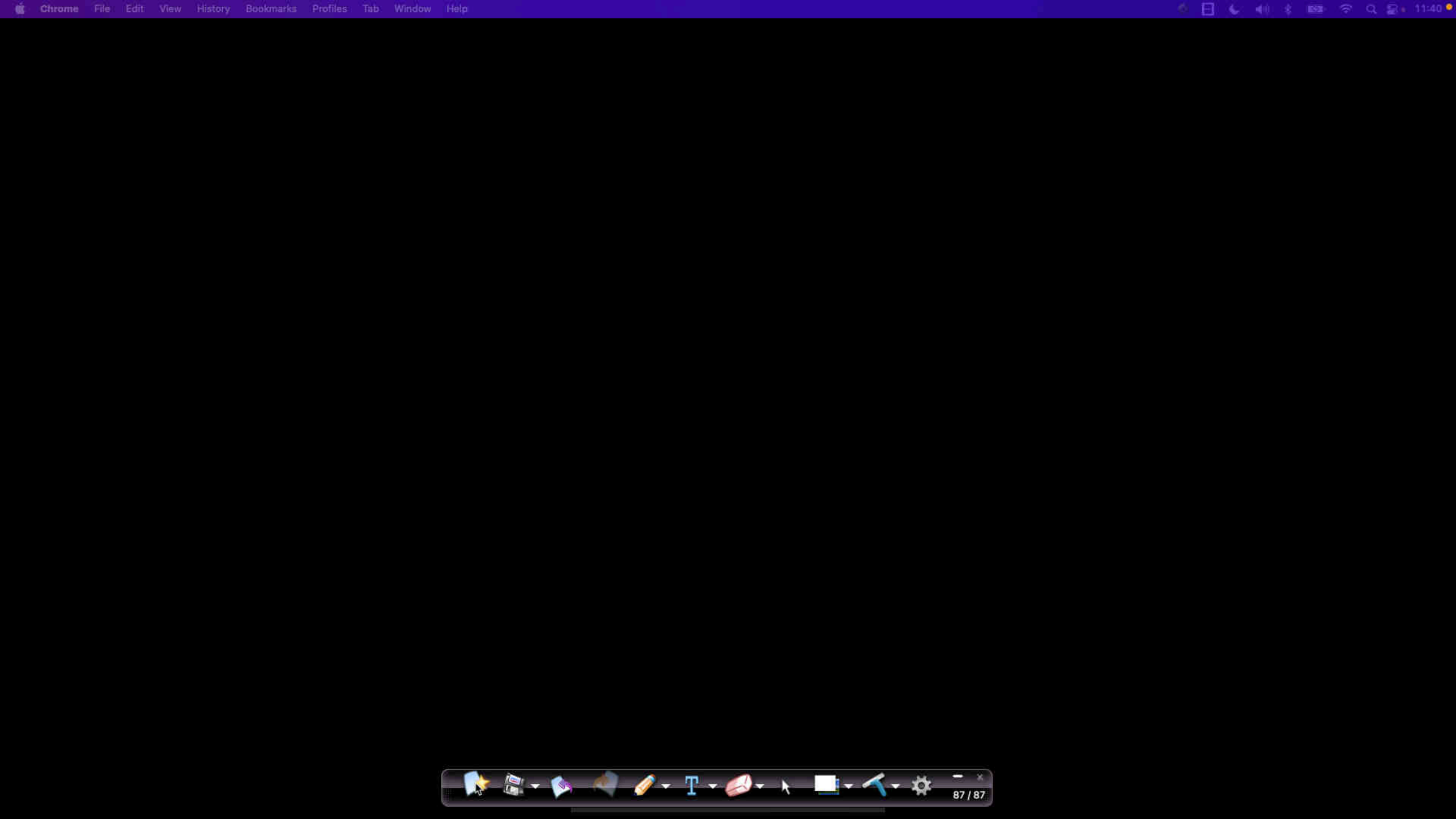
$w \sim$ laplace - distb

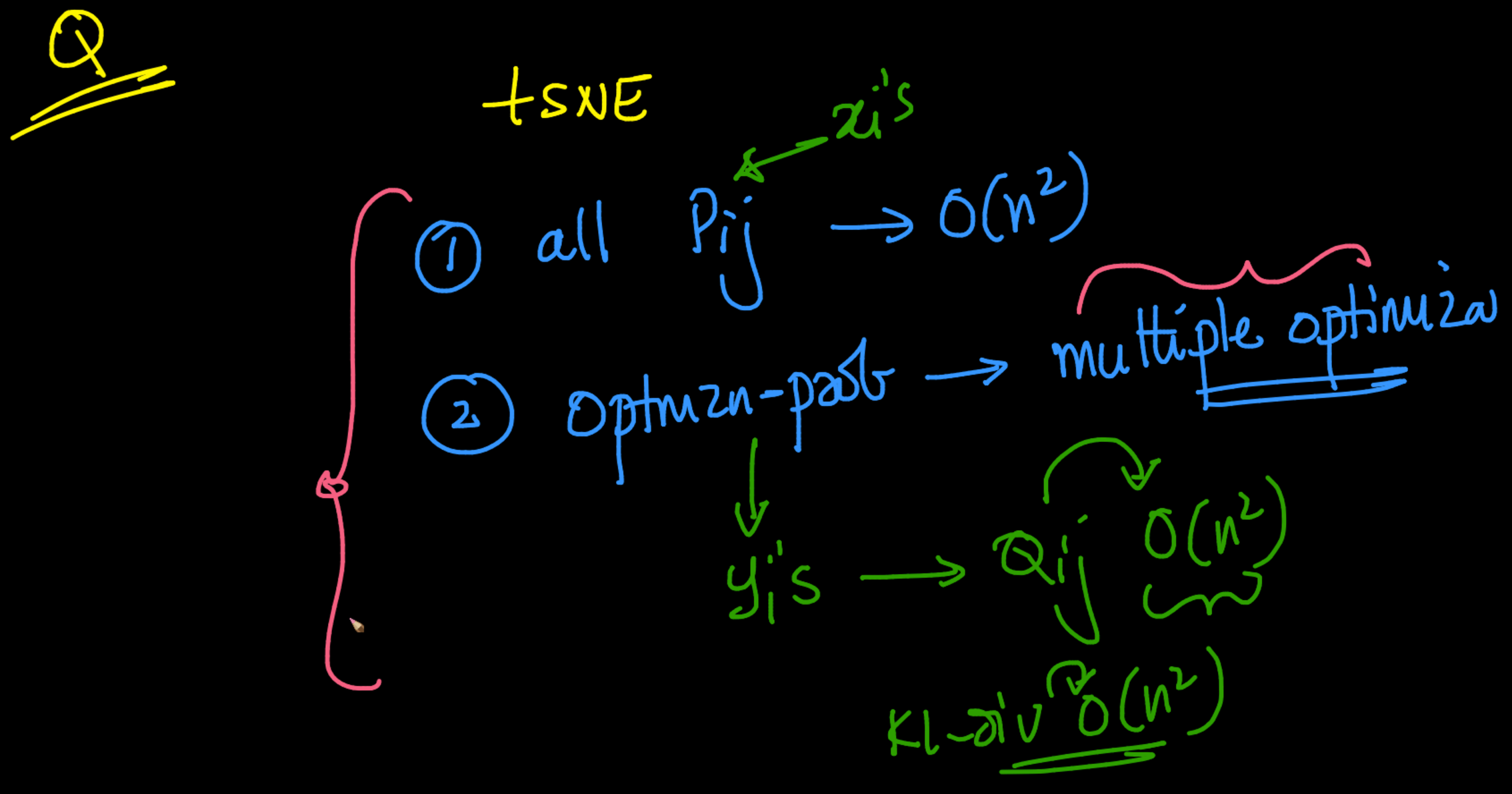
L_1 - reg

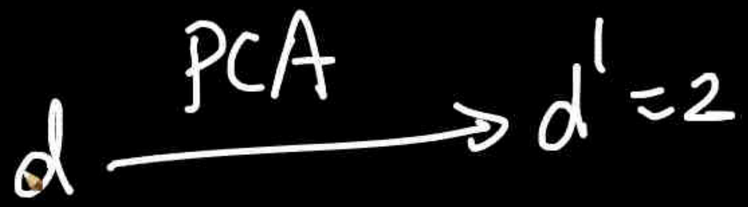
} → prior in MAP

{ $w_j \sim N(0, \sigma)$ → not a very 'strong' assumption




Chrome File Edit View History Bookmarks Profiles Tab Window Help
11:40
87 / 87


tSNE
xi's
1 all Pij → O(n²)
2 optmzn-pass → multiple optimiza
yi's → Qij O(n²)
KL-div O(n²)


d
PCA
d' = 2

$$\text{minimize } \frac{1}{n}\sum_{i=1}\zeta_i + \lambda\|\mathbf{w}\|^2$$

$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1 - \zeta_i \text{ and } \zeta_i \geq 0, \text{ for all } i.$$

This is called the *primal* problem.

$O(n^2)$

Dual [edit]

By solving for the Lagrangian dual of the above problem, one obtains the simplified problem

$$\text{maximize } f(c_1 \ldots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i c_i (\mathbf{x}_i^T \mathbf{x}_j) y_j c_j,$$

$$\text{subject to } \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i.$$

This is called the *dual* problem. Since the dual maximization problem is a quadratic function of the c_i subject to linear constraints, it is efficiently solvable by quadratic programming algorithms.

Here, the variables c_i are defined such that

$$\mathbf{w} = \sum_{i=1}^{n} c_i y_i \mathbf{x}_i.$$

Moreover, $c_i = 0$ exactly when $\mathbf{x}_i$ lies on the correct side of the margin, and $0 < c_i < (2n\lambda)^{-1}$ when $\mathbf{x}_i$ lies on the margin's boundary. It follows that $\mathbf{w}$ can be written as a linear combination of the support vectors.

The offset, b, can be recovered by finding an $\mathbf{x}_i$ on the margin's boundary and solving

$$y_i(\mathbf{w}^T\mathbf{x}_i - b) = 1 \iff b = \mathbf{w}^T\mathbf{x}_i - y_i.$$

(Note that $y_i^{-1} = y_i$ since $y_i = \pm 1$.)

Kernel trick [edit]

Main article: Kernel metho

Suppose now that we would like to learn a nonlinear classification rule which corresponds to a linear classification rule for the

89 / 90

PCA

$d \longrightarrow d'$ } explicitly

RBF - Kernel } implicitly

$d \longrightarrow d'$

$K(x_i, x_j)$

∞-dim

Q

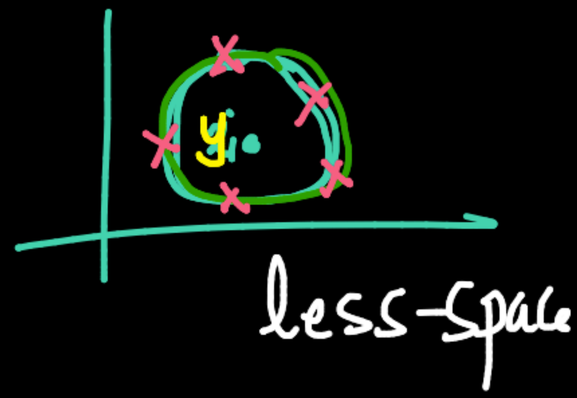
14 dim → 3-dim ✓

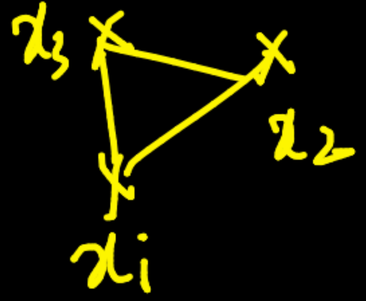
tSNE

14 dim → 2-dim

$x_1, x_2 \ldots x_{30}$

x_i, x_j





perplexity = 30 (default) — num, param

$$\text{minimize } \frac{1}{n}\sum_{i=1}\zeta_i + \lambda\|\mathbf{w}\|^2$$

$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1 - \zeta_i \text{ and } \zeta_i \geq 0, \text{ for all } i.$$

This is called the *primal* problem.

Dual [edit]

By solving for the Lagrangian dual of the above problem, one obtains the simplified problem

$$\text{maximize } f(c_1 \ldots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i c_i (\mathbf{x}_i^T \mathbf{x}_j) y_j c_j,$$

$$\text{subject to } \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i.$$

This is called the *dual* problem. Since the dual maximization problem is a quadratic function of the c_i subject to linear constraints, it is efficiently solvable by quadratic programming algorithms.

Here, the variables c_i are defined such that

$$\mathbf{w} = \sum_{i=1}^{n} c_i y_i \mathbf{x}_i.$$

Moreover, $c_i = 0$ exactly when $\mathbf{x}_i$ lies on the correct side of the margin, and $0 < c_i < (2n\lambda)^{-1}$ when $\mathbf{x}_i$ lies on the margin's boundary. It follows that $\mathbf{w}$ can be written as a linear combination of the support vectors.

The offset, b, can be recovered by finding an $\mathbf{x}_i$ on the margin's boundary and solving

$$y_i(\mathbf{w}^T\mathbf{x}_i - b) = 1 \iff b = \mathbf{w}^T\mathbf{x}_i - y_i.$$

(Note that $y_i^{-1} = y_i$ since $y_i = \pm 1$.)

Kernel trick [edit]

Main article: Kernel metho

Suppose now that we would like to learn a nonlinear classification rule which corresponds to a linear classification rule for the