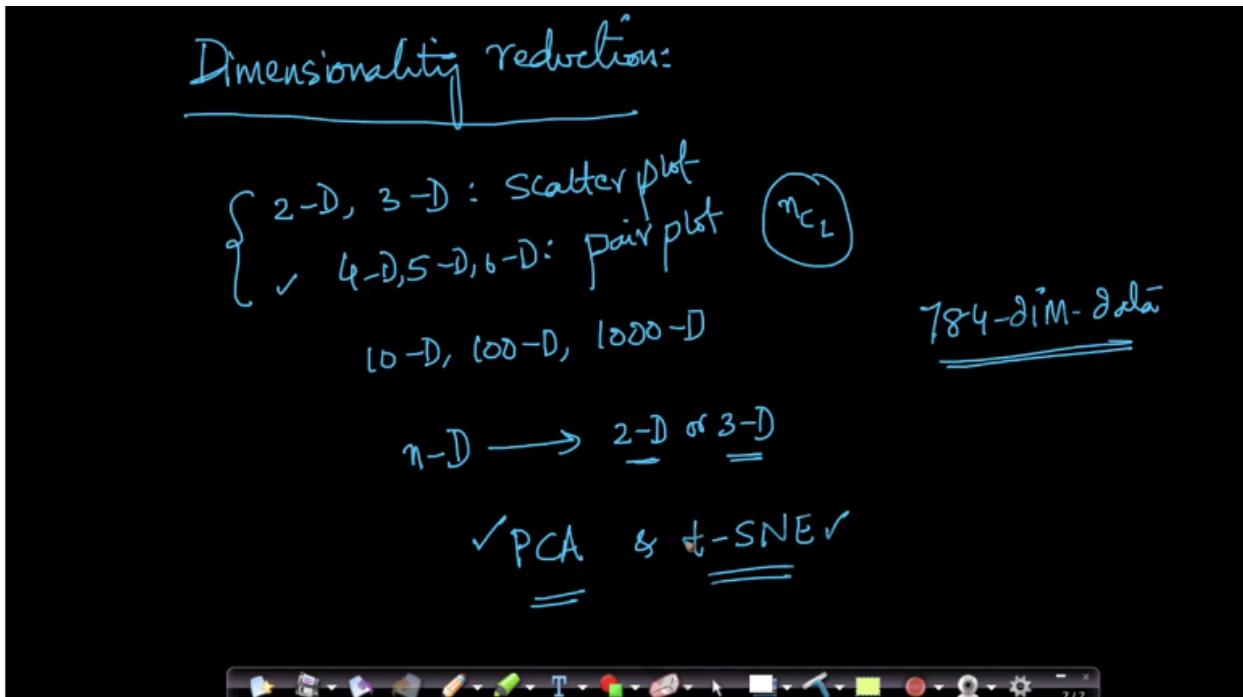


25.1 What is Dimensionality Reduction?



Timestamp: 2:11

We have seen that 2-D and 3-D data can be visualized using scatter plots, but as the number of dimensions increases we need nc_2 number of pair plots, which can be difficult to visualize. Hence we use dimensionality reduction techniques like PCA and t-SNE using which we can reduce the dimensionality of the data from n-D to 2-D and 3-D, which we can visualize easily.

25.2 Row vector and Column vector?

Row-vector & column-vector

flower: $[SL, PL, SW, PW]$]
real-values

\mathbb{R} : real Space

i-th point: $x_i \in \mathbb{R}^d \rightarrow d\text{-dim. columnvector}$

$$x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \\ x_{id} \end{bmatrix}_{d \times 1} : \text{column-vector}$$

$$f_1 = \begin{bmatrix} 2.1 \\ 3.2 \\ 1.6 \\ 4.2 \end{bmatrix}$$

Column-vector

Timestamp: 4:06

An i-th datapoint x_i is often represented either as a column vector or a row vector. If not mentioned explicitly then it is assumed to be a column vector. A column vector looks like as mentioned in the above figure. A d dimensional vector that takes real values is often mentioned as x_i belonging to \mathbb{R}^d where R stands for real-values.

A row vector representation of x_i looks as below.

$$\vec{x}_i = \begin{bmatrix} \downarrow & \downarrow & \downarrow & \downarrow \\ 2 \cdot 1, 3 \cdot 2, 4 \cdot 6, 1 \cdot 2 \end{bmatrix}_{1 \times 4} : \text{row-vector}$$

Timestamp: 3:30

25.3 How to represent a data set?

Dataset :-

$\mathcal{D} = \{x_i, y_i\}_{i=1}^n$

Iris:

$\mathcal{D} = \{x_i, y_i\}_{i=1}^n$

$x_i \in \mathbb{R}^d$; $y_i \in \{setosa, versicolor, virginica\}$

$\begin{bmatrix} SL \\ SW \\ PL \\ PW \end{bmatrix} \quad y_i \in \{setosa, versicolor, virginica\}$

Timestamp: 3:09

There are many ways to represent a dataset but one such way to represent a dataset for the example of iris dataset is as above. A dataset is represented as $D = \{x_i, y_i\}_{i=1}^n$, where x_i represents the datapoint and y_i represent the corresponding label.

x_i is represented as belonging to \mathbb{R}^4 as it contains 4 real valued features and y_i can take any value from $\{setosa, versicolor, virginica\}$ the three classes or labels in the iris dataset.

25.4 How to represent a data set as a Matrix?

Dataset as a data-matrix:

$$X = \begin{bmatrix} f_1 & f_2 & f_3 & \dots & f_j & \dots & f_d \end{bmatrix}$$

each datapoint: row
each column: feature

$\mathcal{D} = \{x_i, y_i\}_{i=1}^n$

$x_i \in \mathbb{R}^d$
 $y_i \in \{s, v_i, v_e\}$

$\rightarrow d\text{-features}$
 $\rightarrow \text{column-vector}$

$\begin{cases} x_i \text{ is a col-vector} \\ x_i^T \text{ is a row-vector} \end{cases}$

Timestamp: 2:09

There are two ways to represent our data matrix X , the first way is to represent it in such a way that each row represents a datapoint and each column represents a feature. Notice that since x_i is a column vector, in order for us to represent it as a row vector, we are using x_i^T .

$$X = \begin{bmatrix} f_1 & f_2 & f_3 & \dots & f_i & \dots & f_n \\ x_1 & x_2 & x_3 & \dots & x_i & \dots & x_n \\ \vdots & \vdots & \vdots & & \vdots & & \vdots \\ x_d & x_d & x_d & \dots & x_d & \dots & x_d \end{bmatrix}_{d \times n}$$

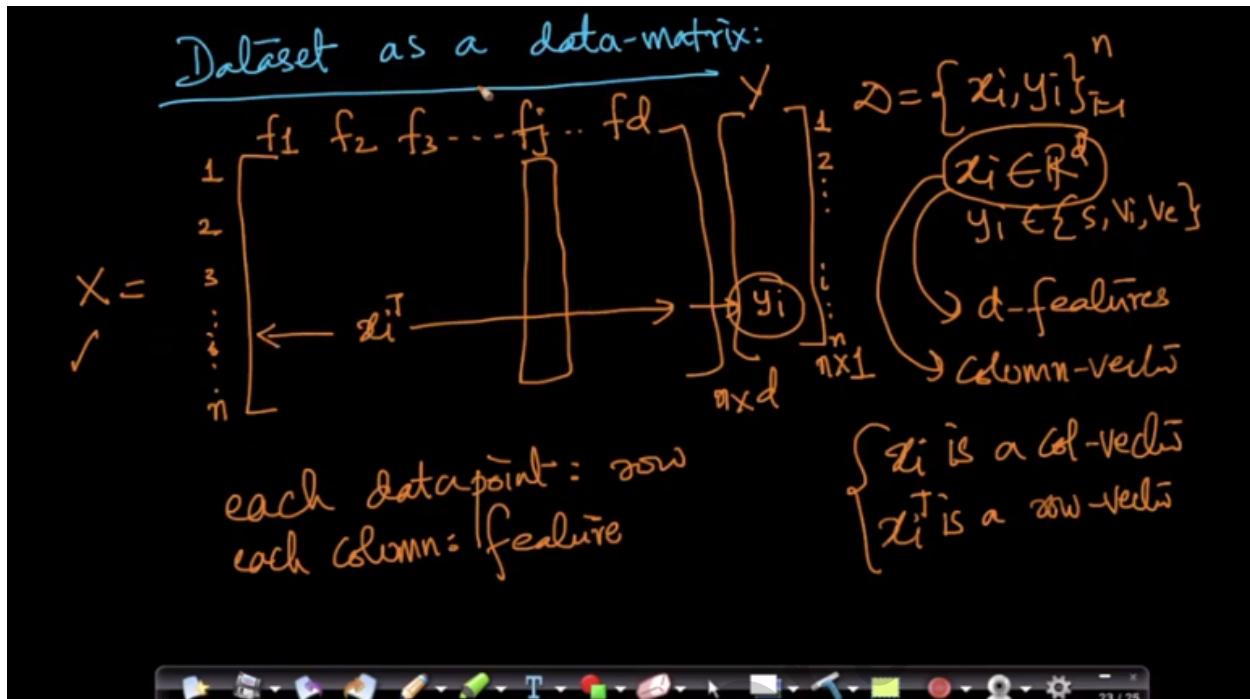
column: data-point
row: feature/variable

$f_1 = PL$
 $f_2 = PW$
 $f_3 = SL$
 $f_4 = SW$

Timestamp: 3:56

The other way is to just represent it as above, where each column represents a data point and each row represents a feature. This data matrix X is just a transpose of the data matrix X in the first approach.

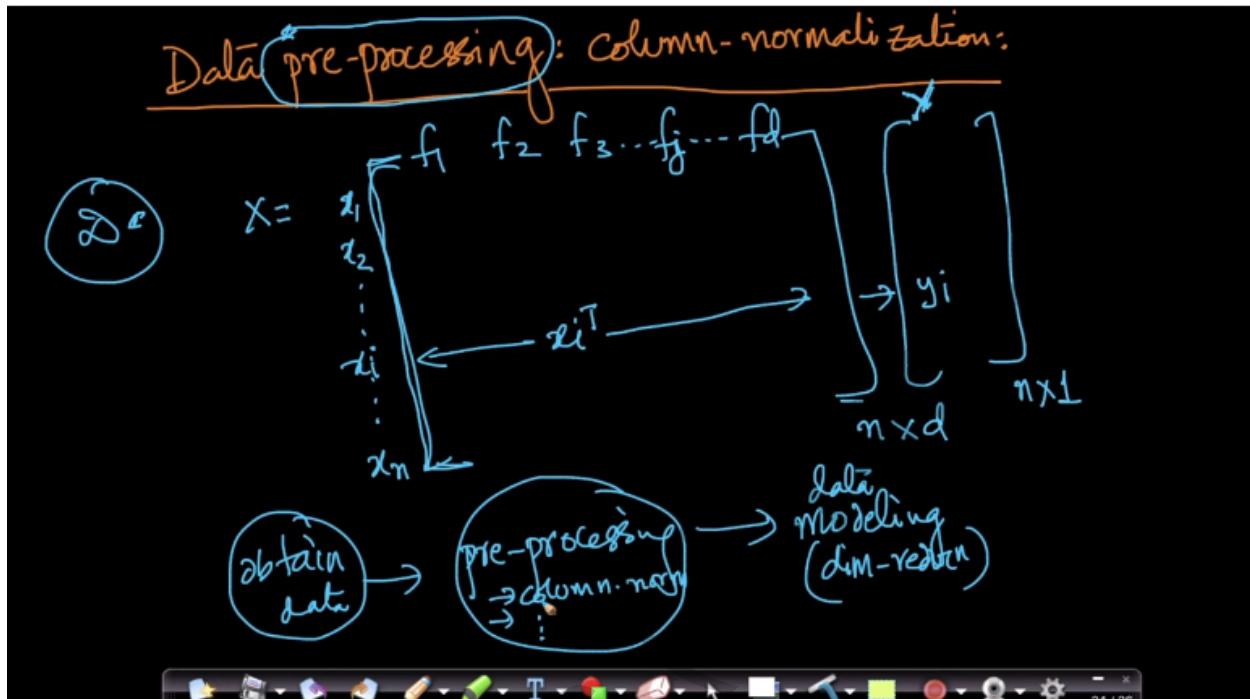
The most common approach is the first one, where each row represents a datapoint and each column represents a feature. The entire dataset now looks like below using the first approach.



Timestamp: 6:35

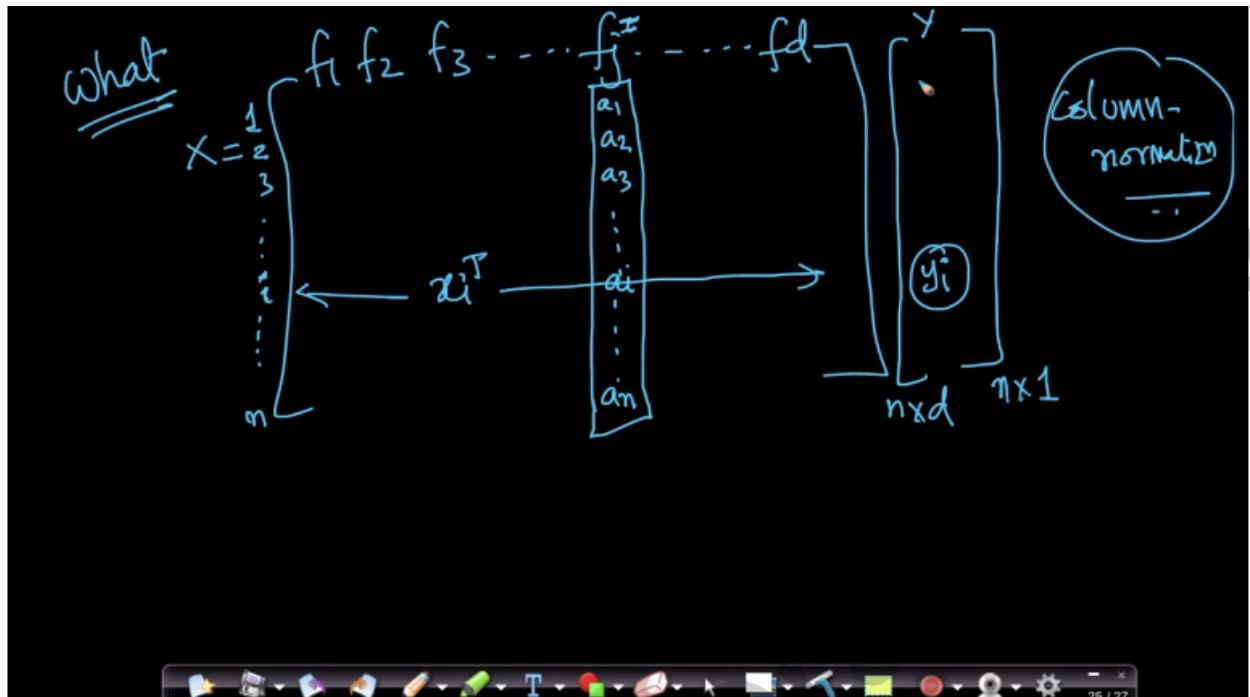
Notice that Y is represented as a column matrix where each row y_i represents the label for the corresponding datapoint x_i in X .

25.5 Data Preprocessing: Feature Normalization



Timestamp: 3:10

After obtaining the data, we need to preprocessing of the data such that our dimensionality reduction algorithms can work better. One such preprocessing is column normalization.



Timestamp: 5:26

For column normalization ,we pick each feature f_j , lets say it contains numbers a_1, a_2, \dots, a_n as above then we do column normalization as follows.

column: $a_1, a_2, \dots, a_i, \dots, a_n$ $\rightarrow n$ -values of f_j

$$\max(a_i) = a_{\max} \geq a_i \quad (i:1 \rightarrow n)$$

$$\min(a_i) = a_{\min} \leq a_i \quad (i:1 \rightarrow n)$$

$$a'_i = \frac{a_i - a_{\min}}{a_{\max} - a_{\min}} \quad a'_i \in [0, 1]$$

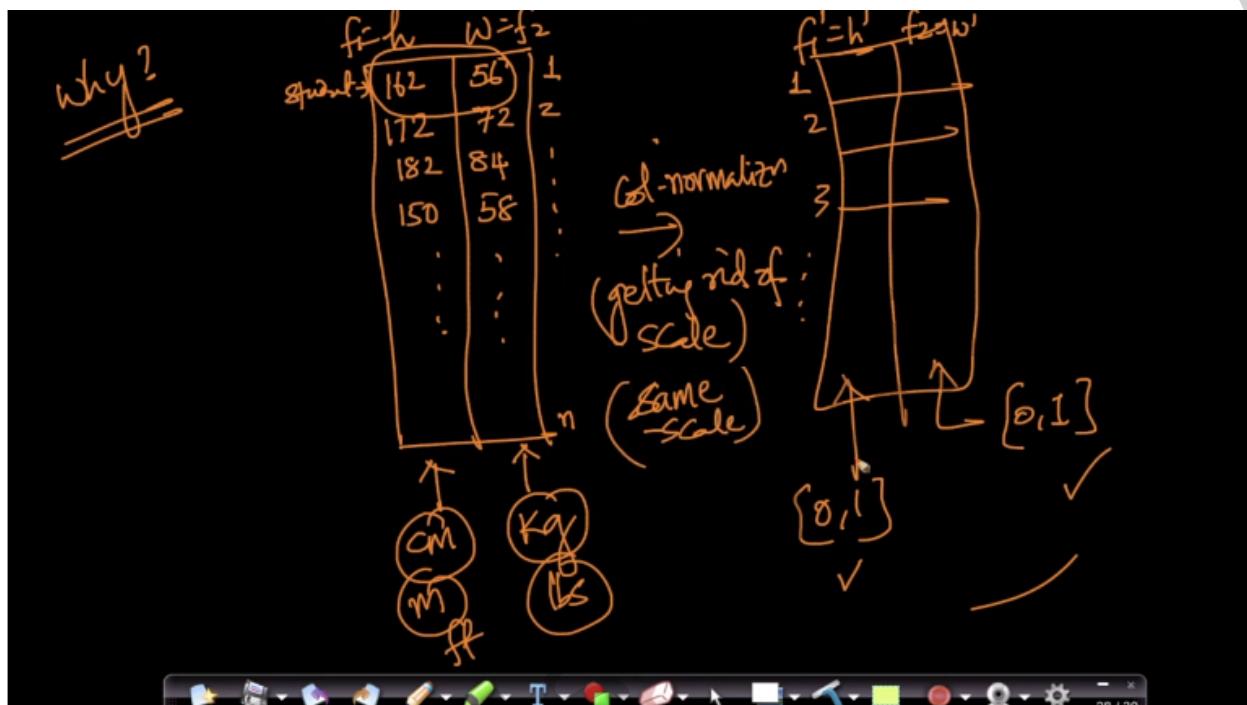
$$a'_{\max} = \frac{a_{\max} - a_{\min}}{a_{\max} - a_{\min}} = 1$$

$$a'_{\min} = \frac{a_{\min} - a_{\min}}{a_{\max} - a_{\min}} = 0$$

Timestamp: 9:30

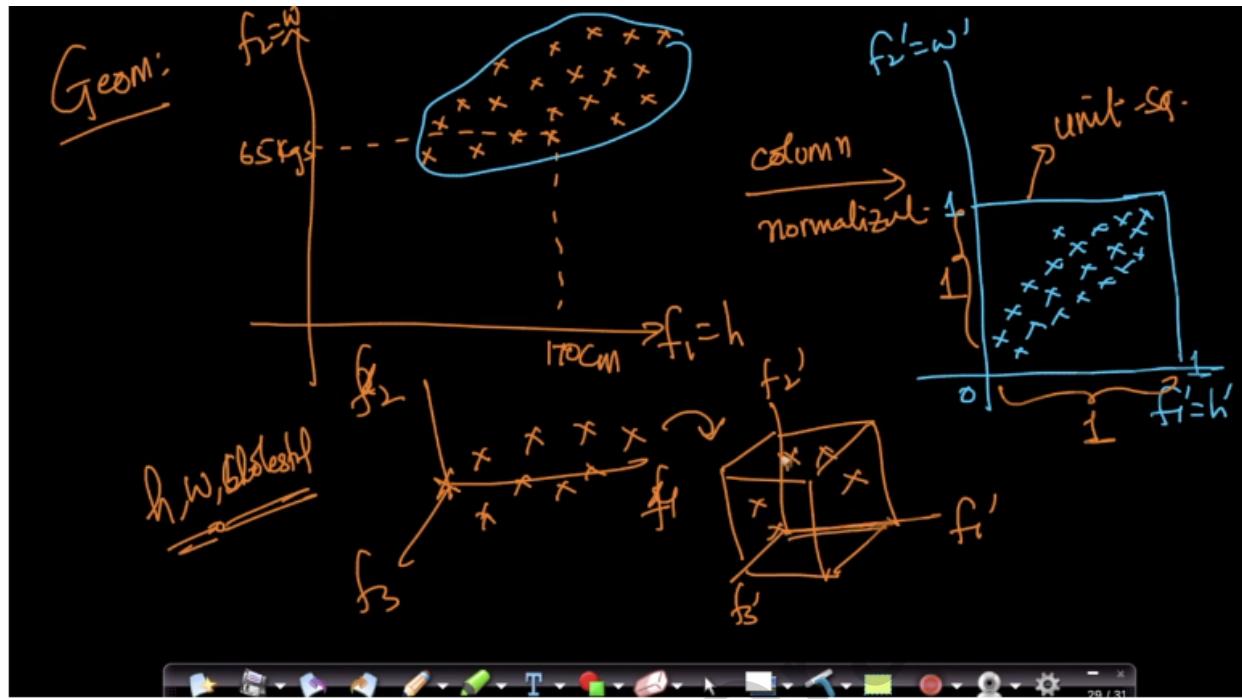
As shown in the above figure, with column normalization what we are doing is, from a_1, a_2, \dots, a_n we are getting a'_1, a'_2, \dots, a'_n as below, such that each a'_i now belongs to $[0,1]$

$$a'_i = (a_i - a_{\min}) / (a_{\max} - a_{\min})$$



Timestamp: 14:32

Since features like height, weight etc can be measured in different scales such as cm,m,etc, we are doing column normalization to get rid of the problems that come up with the scale(which will be discussed later) and we are converting all the features into the same scale using column normalization.



Timestamp: 17:45

Geometrically, through column normalization, we are squishing data points that are spread over the entire space into a unit-square when it is 2-D data or into unit-cube when it is 3-D data and n-D unit hyper cube when the data is of n-D data.

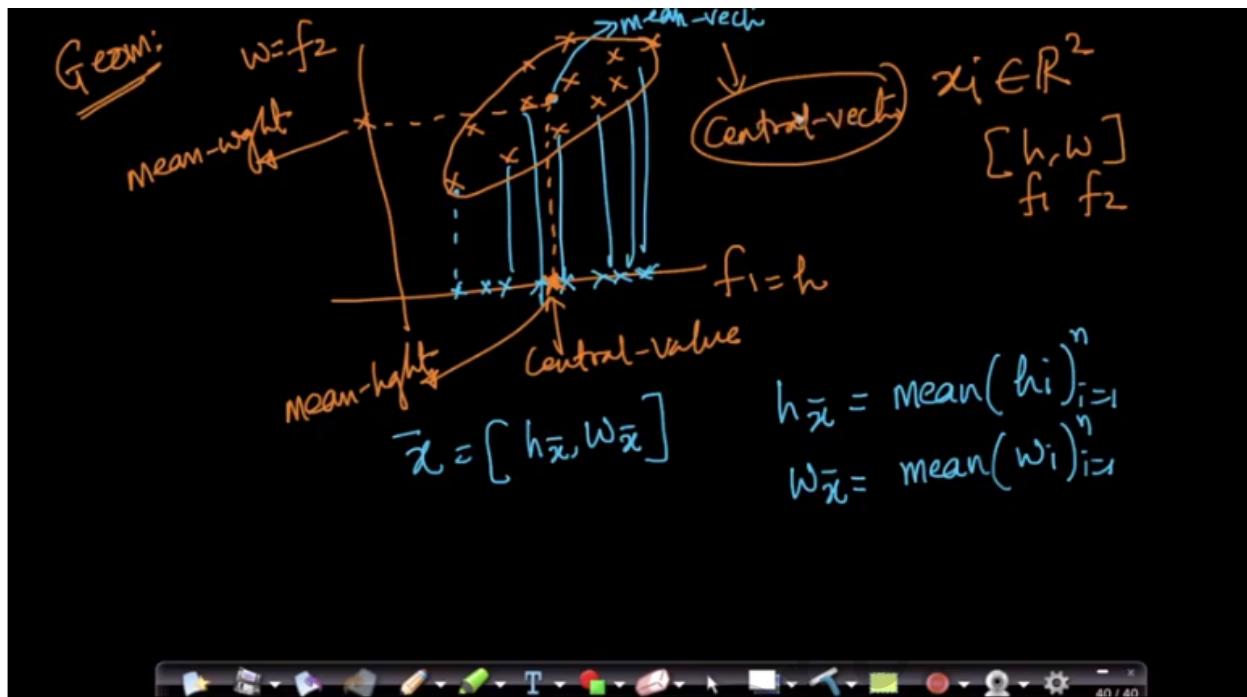
25.6 Mean of a Data Matrix

$$\begin{aligned}
 x_1 &= \begin{bmatrix} f_1 \\ 2 \cdot 2 \\ 4 \cdot 2 \end{bmatrix} \in \mathbb{R}^3 \\
 x_2 &= \begin{bmatrix} 1 \cdot 2 \\ 3 \cdot 2 \end{bmatrix} \in \mathbb{R}^2 \\
 \underline{x_1 + x_2} &= \begin{bmatrix} 3 \cdot 4 \\ 7 \cdot 4 \end{bmatrix} \\
 \bar{x} \in \mathbb{R}^d & \\
 \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i & = \frac{1}{n} (x_1 + x_2 + \dots + x_n) \\
 \text{Mean vector} &
 \end{aligned}$$

Timestamp: 2:31

As shown in the above picture, the sum of two vectors x_1 and x_2 is just a vector where each element is the sum of corresponding elements in the vectors x_1 and x_2 .

The mean vector of n data points, is the sum of n datapoints divided by n as shown above.



Timestamp: 5:47

Geometrically, mean vector is like a central value of the data.

25.7 Data Preprocessing: Column Standardization

Data-preprocessing: Column-standardization

✓ Column-normalization :- $[0, 1]$ \leftarrow get rid of scales of each feature
 \hookrightarrow unit hyper-cube
Col. Std :- more often used in practice

Timestamp: 1:11

Column standardization is a similar preprocessing technique similar to column normalization and it is used more often in practice.

$$X = \begin{bmatrix} f_1 & f_2 & \dots & f_j & \dots & f_d \end{bmatrix}$$

X is an $n \times d$ matrix where $a_i \in [0, 1]$.

The values of f_j are $a_1, a_2, a_3, \dots, a_n$.

Column standardization (Col. Std) is performed to get rid of scales of each feature.

Mean $\{a_i\}_{i=1}^n = 0$

Std-dev $\{a_i\}_{i=1}^n = 1$

Timestamp: 4:26

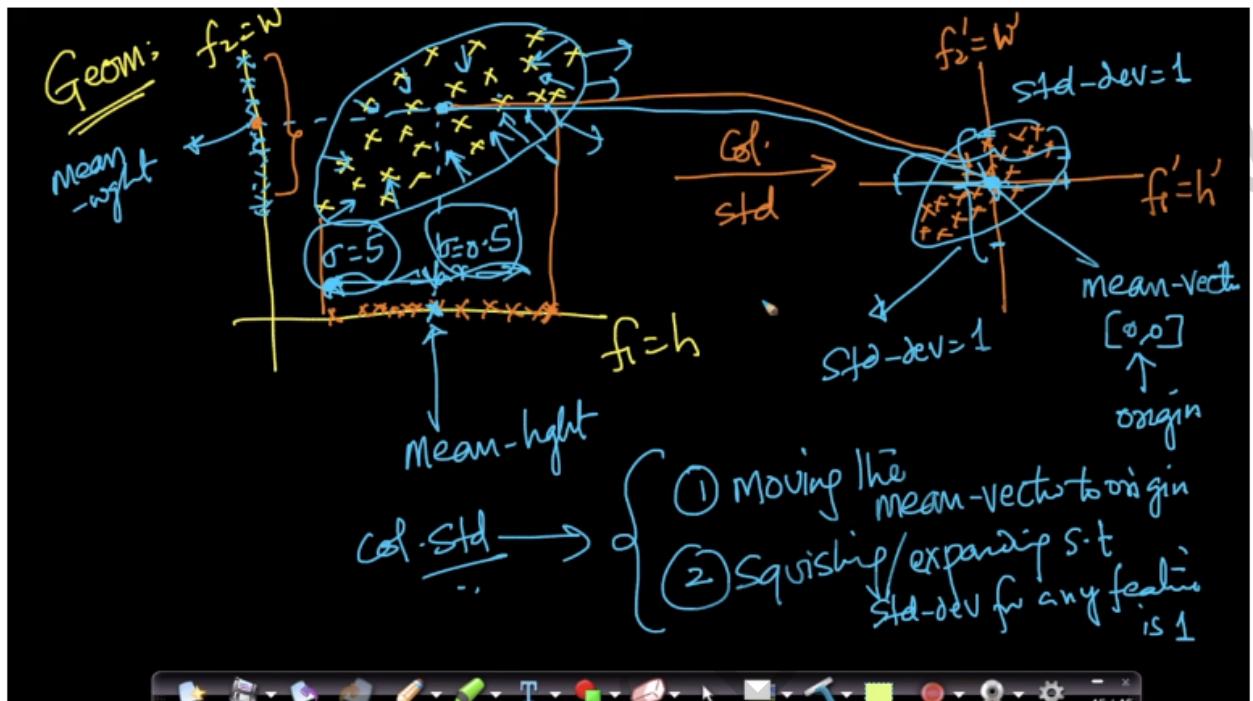
Given a data matrix X, for each feature containing n values a_1, a_2, \dots, a_n , we convert them into a'_1, a'_2, \dots, a'_n such that the mean of a_i 's is 0 and std of a_i 's is 1.

We do it, by getting a'_i from a_i as below

The diagram illustrates the process of standardizing data points a_1, a_2, \dots, a_n . It starts with a box labeled "any - dist" containing the data points. An arrow labeled "Std" points down to the data points, which are then transformed into a'_1, a'_2, \dots, a'_n . This transformation is shown with two arrows: one pointing to the right labeled "mean = 0" and another pointing down labeled "std-dev = 1". Below this, the sample mean $\bar{a} = \text{Mean}\{a_i\}_{i=1}^n$ and sample std-dev $s = \text{std-dev}\{a_i\}_{i=1}^n$ are defined. A formula for the standardized value $a'_i = \frac{a_i - \bar{a}}{s}$ is shown in a circle, with an example calculation: $\text{Mean}\{a'_i\}_{i=1}^n = 0$ and $\text{std-dev}\{a'_i\}_{i=1}^n = 1$.

Timestamp: 6:57

We subtract the mean of a_i and divide by the standard deviation of a_i 's from each a_i to get a'_i .



Timestamp: 14:01

Geometrically, with column standardization we are moving the mean vector of our data to origin and squishing or expanding our data such that the standard deviation for any feature is 1.

Col. Standardizat :- Mean - centering \rightarrow origin
+ scaling \rightarrow $Std - dev = 1$
for all features

Timestamp: 14:50

Basically, column standardization is considered as mean centering (such that mean vector is origin) plus scaling (such that standard deviation is 1 for each feature).

25.8 Co-variance of a Data Matrix

Co-variance matrix

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{def: } S = \begin{bmatrix} s_{11} & s_{12} & \dots & s_{1d} \\ s_{21} & s_{22} & \dots & s_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ s_{d1} & s_{d2} & \dots & s_{dd} \end{bmatrix}$$

f_j = col-vedot jth feature
 s_{ij} = ith row by jth col.
 x_{ij} = jth feature for ith data point

square-matrix $d \times d$

Timestamp: 4:00

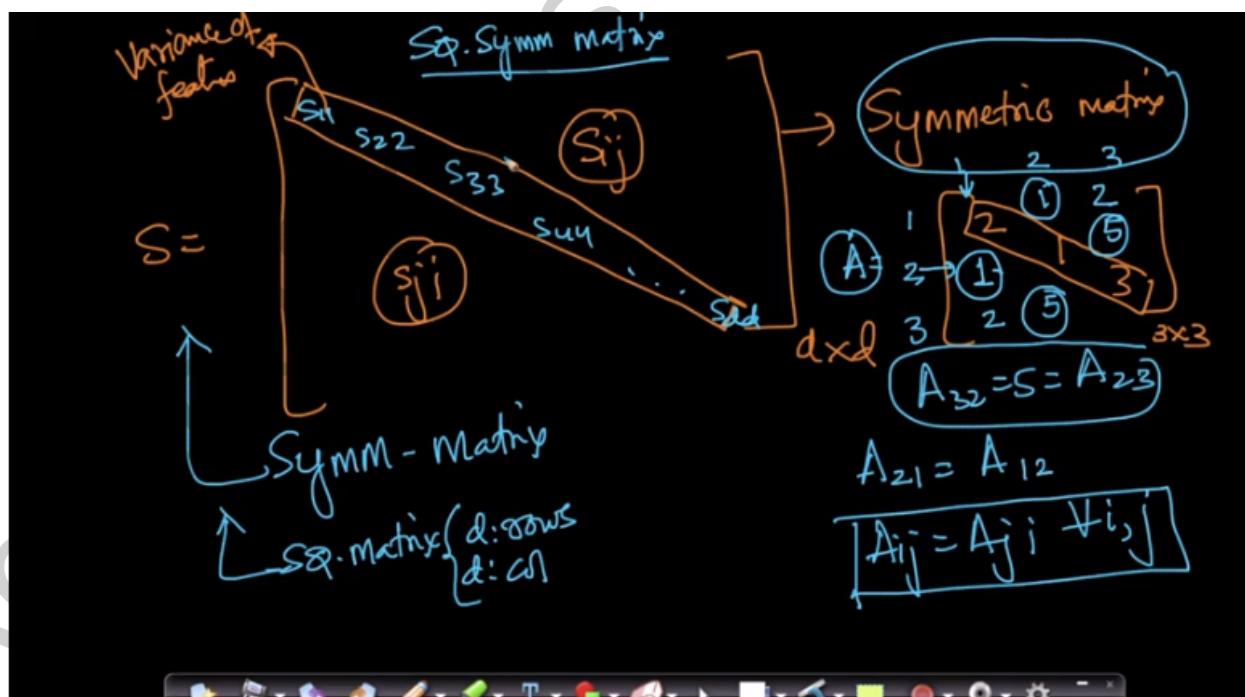
We define the covariance matrix of our data S as a square matrix of $d \times d$ shape, where d is the number of features of our data. Each element s_{ij} which is the element of i th row and j th column is defined as the covariance of i th feature and j th feature as below.

$$S_{ij} = \text{cov}(f_i, f_j) = \text{cov}(f_j, f_i) = S_{ji}$$

$$\begin{aligned} \text{Cov}(f_i, f_j) &= \cancel{\text{Var}}(f_i) \\ \left\{ \begin{array}{l} \checkmark \text{Cov}(x, x) = \text{Var}(x) - ① \\ \checkmark \text{Cov}(f_i, f_j) = \text{Cov}(f_j, f_i) - ② \end{array} \right. \end{aligned}$$

Timestamp: 9:27

Notice that due to how covariance is defined, $\text{cov}(f_i, f_j) = \text{cov}(f_j, f_i)$ hence $s_{ij} = s_{ji}$.



Timestamp: 10:15

As shown above, the covariance matrix S is a square symmetric matrix, as it satisfies all the rules of a square matrix where the number of rows must be equal to the number of columns and that of a symmetric matrix where $S_{ij}=S_{ji}$.

$$X = \frac{1}{n} \begin{bmatrix} f_1 & f_2 & \dots & f_d \\ x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix}$$

Let \textcircled{X} col-standardized \Rightarrow $\text{mean}\{f_i\} = 0$
 $\text{std-dev}\{f_i\} = 1$

$$\text{Cov}(f_1, f_2) = \frac{1}{n} \sum_{i=1}^n (x_{i1} - \text{mean}(f_1)) \cdot (x_{i2} - \text{mean}(f_2))$$

Timestamp: 14:47

If our data matrix X is column standardized then the mean of each feature is 0 and the standard deviation of each feature is 1, hence we are left with $\text{cov}(f_1, f_2)$ as below

$$\text{Cov}(f_1, f_2) = \frac{1}{n} \sum_{i=1}^n x_{i1} * x_{i2}$$

$$\text{Cov}(f_1, f_2) = (f_1^T f_2) * \frac{1}{n}$$

Timestamp: 17:19

So $\text{Cov}(f_1, f_2)$ would just be the summation as above. $\text{Cov}(f_1, f_2)$ can also be represented as $(f_1^T f_2)/n$.

$$S_{d \times d} = \frac{1}{n} \underbrace{(X^T)(X)}_{\substack{\text{data-matrix} \\ n \times d}} = (d \times d) \checkmark$$

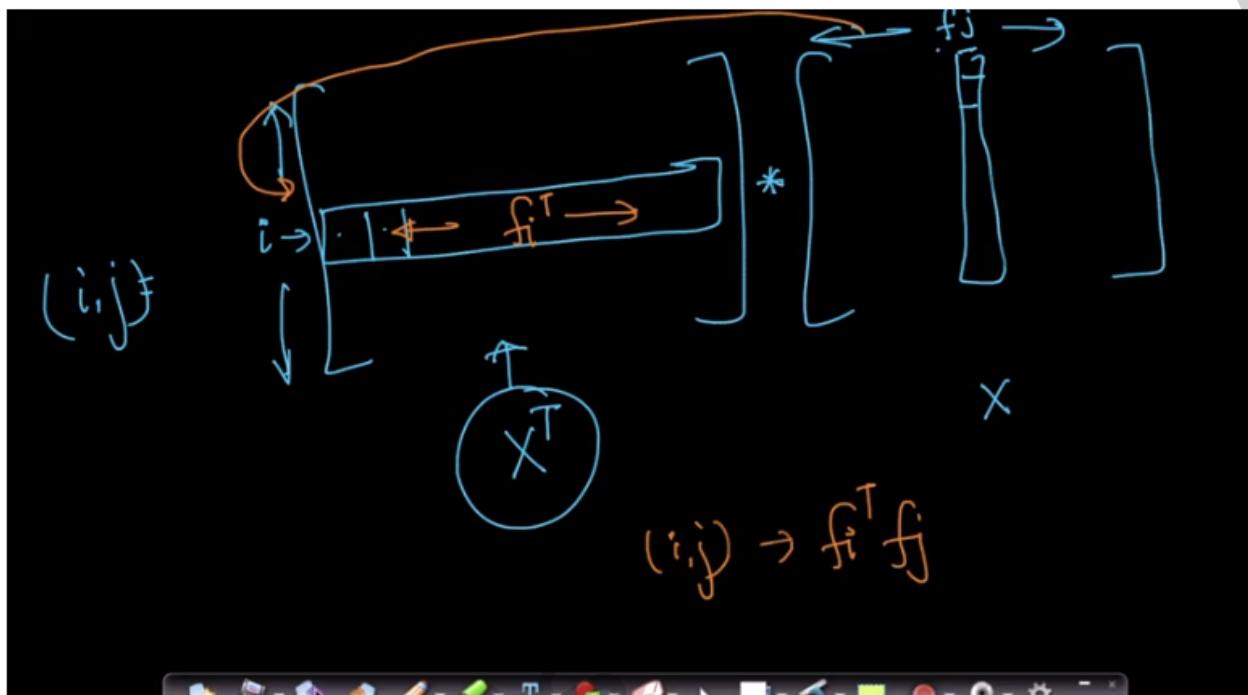
(* assuming X has been col. std.)

LHS $S_{ij} = \text{Cov}(f_i, f_j) = \frac{f_i^T f_j}{n}$

Timestamp: 19:40

Since each element S_{ij} of covariance matrix S is the covariance of feature f_i and f_j , and the $\text{cov}(f_i, f_j) = (f_i^T f_j)/n$, therefore $S_{ij} = (f_i^T f_j)/n$.

We can prove that $S = (X^T X)/n$, by using the below figure.



Timestamp: 22:24

From the above figure we can see the (i,j) the element of $X^T X$ is $f_i^T f_j$ hence the (i,j) the element of $(X^T X)/n$ will be $(f_i^T f_j)/n$ which is how we defined S_{ij} .

Hence $S = (X^T X)/n$ if X has been column standardized.

25.9 MNIST dataset

Visualizing MNIST: An Exploration

MNIST database - Wikipedia

Chekuri Srikan...

colah.github.io/posts/2014-10-Visualizing-MNIST/

MNIST

60K training datapoints
10K test datapoints {Classification}

MNIST is a simple computer vision dataset. It consists of 28x28 pixel images of handwritten digits, such as:

Every MNIST data point, every image, can be thought of as an array of numbers describing how dark each pixel is. For example, we might think of 1 as something like:

Timestamp: 3:43

MNIST dataset is a computer vision dataset, where we are given 28x28 pixel images containing hand written digits and their corresponding labels.

MNIST dataset

This :- 4 dim. dataset

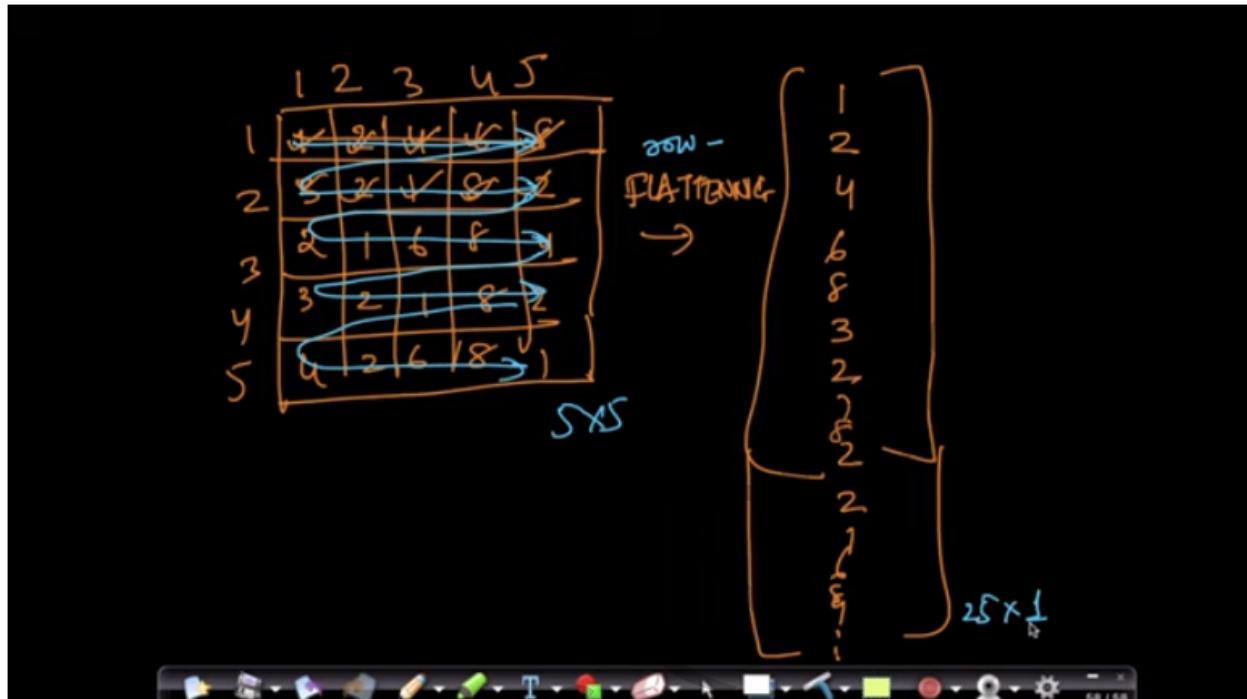
$$\mathcal{D} = \{x_i, y_i\}_{i=1}^{60K}$$

$x_i : \boxed{0}^{28}$ $y_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

obj: classify the written char into one of the 10 numeric char.

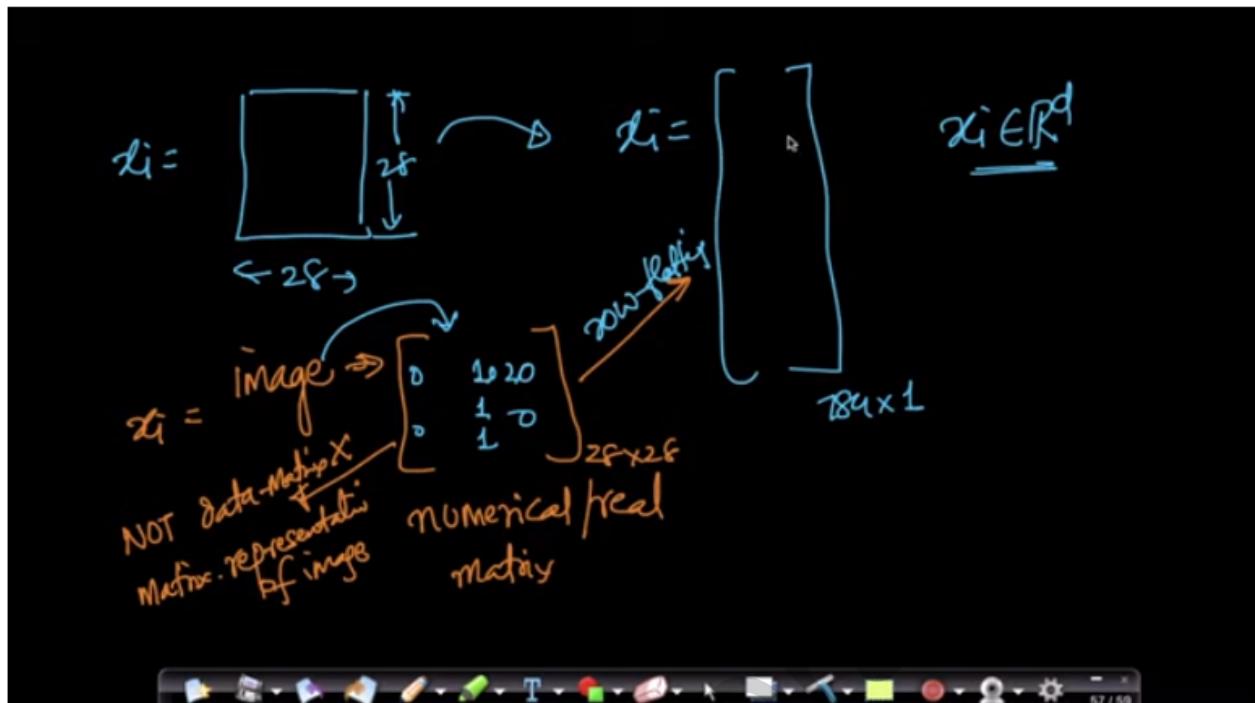
Timestamp: 5:30

The dataset can be represented as $D = \{x_i, y_i\}$ where i is from 1 to n , where each x_i is an image containing 784 pixels which are represented by real values (1 indicating black and 0 indicating white and values between 0 and 1 representing shades of grey) and y_i is the corresponding numeric value of that image between 0 and 9.



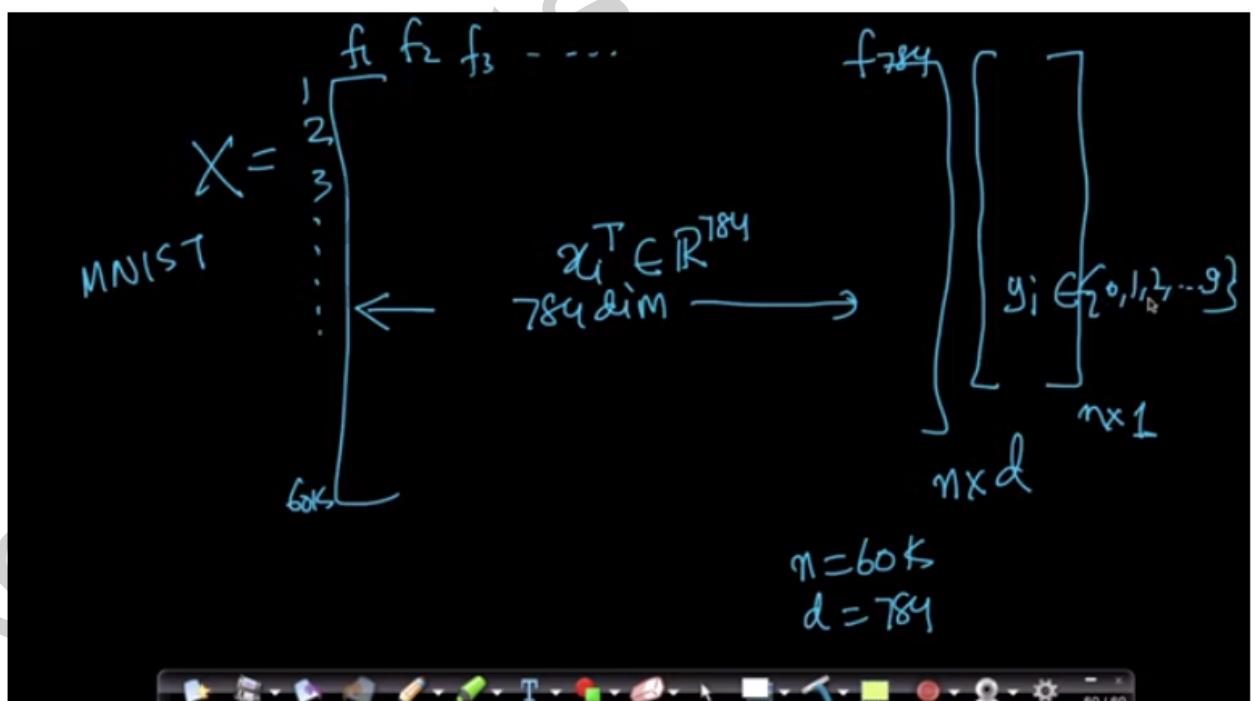
Timestamp: 10:24

Above is an example how we can do row-flattening to convert a matrix of shape 5×5 to 25×1 by first adding all the elements of the first row, followed by all elements of second row and so on.



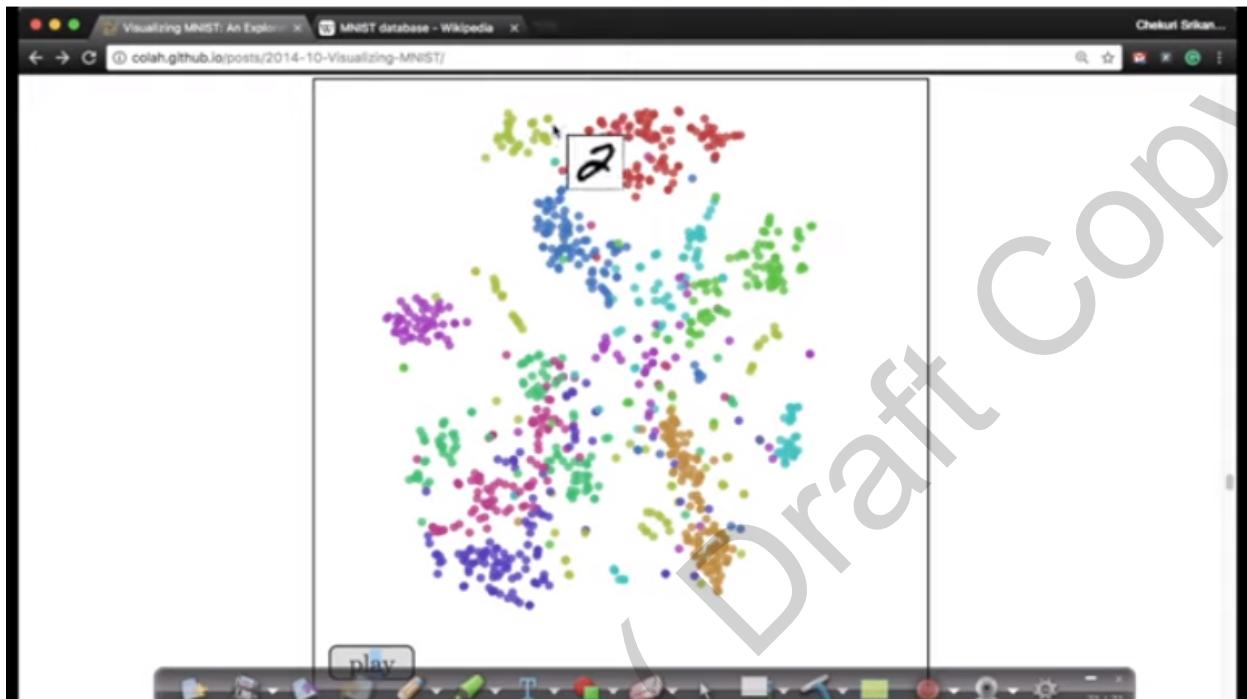
Timestamp: 10:24

So for each data point x_i , if we do row-flattening we get a column vector of 784×1 dimension.



Timestamp: 13:06

So our dataset will be represented as above, where X is the data matrix of dimensions $60K \times 784$ since there are $60K$ training data points and 784 features for each data point and y_i contains the label for the corresponding datapoint x_i and belongs to $[0,9]$.



Timestamp: 18:51

Since we have 784 dimensional data which is difficult to visualize, we use dimensionality reduction technique called t-SNE and above are the results (we will see how to apply t-SNE in later chapters). Notice that each point represents each datapoint in the reduced dimension and the color represents the class-label of the datapoint.

25.10 Code to load MNIST dataset

A screenshot of a Jupyter Notebook interface. The title bar shows 'mnist_loadData_pca_tsne' and 'Digit Recognizer | Kaggle'. The code cell contains Python code for loading the MNIST dataset:

```
In [36]: # MNIST dataset downloaded from Kaggle :  
#https://www.kaggle.com/c/digit-recognizer/data  
  
# Functions to read and show images.  
  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
  
d0 = pd.read_csv('./mnist_train.csv')  
print(d0.head(5)) # print first five rows of d0.  
  
# save the labels into a variable l.  
l = d0['label']  
  
# Drop the label feature and store the pixel data in d.  
d = d0.drop("label",axis=1)
```

Timestep: 1:42

Downloaded the data from the link mentioned, renamed the train csv as mnist_train.csv, imported the necessary libraries and storing the label data and pixel data into l and d respectively.

A screenshot of a Jupyter Notebook interface showing the output of the code execution. The title bar shows 'Code to load MNIST Data set: Dimensionality reduction Lecture 1...' and 'Digit Recognizer | Kaggle'. The code cell shows the command: `d = d0.drop("label",axis=1)`. Below the code, a table is displayed with the following data:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7
0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0

At the bottom of the table, there is a red horizontal line with the text 'pixel8 ... pixel1774 pixel1775 pixel1776 pixel1777 pixel1778 \'. A blue bracket on the left side of the table groups the first five rows (0 to 4). An orange arrow points from the word 'label' in the code cell to the 'label' column header in the table.

Timestamp: 5:06

```
3 4 0 0 0 0 0 0 0 0  
4 0 0 0 0 0 0 0 0 0  
  
pixel8 ... pixel774 pixel775 pixel776 pixel777 pixel778  
0 0 ... 0 0 0 0 0  
1 0 ... 0 0 0 0 0  
2 0 ... 0 0 0 0 0  
3 0 ... 0 0 0 0 0  
4 0 ... 0 0 0 0 0  
  
pixel779 pixel780 pixel781 pixel782 pixel783  
0 0 0 0 0  
1 0 0 0 0  
2 0 0 0 0  
3 0 0 0 0  
4 0 0 0 0  
  
[5 rows x 785 columns]
```

Timestamp: 5:30

As discussed in our data, we have the label column and pixel data from pixel 0 to pixel 783.

In [31]: # display or plot a number.
plt.figure(figsize=(7,7))
idx = 150

grid_data = d.iloc[idx].as_matrix().reshape(28,28) # reshape from 1d to 2d
plt.imshow(grid_data, interpolation = "none", cmap = "gray")
plt.show()

print(l[idx])

0

Timestamp: 9:51

Using the above code, we plot images for us to visualize. The plots look like below.



Timestamp: 9:55