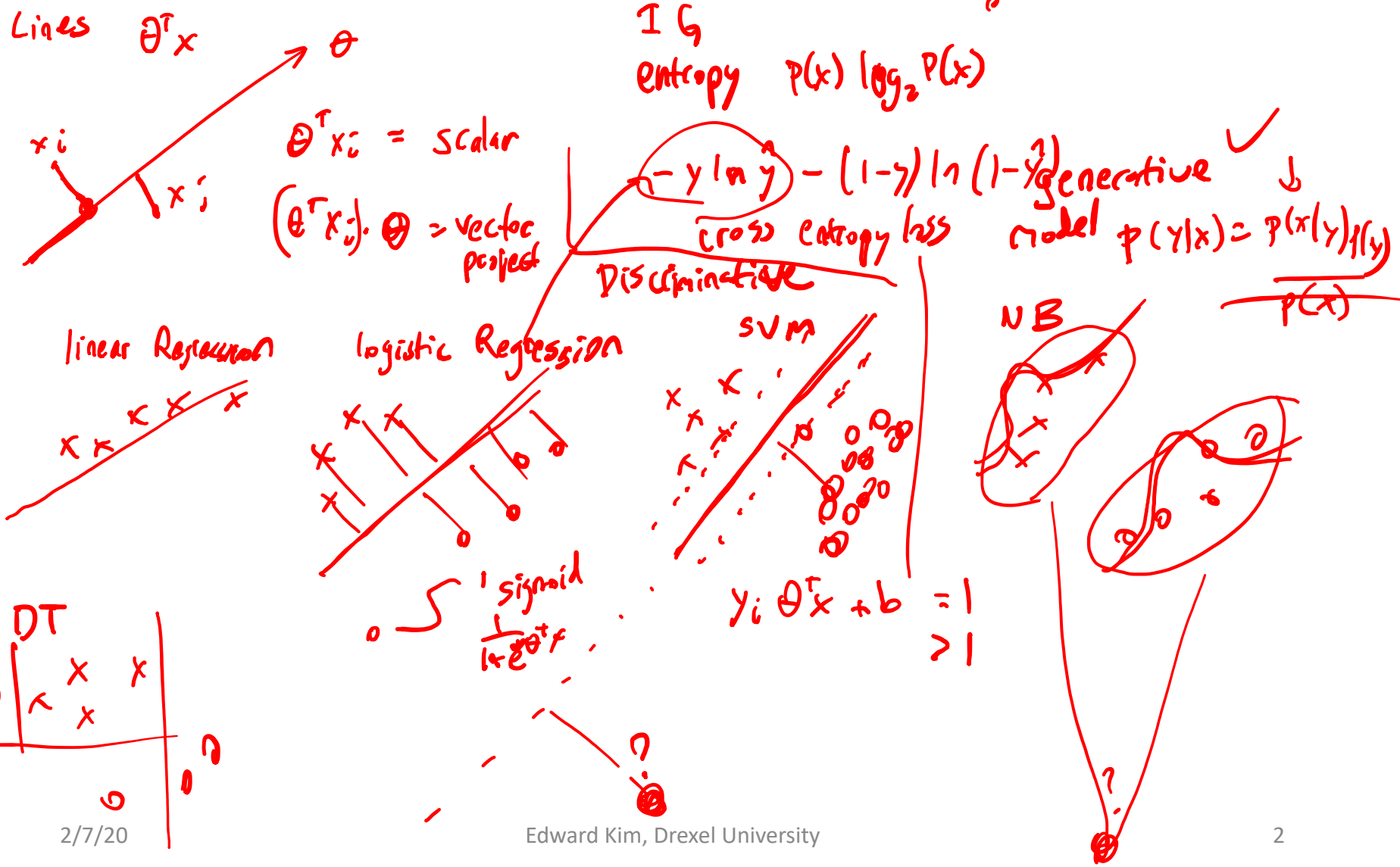
# CS 613 – Machine Learning

## Introduction to Machine Learning

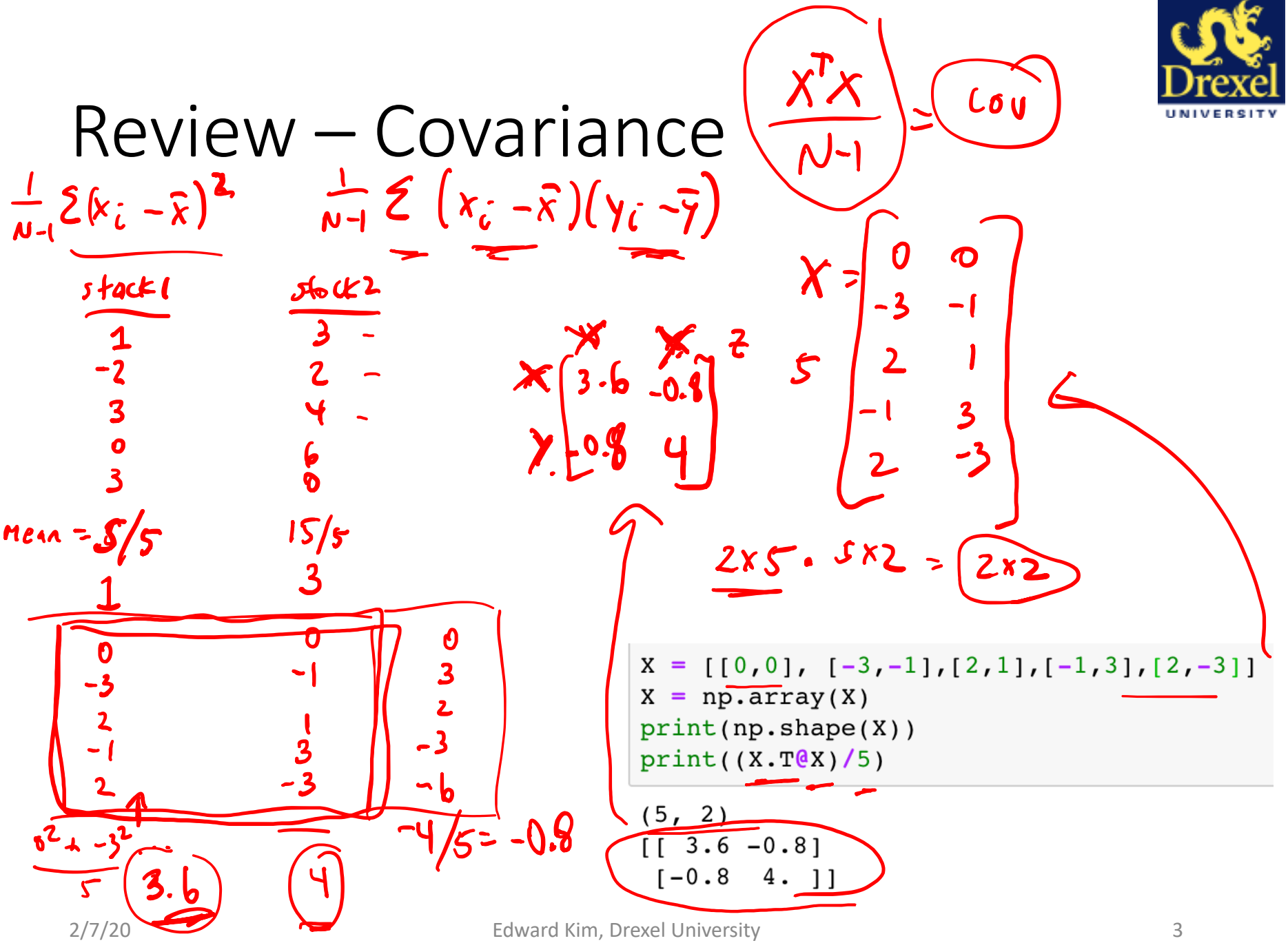
Some slides adapted from Matt Burlick, Drexel

adapted from material created by E. Alpaydin
Prof. Mordohai, Prof. Greenstadt, Pattern Classification (2nd Ed.),
Pattern Recognition and Machine Learning

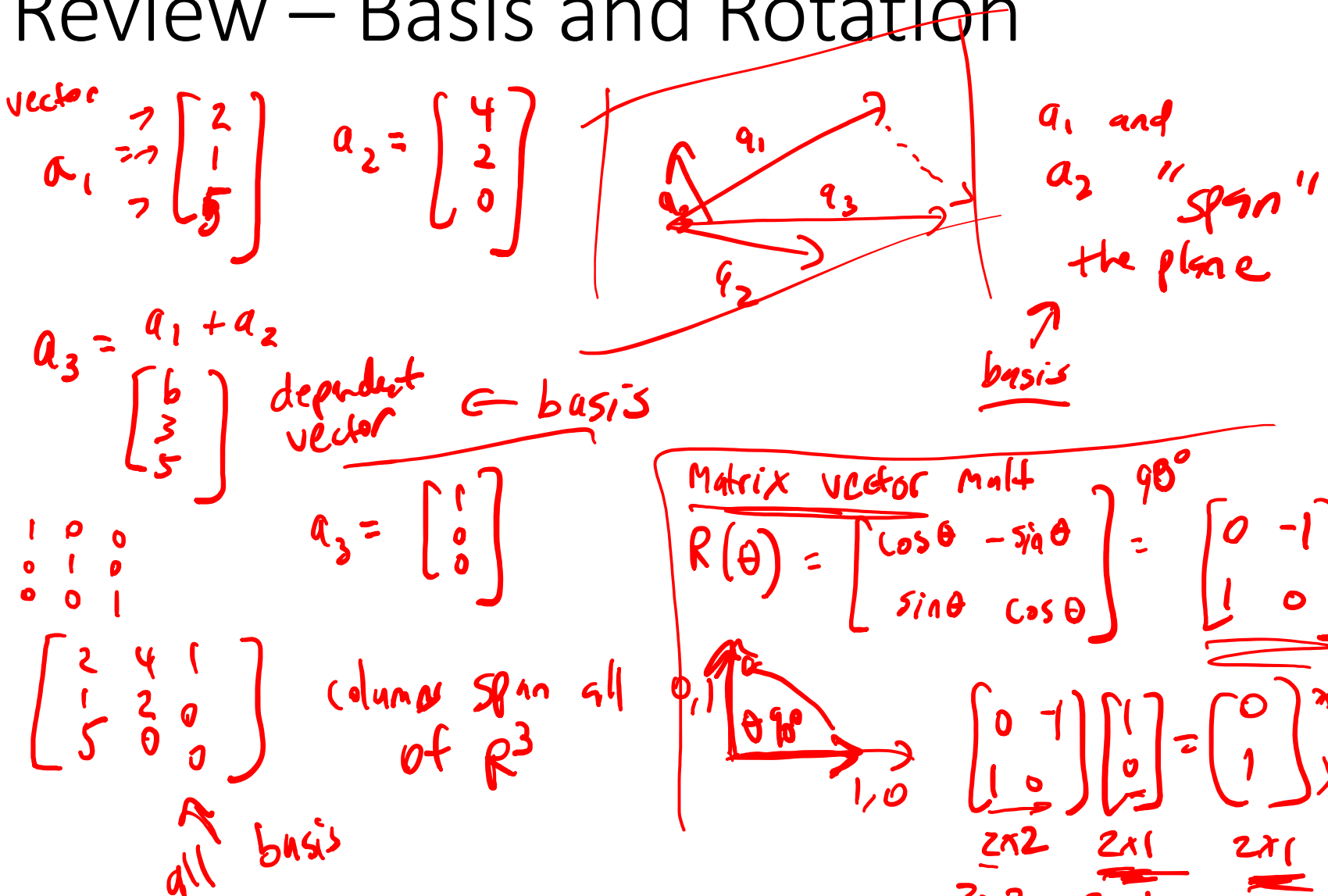
# Review


KNN
K=1
Lines $\theta^T x$
$\theta$
$x_i$
$x_j$
$\theta^T x_i$ = scalar
$(\theta^T x_i) \cdot \theta$ = vector project
IG
entropy $p(x) \log_2 p(x)$
$-y \ln y - (1-y) \ln (1-y)$
cross entropy loss
Discriminative
generative
model $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$
linear Regression
logistic Regression
sigmoid
$\frac{1}{1+e^{\theta^T x}}$
SVM
$y_i \theta^T x + b = 1$
$> 1$
NB
DT

# Review – Covariance



# Review – Basis and Rotation

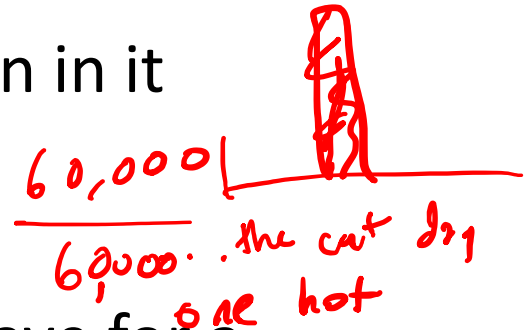


# Data Dimensionality

And the “curse”

# Data Dimensionality

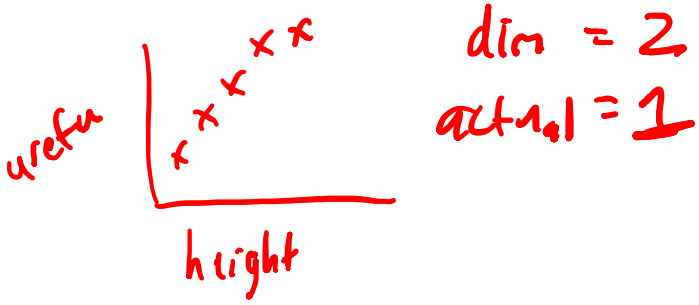
- Typically our data has a lot of information in it
  - An image has millions of pixels
  - A textbook has thousands of words
- We call the amount of information we have for a given data sample, it's *dimension*
- Therefore if data sample $X_i$ has *D* values associated with it (which we call **features**) then we can call *D* the *dimensionality* of $X_i$



# Data Dimensionality

- Can there be drawbacks of too much information (too high of dimensionality)?
- Computation cost
  - Both time and space efficiency
- Redundant
  - Representing the same thing
- Visualization
  - How can we look at the data to understand its structure?

# Redundant Features




dim = 2
actual = 1

collecting data predict standard of living
# accident icy road
# burst pipes
cost snow plows taxes
# school closures
# people w/ heat stroke
temp

$$A = \begin{bmatrix} 1 & 0.8 \\ 1 & 0.8 \\ 1 & 0.8 \end{bmatrix}$$

D

linear dependence

A is singular non invertible

$Ax = 0$ some $x$ non zero

determinant of A = 0

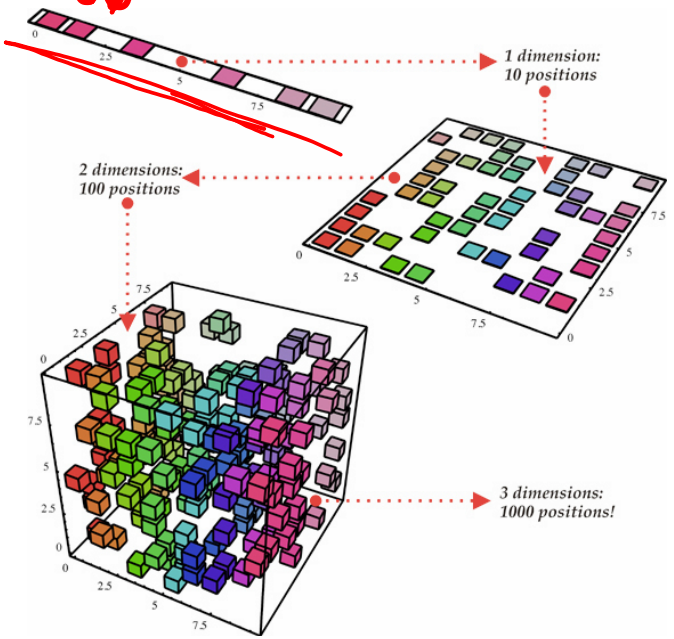
# Redundant Features

```
X = [[1,1], [1,1]]
X = np.array(X)
X = np.linalg.inv(X)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
<ipython-input-42-4d9e91b4b701> in <module>
      1 X = [[1,1], [1,1]]
      2 X = np.array(X)
----> 3 X = np.linalg.inv(X)

<__array_function__ internals> in inv(*args, **kwargs)

/opt/local/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/numpy/linalg/linalg.py in inv
(a)
    549     signature = 'D->D' if isComplexType(t) else 'd->d'
    550     extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 551     ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
    552     return wrap(ainv.astype(result_t, copy=False))
    553

/opt/local/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/numpy/linalg/linalg.py in _ra
ise_linalgerror_singular(err, flag)
     95
     96 def _raise_linalgerror_singular(err, flag):
---> 97     raise LinAlgError("Singular matrix")
     98
     99 def _raise_linalgerror_nonposdef(err, flag):

LinAlgError: Singular matrix
```

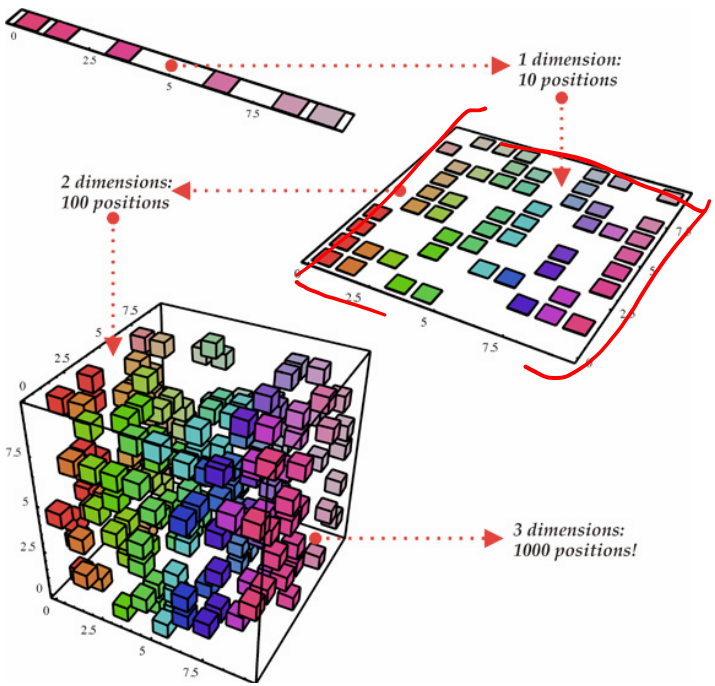
# Curse of Dimensionality

- Imagine we only have one dimension
- Ideally we'd have a sample for every single possible location in this space.
- If we have discrete space, and the range is [0, 9], then we'd just need 10 samples to have complete coverage
  - Great. No big deal!



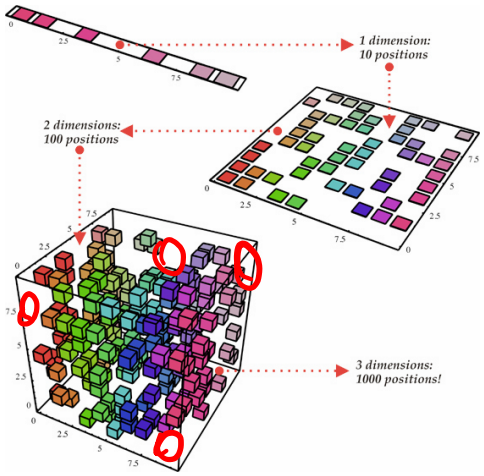
# Curse of Dimensionality

- How about if we had two features ($D = 2$)?
- To have the same coverage we'd need $10^2$=100 samples



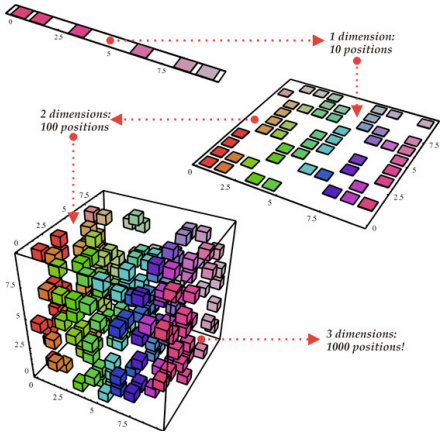
# Curse of Dimensionality

- Ok. But in the real world our data ends up having $D$ be really large
  - If we're trying to classify an image, each pixel is a feature, and thus $D = millions$
  - And therefore to have complete coverage we'd need $10^{millions}$ samples
  - Impossible!



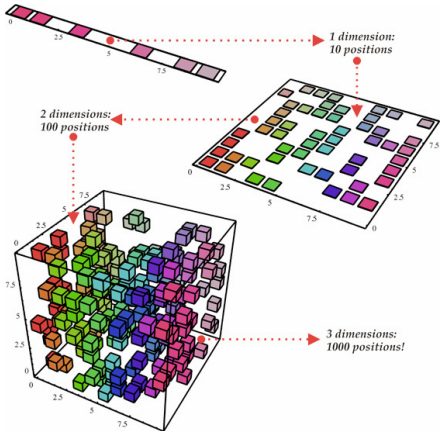
# Curse of Dimensionality

- This is the *curse of dimensionality*
  - In higher dimension space we need exponentially more samples for equivalent coverage.
- Therefore if our data is in high dimensional space it will likely sparsely cover that space
  - And instances will be far apart from one another



# Curse of Dimensionality

- This might be one motivation for *dimensionality reduction*
  - Going from higher dimension data to lower
- However we want to do this "intelligently"
  - We don't want to lose much important information by doing this!



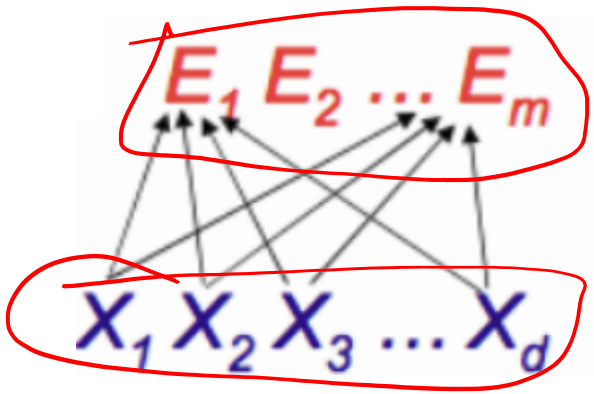
# Dimensionality Reduction

# Dimensionality Reduction

- Goal: Represent instances with fewer variables
  - Try to preserve as much structure in the data as possible
  - If there is class information
    - Discriminative: increase class separability
- Benefits:
  - Need less data to cover the feature space
  - Easier learning – fewer parameters to learn
  - Easier visualization – hard to visualize more than 3D or 4D

# Dimensionality Reduction

- Approaches
  - Feature selection
    - Pick a subset of the original dimensions
    - If there is class information
      - Discriminative: Pick good class predictors
  - Feature construction
    - Construct a new set of features from existing
    - In particular we'll look at feature projection



# Objectives

- Feature Projection (PCA, LDA)

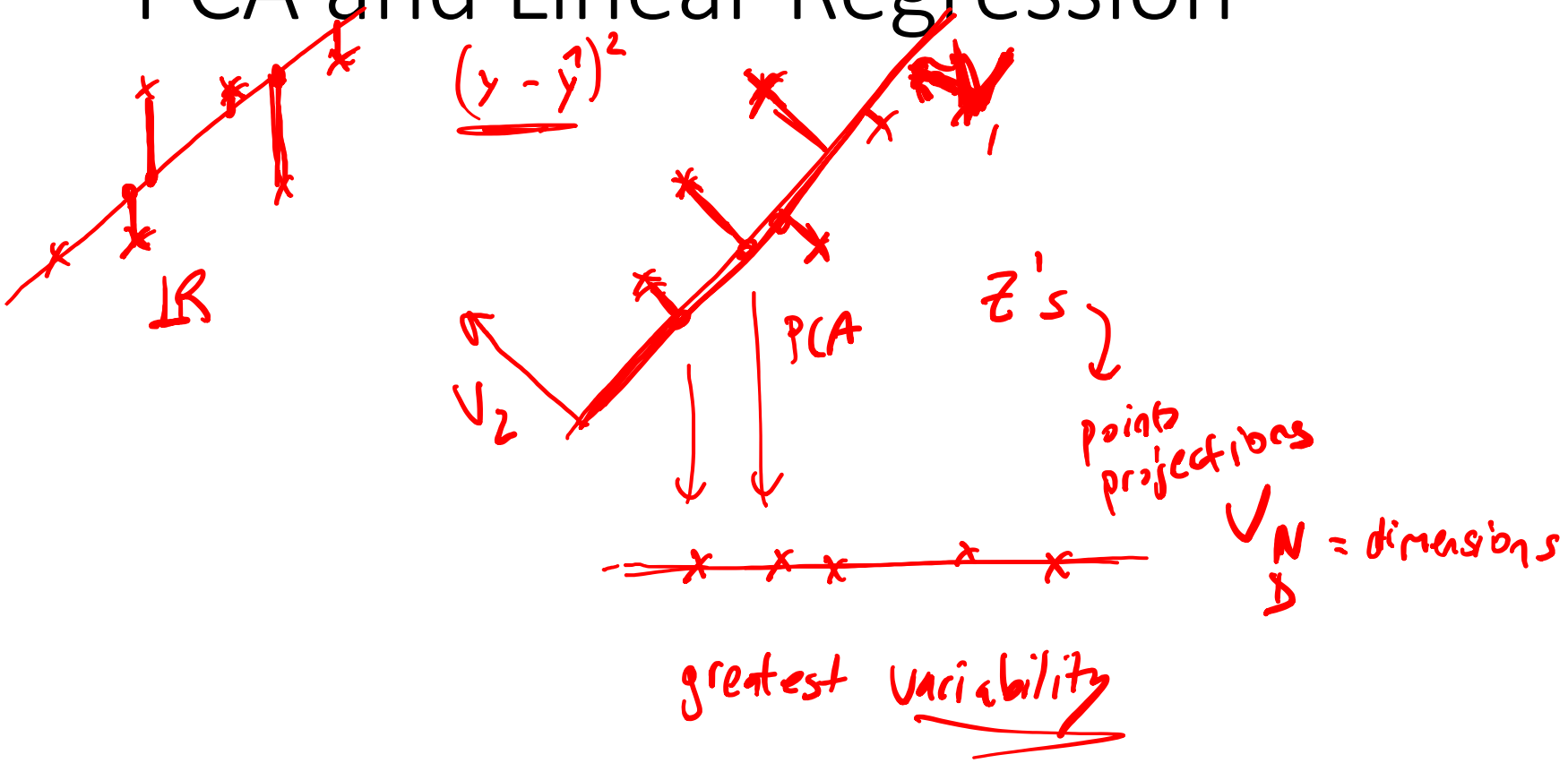
# Additional Resources

- http://people.cs.pitt.edu/~milos/courses/cs3750-Fall2011/lectures/class16.pdf
- http://www.sci.utah.edu/~shireen/pdfs/tutorials/Elhabian_LDA09.pdf

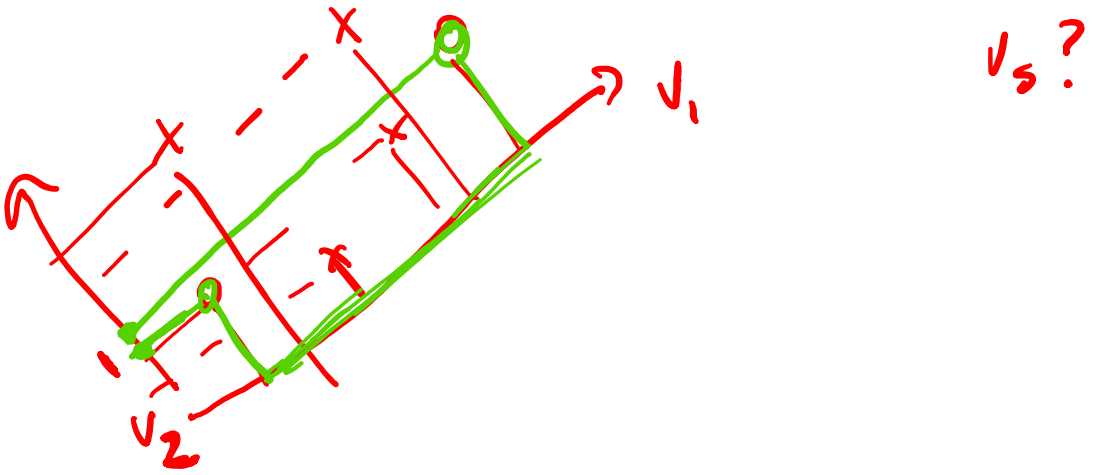
# Feature Construction

- We said another way to reduce the dimensionality of our data is to construct new features from the existing ones.
- Hopefully we can some "encode" information from higher dimensional data in these new features so that we don't lose much information
- In particular we'll look at the idea of *feature projection*

# PCA and Linear Regression

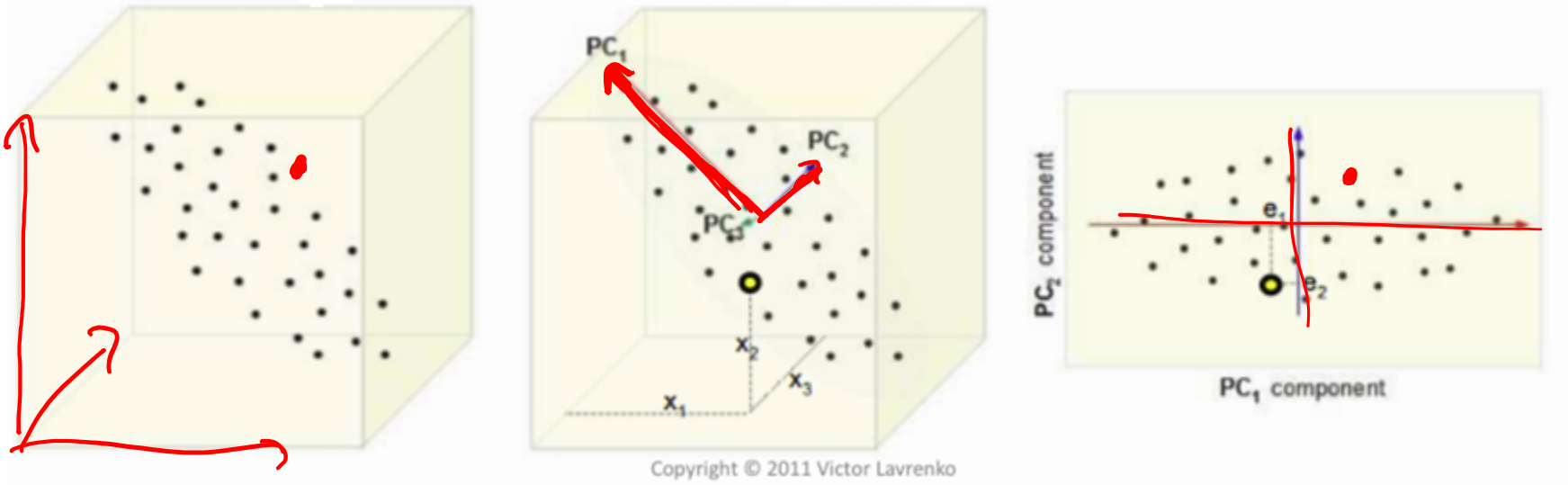


# Why greatest variability?



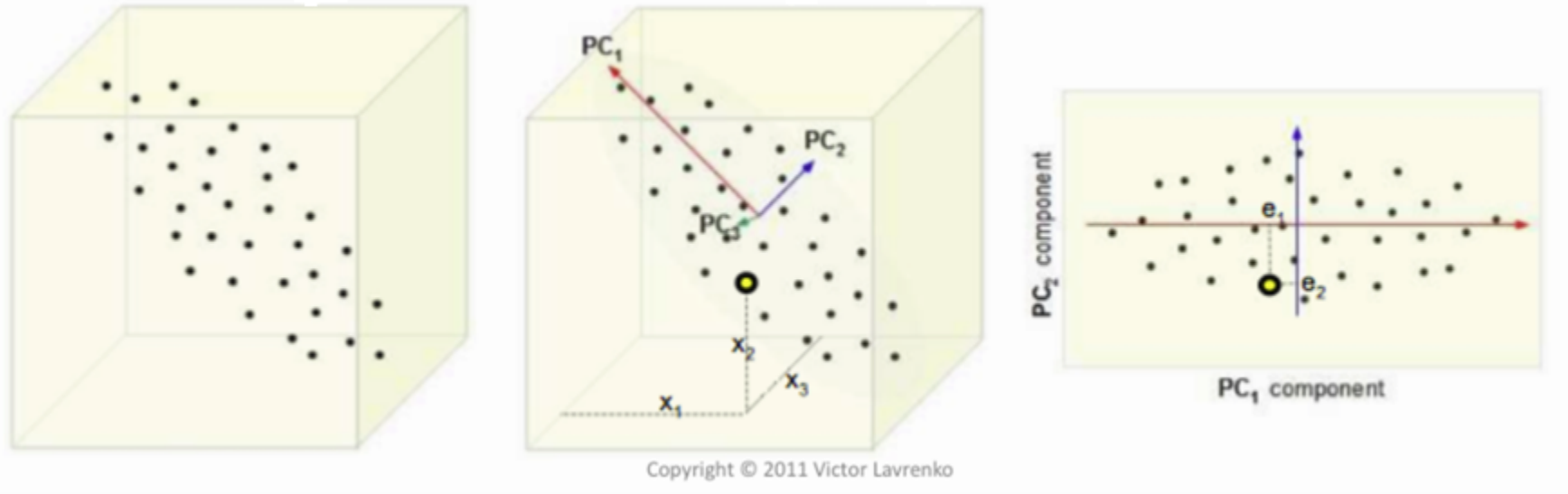
# Feature Projection by PCA

- Principal component analysis (PCA) defines a set of principal components (basis)
  - 1st: direction of the greatest variability in the data
  - 2nd: perpendicular to 1st, greatest variability of what's left
  - Etc… until $D$, the original dimensionality



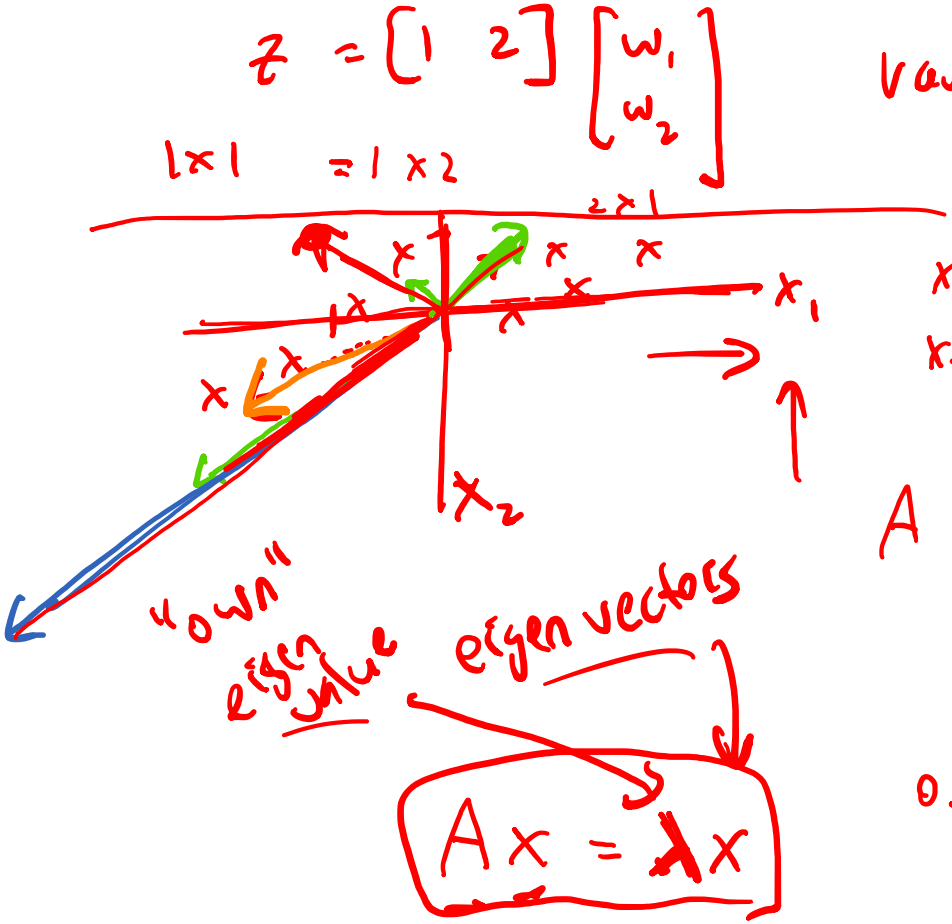
# Feature Projection by PCA

- We can then choose the number of dimensions we want, $k < D$ and project the original data onto the principal components
- Each projection will result in a point on that axis, resulting in an new $k$-dimensional feature vector



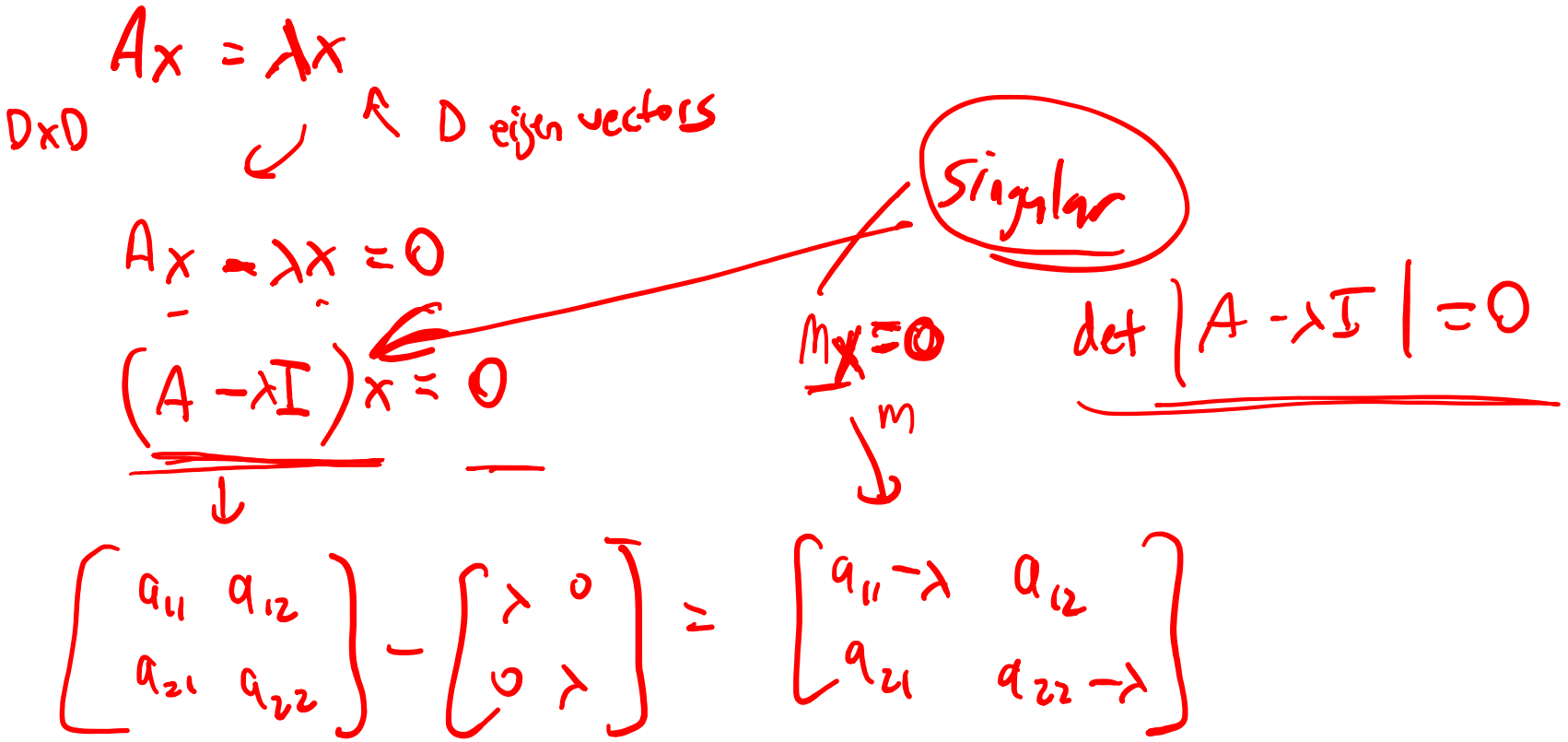
# PCA Derivation

- We want to find new points $Z = Xw$



# PCA Derivation

- We want to find new points $Z = Xw$



# PCA Derivation

$\Sigma = A$

$Aw = \lambda w$

- We want to find new points $Z = Xw$

$\max \; var(Xw)$

$(Xw)^T(Xw)$

$\frac{w^T X^T X w}{N-1} \Rightarrow \max \; w^T \Sigma w$

Vectors maximize variance are eigenvectors of covariance matrix

$J(w) = w^T \Sigma w$

s.t. $\|w\| = 1$

$\lambda J(w) = w^T \Sigma w - \lambda(w^T w - 1)$

$\frac{\partial J}{\partial w} = 2\Sigma w - 2\lambda w = 0$

$\Sigma w = \lambda w$

# PCA Derivation

- We want to find new points $Z = Xw$
- Given $w$ as a $D \times 1$ column vector, the projection onto this axis is:

$$Z = Xw$$

- We want to maximize the variance of $Z$

$$\mathrm{w}^* = argmax_w\big(Var(Z)\big)$$

$$= arg\max_w\big(Var(Xw)\big)$$

- Assuming our data's columns are zero mean (which they should be if we standardized our data)

$$\mathrm{Var}(Xw) = \frac{(Xw)^T(Xw)}{N-1} = \frac{w^T X^T Xw}{N-1} = w^T \Sigma w$$

  - Where $\Sigma$ is the covariance matrix of $X$

# PCA Derivation

$$w^* = argmax_w(w^T \Sigma w)$$

- To enforce that $w$ is unit length (otherwise, we would maximize this by just setting $w = [\infty, \dots, \infty]^T$) we can add the penalty term $\lambda(w^T w - 1)$
- All put together we get the cost function:

$$J(w) = \mathrm{w}^{\mathrm{T}} \Sigma w - \lambda(w^T w - 1)$$

- Maximizing this, we get the ***Lagrange*** problem:

$$w^* = \mathrm{argmax}_w \left( \mathrm{w}^{\mathrm{T}} \Sigma w - \lambda(w^T w - 1) \right)$$

# PCA Derivation

$$\text{argmax}_w \left( \text{w}^{\text{T}} \Sigma w - \lambda (w^T w - 1) \right)$$

- Taking the derivative and setting this equal to zero we get:
$$2\Sigma w - 2\lambda w = 0$$
- Which becomes $(\Sigma - \lambda I)w = 0$
- We could also write this as $\Sigma w = \lambda w$
- Since $\Sigma$ is a matrix, $\lambda$ is a scalar, and the thing that we're solving for, $w$, is a vector, this can be solved using ***eigen-decomposition***!

# Eigen Decomposition

$$(\Sigma - \lambda I)w = 0$$

- We know the matrix $\Sigma$ and we want to determine the pairs $(\lambda, w)$ that make this true.
- We call these *eigenvalue, eigenvector* pairs.
- If $\Sigma$ is a $D \times D$ square matrix, there will be $D$ eigenvalue, eigenvector pairs that make this equation true.

# Eigenvalues/Eigenvectors

- While you'll usually just use some package to get the eigenvalues and eigenvectors for you, let's make sure we can do it manually as well.
- The following is just a repeat of what in the Week 0 Linear Algebra slides.

# Eigenvalues/Eigenvectors

- The basic equation in the eigenvalue problem is:

$$Ax = \lambda x$$

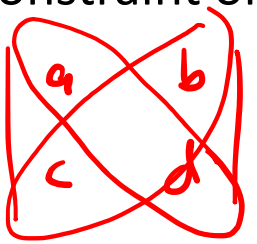
- Where $A$ is a square matrix, $x$ is a vector, and $\lambda$ is a scalar
- The vector $x$ is called an *eigenvector* and the scalar $\lambda$ is called an *eigenvalue*
- Real eigenvalues only exist if $A$ is a square matrix and the determinant of $A - \lambda \mathrm{I}$ is equal to zero

# Finding Eigenvalues/Eigenvectors

$(A - \lambda I)x = 0$

- To find the eigen-values and vectors let's use that constraint on the determinant:

$$|A - \lambda \mathrm{I}| = 0$$




$= ad - bc$

- Let's look at finding them for a 2x2 matrix
  - $|A - \lambda I| = \begin{vmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{vmatrix}$
  - $= (a_{11} - \lambda)(a_{22} - \lambda) - a_{12}a_{21} = 0$
  - $\lambda^2 - \lambda(a_{11} + a_{22}) + (a_{11}a_{22} - a_{12}a_{21}) = 0$
- We want to solve for $\lambda$
  - What type of equation is this?
  - How can we solve for it?

$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

- Once we know the values of $\lambda$ we can then solve for the eigenvectors $x$
  - For each value of $\lambda$ take the original equation $Ax = \lambda x$ and solve the equation $(A - \lambda I)x = 0$

# Eigenvalues/Vectors

- The general procedure (called eigen-decomposition) is:
  1. Compute $|A - \lambda I|$
  2. Find the roots of the polynomial given by $|A - \lambda I| = 0$
  3. Solve the system of equations $(A - \lambda I)x = 0$
- In MATLAB you can get these by:
  - `W,V = np.linalg.eig(A)`
  - The columns of $V$ will be the eigen-vectors
  - You also may want to normalize the eigen-vectors to be of unit length: $V_{:,i}/|V_{:,i}|$
- Singular Value Decomposition (SVD) provides another, faster way to get eigenvalues and eigenvectors
  - We won't go into it, but if your using a package that has it, use it

# Eigen Example

- Find the eigenvalues and eigenvectors of

$$A = \begin{bmatrix} 1 & 2 \\ 5 & 3 \end{bmatrix}$$

  - Recall for a 2x2 matrix
    $\lambda^2 - \lambda(a_{11} + a_{22}) + (a_{11}a_{22} - a_{12}a_{21}) = 0$
  - Then we need to solve $(A - \lambda I)x = 0$

$\lambda^2 - \lambda(1+3) + (3 - 10) = \lambda^2 - 4\lambda - 7$

$\lambda = \frac{4 \pm \sqrt{16 - 4 \cdot 1 \cdot -7}}{2a} = \frac{4 \pm \sqrt{44}}{2} = \frac{4 \pm 2\sqrt{11}}{2}$

$\lambda = 2 \pm \sqrt{11}$

$\lambda = 2 \pm \sqrt{11}$ $\quad (A - \lambda I)x = 0$ $\quad \begin{bmatrix} 1 & 2 \\ 5 & 3 \end{bmatrix}$ $\quad \frac{\theta}{\|\theta\|}$

$$\begin{bmatrix} 1-(2+\sqrt{11}) & 2 \\ 5 & 3-(2+\sqrt{11}) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$

$$x_1 = \begin{bmatrix} 1 \\ \frac{1+\sqrt{11}}{2} \end{bmatrix}$$

$$\begin{bmatrix} -1-\sqrt{11} & 2 \\ 5 & 1-\sqrt{11} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$

$1^2 + \left(\frac{1+\sqrt{11}}{2}\right)^2$

$(-1-\sqrt{11})x_1 + 2x_2 = 0 \quad \Leftarrow \quad x_2 = \frac{1+\sqrt{11}}{2} x_1$

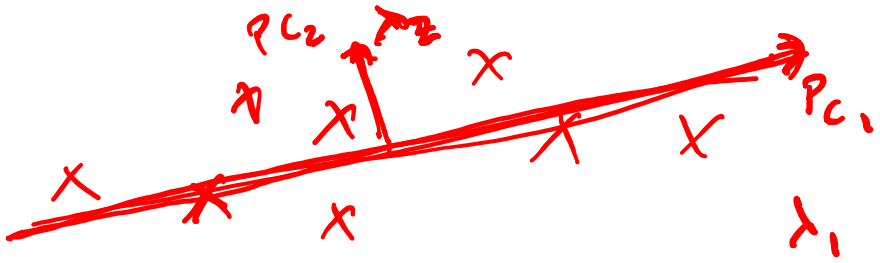
$5x_1 + (1-\sqrt{11})x_2 = 0$

$5x_1 + (1-\sqrt{11})\left(\frac{1+\sqrt{11}}{2}\right)x_1 = 0$ $\quad \frac{-10}{2} = -5$

$5x_1 - 5x_1 = 0$

$5x_1 = 5x_1 \quad x_1 = x_1$

# PCA

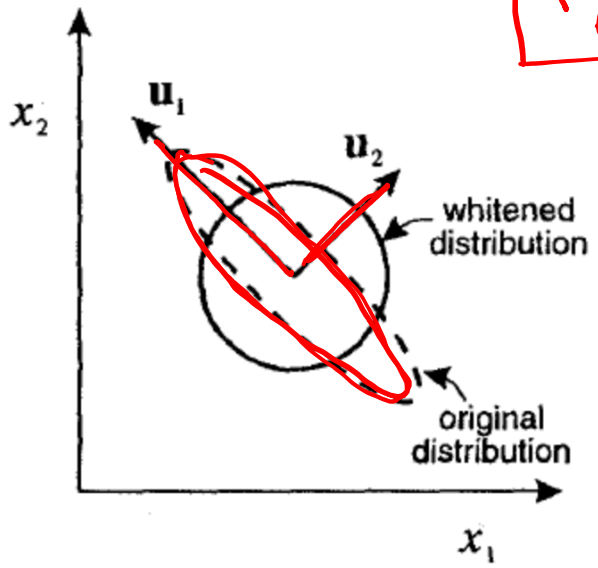


1. Start from $N \times D$ data matrix $X = \{X_i\}_{i=1}^{N}$ ($N$ data points/samples, d measurement types/features)
2. Standardize the data
3. Compute covariance matrix (results in $D \times D$ matrix), $\Sigma$
   - Revisit Lecture 0 Linear Algebra review
4. Compute eigenvectors and eigenvalues of $\Sigma$ and normalize them (unit length)
   - Python: `val,vec =` $\text{np.linalg.}eig(\Sigma)$
   - Eigenvectors are the columns of `vec`
   - Eigenvalues are on the diagonal of `val`

# PCA Whitening

- Want to make your features more discriminative
- Decorrelate the data

$$\mathbf{w} = \Lambda^{-1/2}\mathbf{y} = \Lambda^{-1/2}\Phi^T\mathbf{x}$$



# Choosing $k$

- Next we want to choose the $k$ eigenvectors that with highest eigenvalues
- How do we decide $k$?
  - User?
  - Fraction of variation explained by first $k$ principle components:

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i} \geq \alpha$$

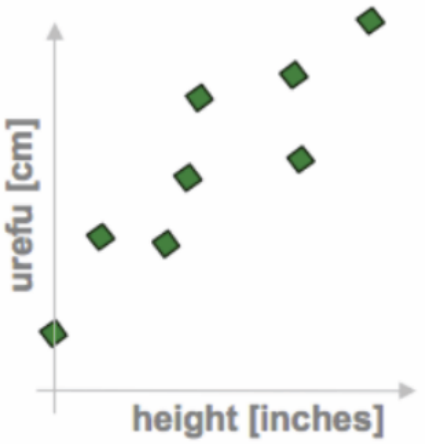
    - Typical threshold $\alpha$ values 0.9 or 0.95

# Using PCA for Dimensionality Reduction

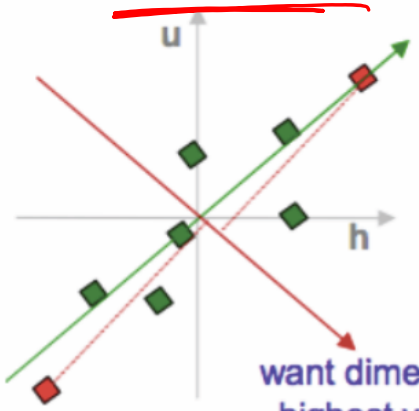
- Now we have a set of k principal components $e_1, \ldots, e_k$
  - Orthogonal, unit length
- Concatenated, they form an $D \times k$ *projection matrix* $W = [e_1, \ldots, e_k]$
- Now can *project* our $D$-dimensional data into $k$-dimensions
  - $Z = XW$

# PCA in a nutshell

1. correlated hi-d data
("urefu" means "height" in Swahili)




2. center the points




want dimension of highest variance

3. compute covariance matrix

$$\begin{array}{c|cc} & h & u \\ h & 2.0 & 0.8 \\ u & 0.8 & 0.6 \end{array} \quad \text{cov}(h,u) = \frac{1}{n}\sum_{i=1}^{n} h_i u_i$$

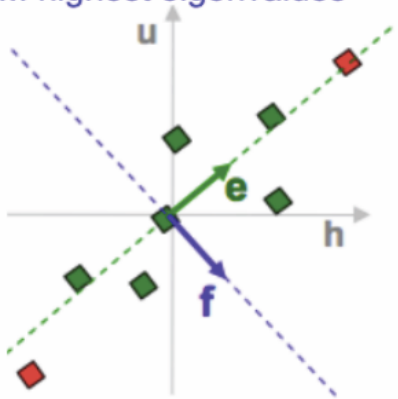
4. eigenvectors + eigenvalues

$$\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix} \begin{pmatrix} e_h \\ e_u \end{pmatrix} = \lambda_e \begin{pmatrix} e_h \\ e_u \end{pmatrix}$$

$$\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix} \begin{pmatrix} f_h \\ f_u \end{pmatrix} = \lambda_f \begin{pmatrix} f_h \\ f_u \end{pmatrix}$$

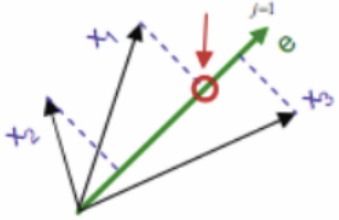
`eig(cov(data))`

5. pick m<d eigenvectors w. highest eigenvalues

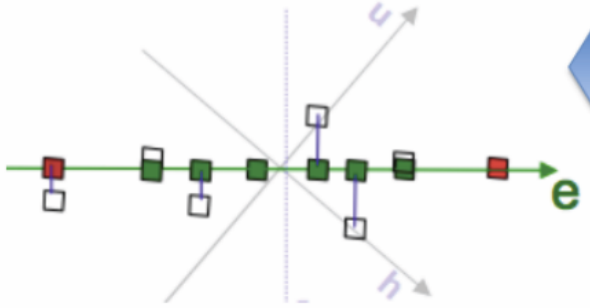



6. project data points to those eigenvectors

$$x_e' = x^T e = \sum_{j=1}^{d} x_j e_j$$




7. uncorrelated low-d data



# Example

- Assume data

$$X = \{(4,1,2), (2,4,0), (2,3,-8), (3,6,0), (4,4,0), (9,10,1), (6,8,-2)(9,5,1), (8,7,10), (10,8,-5)\}$$

- What is the first principal component?
- What are the observations' values projected onto that component?

# Example

- Assume data

$$X = \{(4,1,2), (2,4,0), (2,3,-8), (3,6,0), (4,4,0), (9,10,1), (6,8,-2)(9,5,1), (8,7,10), (10,8,-5)\}$$

1. First lets standardize our data
   - $\mu_1$=(4+2+2+3+4+9+6+9+8+10)/10 = 5.7
   - $\mu_2$= (1+4+3+6+4+10+8+5+7+8)/10 = 5.6
   - $u_3 = -0.1$
   - $\sigma_1 = 3.093, \sigma_2 = 2.7162, \sigma_3 = 4.7011$
   - So our new (standardized) data is

$$X = [(-0.55, -1.69, 0.45), (-1.2, -0.59, 0.02), (-1.2, -0.96, -1.68), (-0.87, 0.147, 0.21), (-0.55, -0.59, 0.21), (1.067, 1.62, 0.23), (0.097, 0.88, -0.40), (1.067, -0.22, 0.23), (0.74, 0.515, 2.15), (1.39, 0.88, -1.04)]$$

# Example

2. Compute covariance matrix

$$\Sigma(X) = \sigma(X, X)$$

- This is actually quite easy if we have already centered the data!
  - $\Sigma(X) = \frac{X^T X}{N-1} = \begin{bmatrix} 1.0 & 0.6851 & 0.2575 \\ 0.6851 & 1.0 & 0.1096 \\ 0.2575 & 0.1096 & 1.0 \end{bmatrix}$

# Example

- $\Sigma(X) = \frac{X^T X}{N-1} = \begin{bmatrix} 1.0 & 0.6851 & 0.2575 \\ 0.6851 & 1.0 & 0.1096 \\ 0.2575 & 0.1096 & 1.0 \end{bmatrix}$

3. Get the Eigenvalues/vectors of the covariance matrix
   - Eigenvalues = [0.2982, 0.9287, 1.7731]
   - Eigenvectors

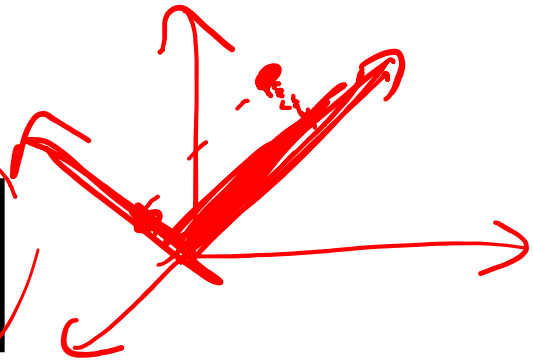
$$\begin{bmatrix} 0.7188 \\ -0.677 \\ -0.158 \end{bmatrix}, \begin{bmatrix} 0.1141 \\ 0.3391 \\ -0.9338 \end{bmatrix}, \begin{bmatrix} 0.6858 \\ 0.6532 \\ 0.3211 \end{bmatrix}$$

# Example

$X =$
$[(-0.55, -1.69, 0.45), (-1.2, -0.59, 0.02), (-1.2, -0.96, -1.68),$
$(-0.87, 0.147, 0.21), (-0.55, -0.59, 0.21), (1.067, 1.62, 0.23),$
$(0.097, 0.88, -0.40), (1.067, -0.22, 0.23), (0.74, 0.515, 2.15),$
$(1.39, 0.88, -1.04)]$

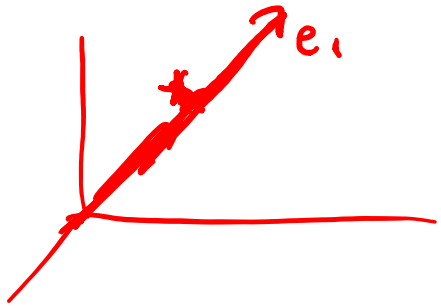
- Eigenvalues = [0.2982, 0.9287, 1.7731]
- Eigenvectors

$$\begin{bmatrix} 0.7188 \\ -0.677 \\ -0.158 \end{bmatrix}, \begin{bmatrix} 0.1141 \\ 0.3391 \\ -0.9338 \end{bmatrix}, \begin{bmatrix} 0.6858 \\ 0.6532 \\ 0.3211 \end{bmatrix}$$



- Finally let's project the points onto the single best vector (i.e. the one with the highest eigenvalue). Note we'll do this on the standardized data
  - $Z_{1,1} = X_1 W$ = [-0.55,-1.69, 0.45] $\begin{bmatrix} 0.6858 \\ 0.6532 \\ 0.3211 \end{bmatrix}$ = -1.3397
  - Etc…
  - But actually just do it all at once as $Z = XW$

# Reconstruction




- The reconstruction of our observation can be done by taking a linear combination of the eigen-vectors with the projection.
- From our previous example we had ten observations, each with three features.
- Therefore we had three eigen-vectors.
- We then projected an observation into this new space.
- Using just one eigen-vector we got:
  - Observation 1 $[-0.55, -1.69, 0.45] \rightarrow [-1.3397]$
- Conceptually we can think of this as -1.3397 units in the direction of the first eigen-vector, $\begin{bmatrix} 0.6858 \\ 0.6532 \\ 0.3211 \end{bmatrix}$

# Reconstruction

- We could have projected this observation onto all three eigen-vectors:
  - Observation 1 $[-0.55, -1.69, 0.45] \rightarrow [-1.3397, -1.0542, 0.6809]$
- Which again we can think about as -1.3397 units down the first eigen-vector, -1.0542 down the second, etc..
- Think about what the vector (2,3) would mean in Cartesian coordinates….

# Reconstruction

- Let $z_i$ be the $i^{th}$ coordinate in our new projection space, and $e_i$ be the $i^{th}$ eigen-vector, and $W$ the matrix of the $k$ most relevant eigen-vectors.
- If we want to return to the original space we can do the reconstruction as:

$$\hat{x} = \sum_{i=1}^{k} z_i e_i = zW^T$$

- Proof:
  - Given: $z = xw, w^T w = 1$
  - If $x = zw^T$ then:

$$z = xw = (zw^T)w = z$$

# Reconstruction

$$\hat{x} = zW^T$$

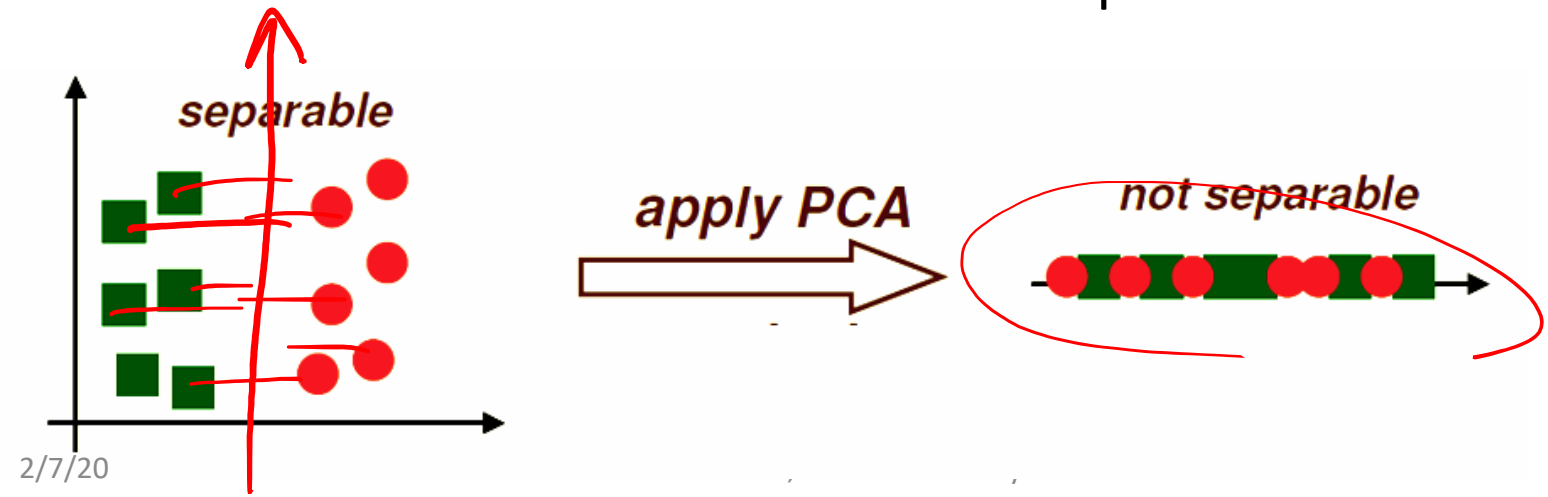
- So for observation 1, if we use $k = 3$ this is:

$$[-1.3397, -1.0542, 0.6809] \begin{bmatrix} 0.6858 & 0.1141 & 0.7188 \\ 0.6532 & 0.3391 & -0.677 \\ 0.3211 & -0.9338 & -0.158 \end{bmatrix}^T$$

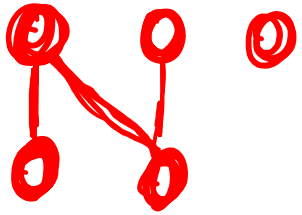
$$= [-0.5496, -1.6935, 0.4467]$$

# PCA Issues

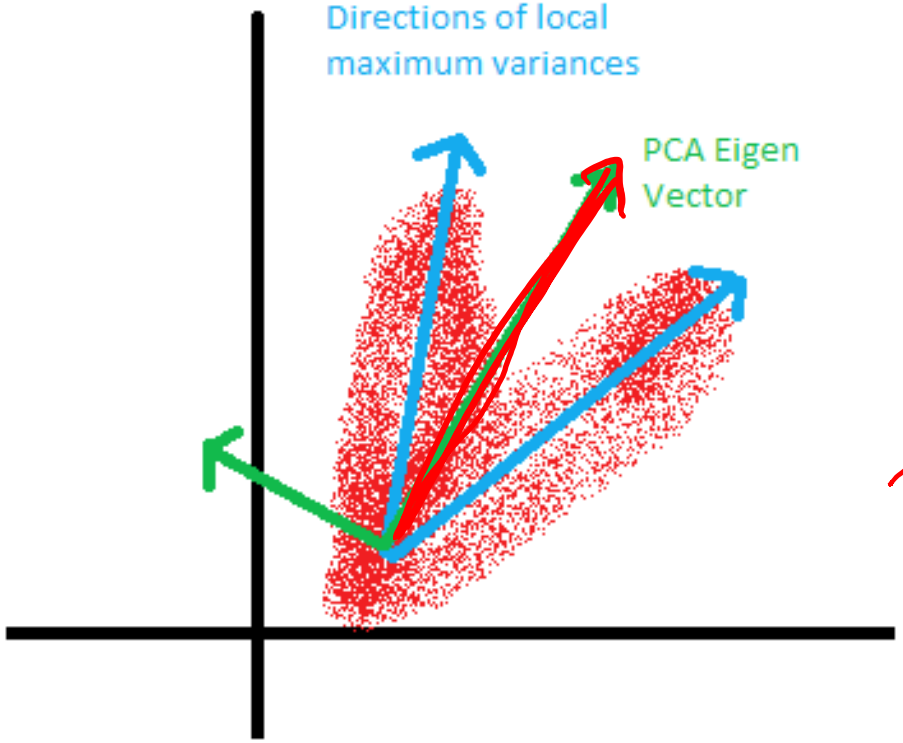
- PCA finds directions to project the data so that variance is maximized
- PCA **does not consider class labels**
- Variance maximization is not necessarily beneficial for classification
  - Where we want to maximize class separation

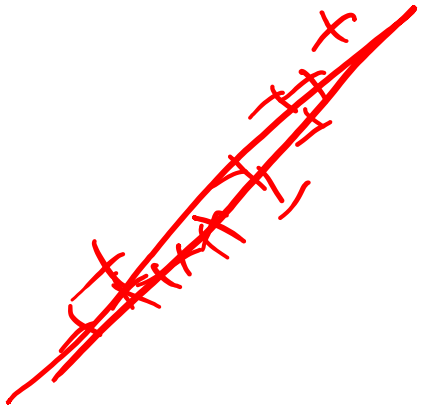


# When PCA fails



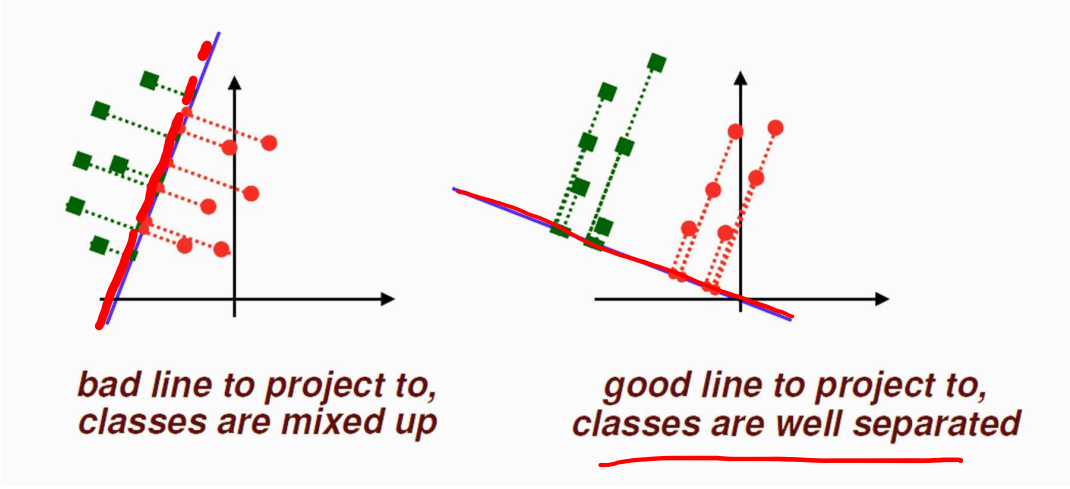
- https://cnl.salk.edu/~tewon/Blind/blind_audio.html



# Feature Projection by LDA

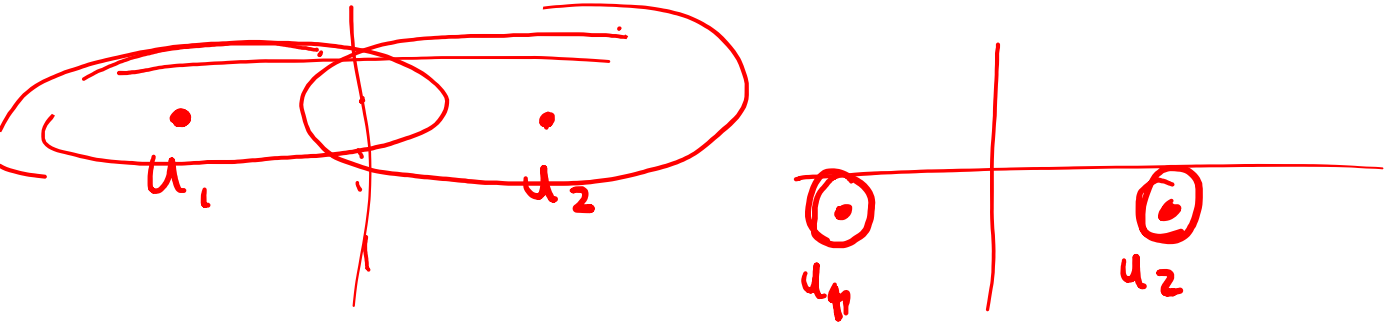
- Linear Discriminant Analysis (LDA) projects to a line which preserves direction useful for data classification
  - **Limitation**: If we have $C$ classes, then LDA returns a $(C-1)$ dimensional feature
    - So for binary classification, we'll just get one feature per observation.
- Main idea: find projection to a line such that samples from different classes are well separated



# Feature Projection by LDA

- Suppose we have 2 classes and $D$-dimensional data
  - Let $C_1$ be the set of samples from class 1
  - Let $C_2$ are the samples from class 2
- We want to find some projection matrix $W$

$\mu_1$ $\mu_2$ $J(w) = (\mu_1 w - \mu_2 w)^2$




$$J(w) = (\mu_1 w - \mu_2 w)^2 - \lambda\left((\sigma_1 w)^2 + (\sigma_2 w)^2\right)$$

$$(\mu_1 w - \mu_2 w)^T (\mu_1 w - \mu_2 w) - \lambda\left((\sigma_1 w)^T (\sigma_1 w) + (\sigma_2 w)^T (\sigma_2 w)\right)$$

# Feature Projection by LDA

- Suppose we have 2 classes and $D$-dimensional data
  - Let $C_1$ be the set of samples from class 1
  - Let $C_2$ are the samples from class 2
- We want to find some projection matrix $W$
- Let $\mu_1$ and $\mu_2$ be the mean of classes 1 and 2, respectively, **before** projection.
- Let $\sigma_1$ and $\sigma_2$ the standard deviation of classes 1 and 2, respectively, **before** projection.

# Feature Projection by LDA

- Idea: Find a projection that maximizes the difference in the means **after** projection:

$$J(w) = argmax_w(|\mu_1 w - \mu_2 w|)$$

- Issue: $|\mu_1 - \mu_2|$ is non differentiable!
  - So let's maximize $(\mu_1 w - \mu_2 w)^2$

$$J(w) = (\mu_1 w - \mu_2 w)^T (\mu_1 w - \mu_2 w)$$

- Issue: This doesn't take variance of the classes into account ☹
  - Let's add a penalty if the within-class variance is large:

$$J(w) = (\mu_1 w - \mu_2 w)^T (\mu_1 w - \mu_2 w) - \lambda\left((\sigma_1 w)^T (\sigma_1 w) + (\sigma_2 w)^T (\sigma_2 w)\right)$$

# Feature Projection by LDA

$$J(w) = (\mu_1 w - \mu_2 w)^T(\mu_1 w - \mu_2 w) - \lambda\big((\sigma_1 w)^T(\sigma_1 w) + (\sigma_2 w)^T(\sigma_2 w)\big)$$

- So we want to find the value(s) of $w$ that maximize this.
- To make this easier, let's clean up some stuff:

$$J(w) = w^T(\mu_1 - \mu_2)^T(\mu_1 - \mu_2)w - \lambda(w^T(\sigma_1^T\sigma_1 + \sigma_2^T\sigma_2)w)$$

$A = S_B$ $\quad B = S_W$

$$J(w) = w^T A w - \lambda w^T B w$$

$$\frac{\partial J}{\partial w} = 2Aw - 2\lambda Bw = 0$$

$$2Aw = 2\lambda Bw$$

$$Aw = \lambda Bw$$

$$B^{-1}Aw = \lambda B^{-1}Bw$$

$$(B^{-1}A)w = \lambda w$$

$$Mw = \lambda w$$

# Feature Projection by LDA

$$J(w) = (\mu_1 w - \mu_2 w)^T(\mu_1 w - \mu_2 w) - \lambda\big((\sigma_1 w)^T(\sigma_1 w) + (\sigma_2 w)^T(\sigma_2 w)\big)$$

- So we want to find the value(s) of $w$ that maximize this.
- To make this easier, let's clean up some stuff:

$$J(w) = w^T(\mu_1 - \mu_2)^T(\mu_1 - \mu_2)w - \lambda(w^T(\sigma_1^T\sigma_1 + \sigma_2^T\sigma_2)w)$$

- We can now do some substitution to make this more manageable
  - $\sigma_1^2 = \sigma_1^T\sigma_1$ is called the **scatter matrix** for class 1 is defined as

$$\sigma_1^2 = \sum_{x_i \in C_1} (x_i - \mu_1)^T(x_i - \mu_1) = (|C_1| - 1)cov(C_1)$$

  - Let the **within class scatter matrix** be defined as

$$S_W = \sigma_1^2 + \sigma_2^2$$

  - And let the **between class scatter matrix** be defined as

$$S_B = (\mu_1 - \mu_2)^T(\mu_1 - \mu_2)$$

- We can now write $J(w)$ as

$$J(w) = w^T S_B w - \lambda w^T S_W w$$

# Feature Projection by LDA

$$J(w) = w^T S_B w - \lambda w^T S_W w$$

- Taking the derivative with respect to $w$ and setting it equal to zero we get:

$$2S_B w - 2\lambda S_w w = 0$$

- We can re-arrange as:

$$(S_B - \lambda S_w)w = 0$$

- Or

$$S_W^{-1} S_B w = \lambda w$$

- This is another eigen-decomposition problem!!!

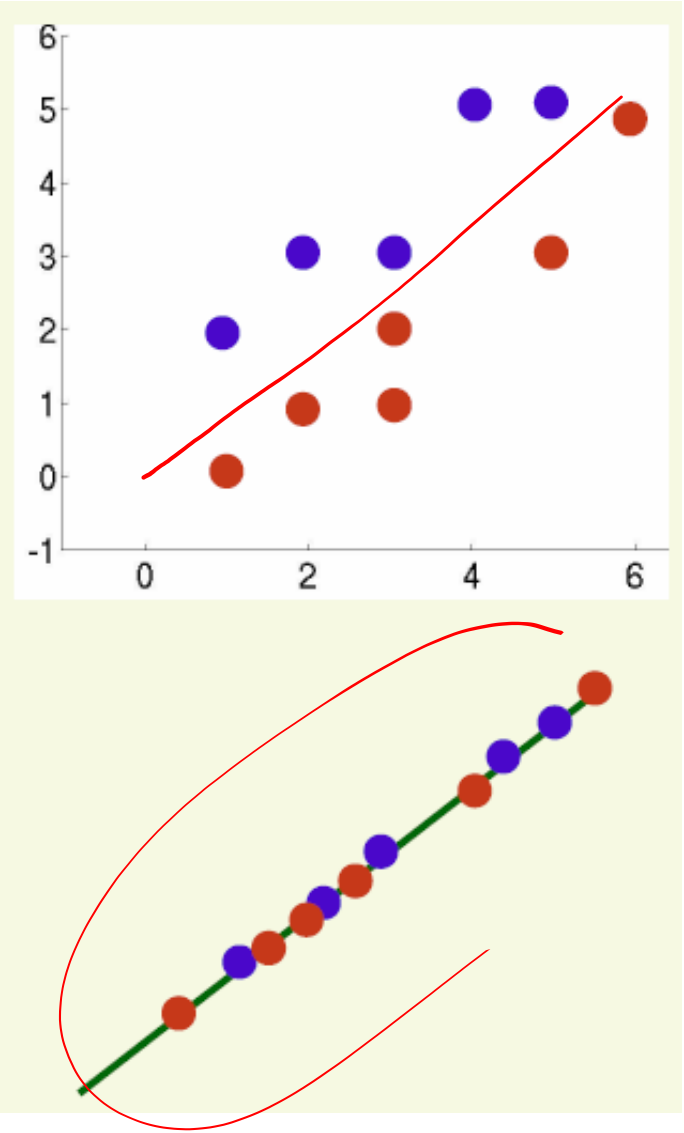
# Feature Extraction by LDA

$$S_W^{-1} S_B W = \lambda W$$

- $S_W^{-1} S_B$ has only one non-zero eigenvalue (and therefore one eigenvector)
- Use that eigenvector as your projection matrix

# Example: LDA

- Data:
  - Class 1 has samples $C_1$={(1,2),(2,3),(3,3),(4,5),(5,5)}
  - Class 2 has samples $C_2$={(1,0),(2,1),(3,1),(3,2),(5,3),(6,5)}
- If we did PCA and projected the points onto the “best line” we would get poor separation



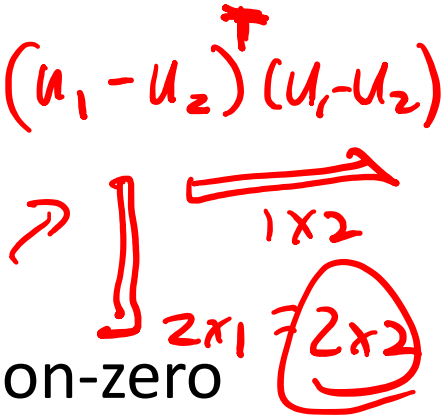
# Example: LDA

1. First standardize all the data
2. Next compute the means for each class
   - $\mu_1 = [-0.1094, 0.5023]$, $\mu_2 = [0.0911, -0.4186]$
3. Then compute the scatter matrices for each class
   - $\sigma_1^2 = (5-1) * cov(C_1) = \begin{bmatrix} 3.62 & 2.77 \\ 2.77 & 2.39 \end{bmatrix}$
   - $\sigma_2^2 = (6-1) * cov(C_2) = \begin{bmatrix} 6.27 & 5.54 \\ 5.54 & 5.30 \end{bmatrix}$

# Example: LDA

4. Followed by the within class scatter matrix
   - $S_W = \sigma_1^2 + \sigma_2^2 = \begin{bmatrix} 9.89 & 8.31 \\ 8.31 & 7.69 \end{bmatrix}$
   - $S_W^{-1} = \begin{bmatrix} 1.10 & -1.19 \\ -1.19 & 1.42 \end{bmatrix}$
     - Matlab also has `inv(sw)`

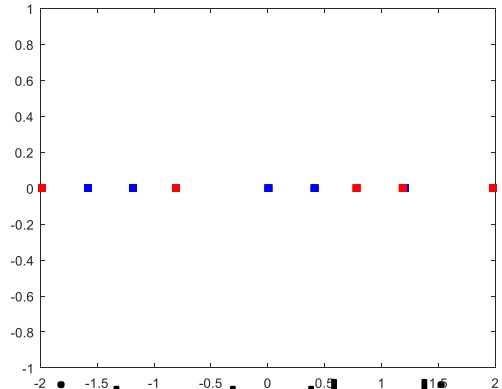
   Remember how to do this?




5. Perform eigen-decomposition on $S_W^{-1} S_B$
6. The eigenvector pertaining to the only non-zero eigenvalue is our projection matrix:
7. There is only one non-zero eigen-value so its corresponding eigen-vector becomes our direction of projection:

$$w = \begin{bmatrix} -0.6491 \\ 0.7607 \end{bmatrix}$$

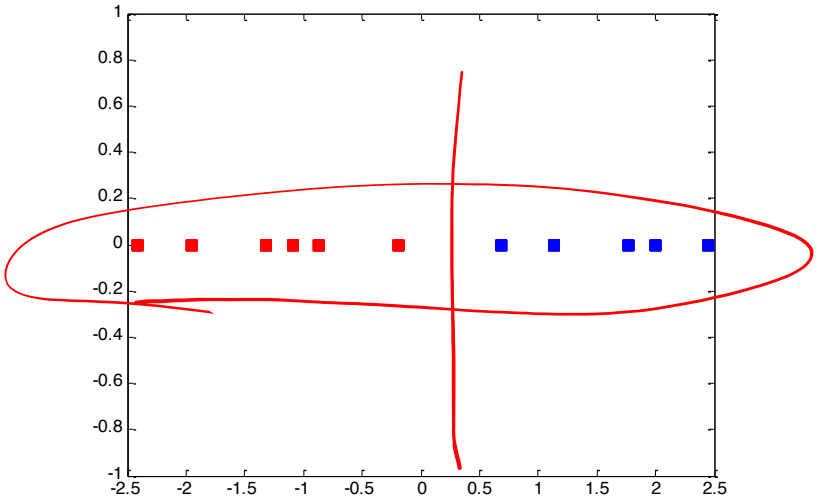
# Example LDA

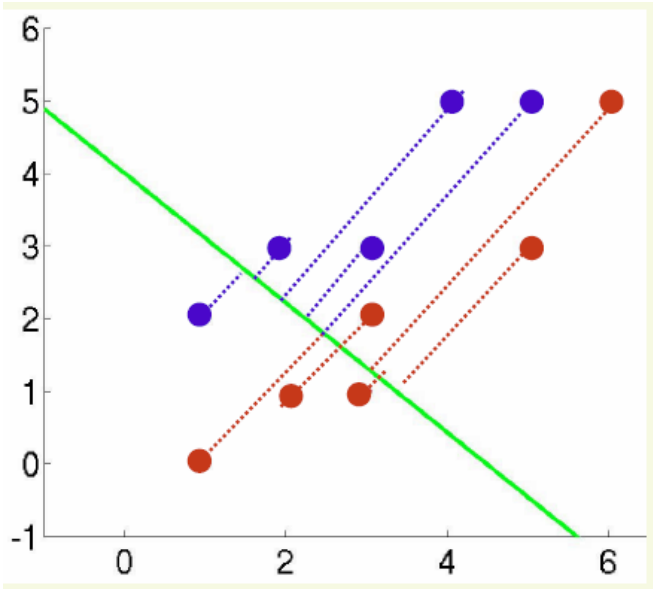



- Now we just project each point onto the line(s)

$$Z = XW$$

- Binary Classification reduces feature space to 1-D
  - Is that good enough?



```
#LDA
c = [[1,2],[2,3],[3,3],[4,5],[5,5],[1,0],[2,1],[3,1],[3,2],[5,3],[6,5]]
c = np.array(c)
mean = np.mean(c,axis=0)
std = np.std(c,axis=0,ddof=1)
print(mean)
print(std)
c = (c-mean)/std
print(c)
```

```
[3.18181818 2.72727273]
[1.66241883 1.73729152]
[[-1.31243592 -0.41862446]
 [-0.71090279  0.15698417]
 [-0.10936966  0.15698417]
 [ 0.49216347  1.30820144]
 [ 1.0936966   1.30820144]
 [-1.31243592 -1.56984173]
 [-0.71090279 -0.9942331 ]
 [-0.10936966 -0.9942331 ]
 [-0.10936966 -0.41862446]
 [ 1.0936966   0.15698417]
 [ 1.69522973  1.30820144]]
```

```
c1mean = np.mean(c[:5,:],axis=0)
c1 = c[:5,:]-c1mean
print(c1)
```

```
[[-1.20306626e+00 -9.20973817e-01]
 [-6.01533129e-01 -3.45365181e-01]
 [-4.16333634e-17 -3.45365181e-01]
 [ 6.01533129e-01  8.05852090e-01]
 [ 1.20306626e+00  8.05852090e-01]]
```

```
c2mean = np.mean(c[5:,:],axis=0)
c2 = c[5:,:]-c2mean
print(c2)
```

```
[[-1.4035773  -1.15121727]
 [-0.80204417 -0.57560864]
 [-0.20051104 -0.57560864]
 [-0.20051104  0.        ]
 [ 1.00255521  0.57560864]
 [ 1.60408834  1.72682591]]
```

```
covc1 = 4*(c1.T@c1 / 4)
print(covc1)
```

```
[[3.61842105 2.76998131]
 [2.76998131 2.38554217]]
```

```
covc2 = 5*(c2.T@c2 / 5)
print(covc2)
```

```
[[6.27192982 5.53996262]
 [5.53996262 5.30120482]]
```

```
SW = covc1 + covc2
print(SW)
```

```
[[9.89035088 8.30994393]
 [8.30994393 7.68674699]]
```

```
c2meanvec = c2mean.reshape(1,2)
c1meanvec = c1mean.reshape(2,1)
SB = c1meanvec @ c2meanvec
print(SB)
```

```
[[-0.0099681   0.04578482]
 [ 0.04578482 -0.21029573]]
```

```
SWinv = np.linalg.inv(SW)
A = SWinv @ SB
E, V = np.linalg.eig(A)
print(E)
print(V)
```

$S_W^{-1} S_B x = \lambda x$

```
[ 5.55111512e-17 -4.18606414e-01]
[[ 0.97711039 -0.64911357]
 [ 0.21273289  0.76069151]]
```

0 1

```
print(V[:,1])
```

```
[-0.64911357  0.76069151]
```

```
print(c1mean @ V[:,1])
print(c2mean @ V[:,1])
```

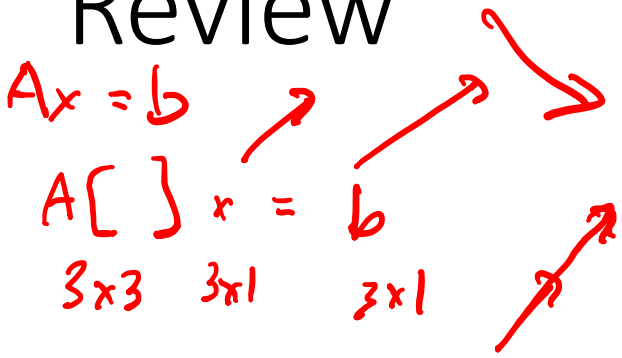
```
0.45312621828027333
-0.3776051819002279
```

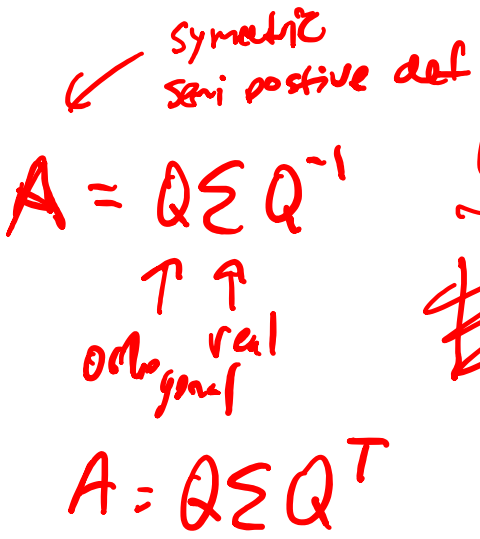
0

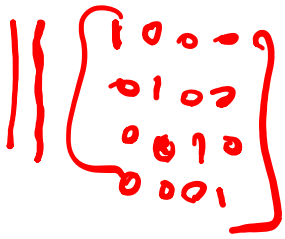
# Multi-Class LDA

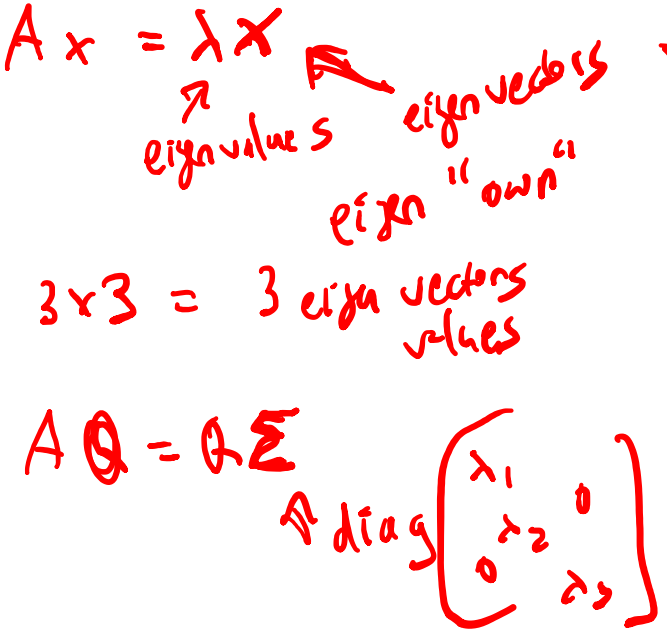
- To do multi-class LDA ($C > 2$) we just need to make a few changes:
  - $S_W = \sum_{i=1}^{C}(|C_i| - 1)cov(C_i)$
  - $S_B = \sum_{i=1}^{C}|C_i|(\mu_i - \mu)^T(\mu_i - \mu)$
    - Where $\mu$ is the mean of **al**l data
  - Solve the eigenvalue problem and choose which eigenvectors to use based on the largest eigenvalues
    - There will be at most $C - 1$ non-zero ones.

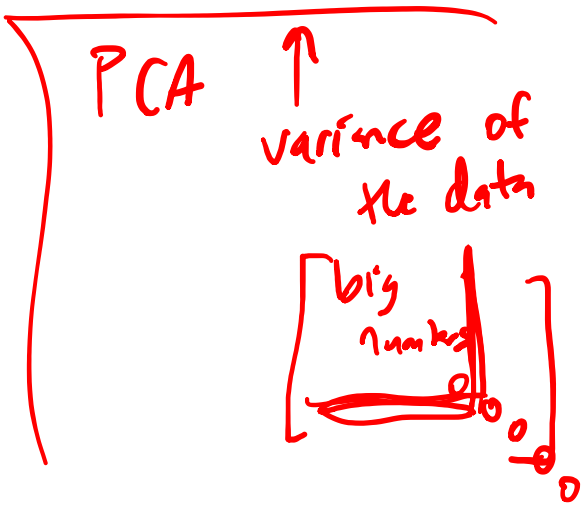
# Review


Ax = b
A[ ] x = b
3x3 3x1 3x1


symmetric
semi positive def
A = QΣQ⁻¹
Q⁻¹ = Qᵀ
orthogonal
real
A = QΣQᵀ




Ax = λx
eigenvalues
eigenvectors
eigen "own"
3x3 = 3 eigen vectors values
AQ = QΣ
diag
AQQ⁻¹ = QΣQ⁻¹
I


PCA
variance of the data
big numbers

# SVD

singular value decomposition

square $A$

$AV = U\Sigma$

$V$: row space of $A$; $U$: column space

$V = U = Q$

$A = U\Sigma V^T$ — $U, V$'s?

$A^TA = (U\Sigma V^T)^T(U\Sigma V^T)$

$V\Sigma^T U^T U \Sigma V^T$

$U^TU = I$

$V\Sigma^T\Sigma V^T$

eigen decomp of $A^TA$

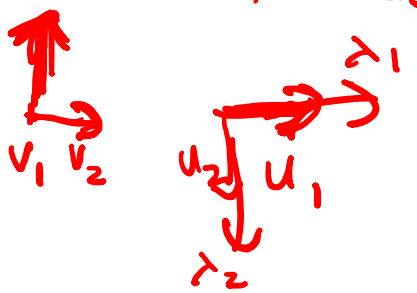
$V \begin{bmatrix} \lambda_1^2 & & \\ & \lambda_2^2 & \\ & & \lambda_3^2 \end{bmatrix} V^T = A^TA$

$AA^T = U\Sigma^T\Sigma U^T$

$X = \begin{bmatrix} 5 & 3 \\ 3 & 7 \\ 1 & 1 \end{bmatrix}$

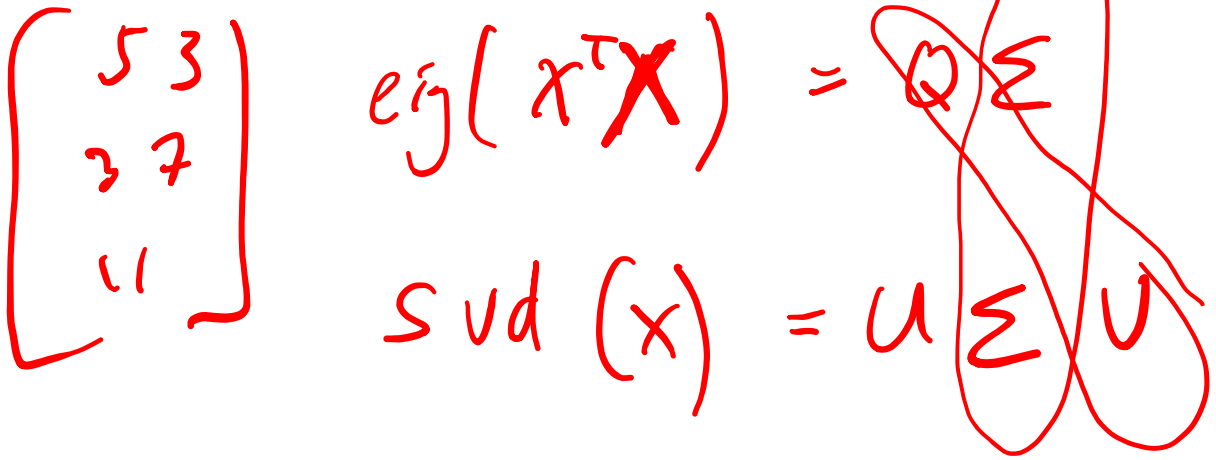
$X^TX = Q\Sigma Q^T$

$AQ = Q\Sigma$



# SVD

X = [ 5 3
3 7
1 1]




Matrix, what to expect out of eig, svd?

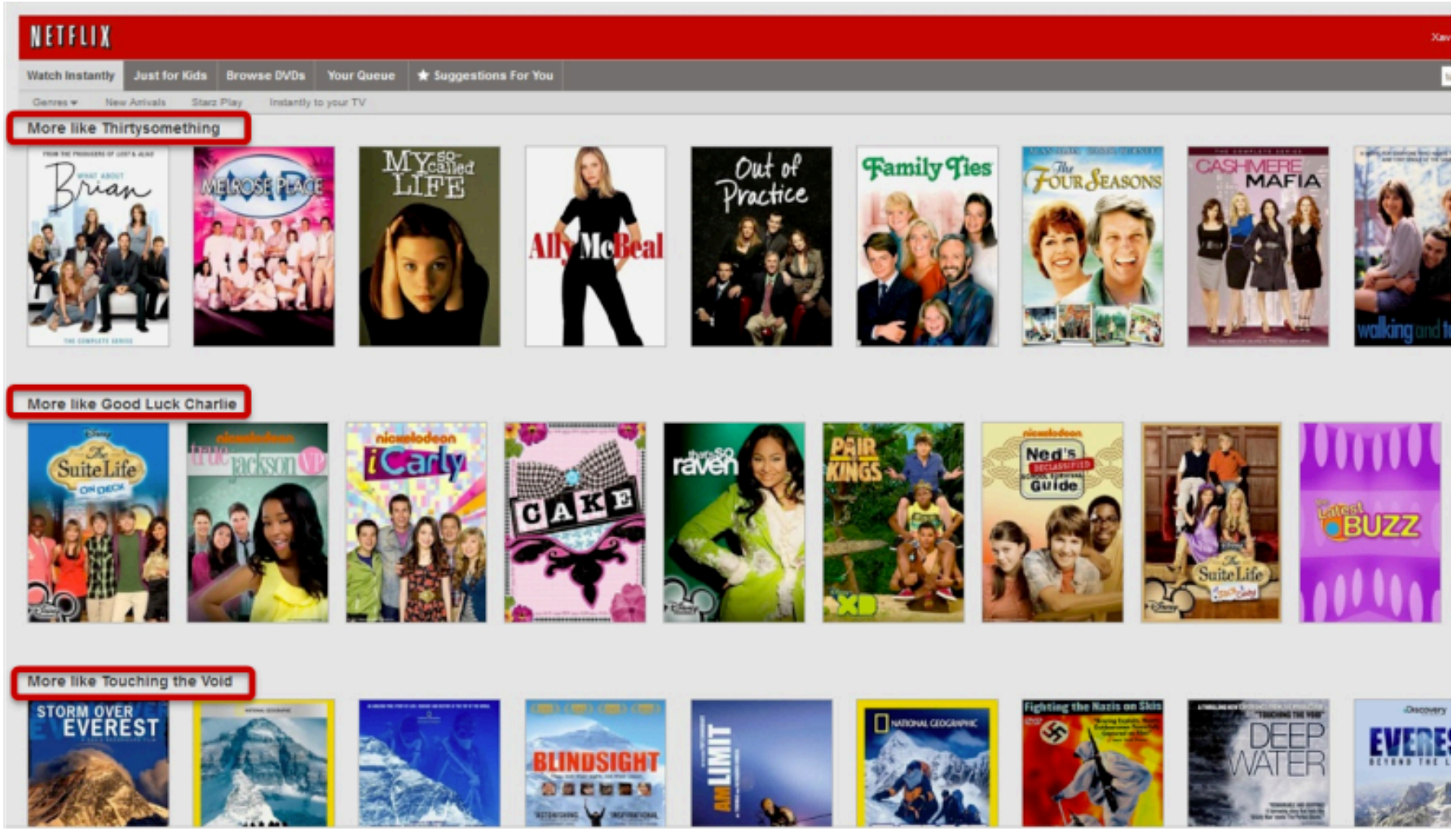
In [7]:
```
a = [[5,3],[3,7],[1,1]]
a = np.array(a)
cov = a.T @ a
e,v = np.linalg.eig(cov)
print(e)
print(v)
```

```
[ 8.10269932 85.89730068]
[[-0.80885867 -0.5880031 ]
 [ 0.5880031  -0.80885867]]
```

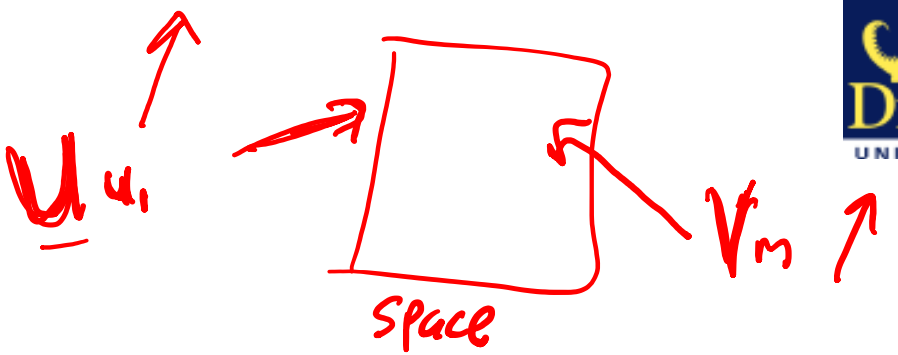
In [10]:
```
a = [[5,3],[3,7],[1,1]]
a = np.array(a)
u,e,v = np.linalg.svd(a)
print(u)
print(e**2)
print(v)
```

```
[[-0.57904029  0.80107667 -0.15161961]
 [-0.80124689 -0.59351183 -0.0758098 ]
 [-0.1507175   0.07758781  0.98552746]]
[85.89730068  8.10269932]
[[-0.5880031  -0.80885867]
 [ 0.80885867 -0.5880031 ]]
```

# SVD Recommender Systems



# SVD



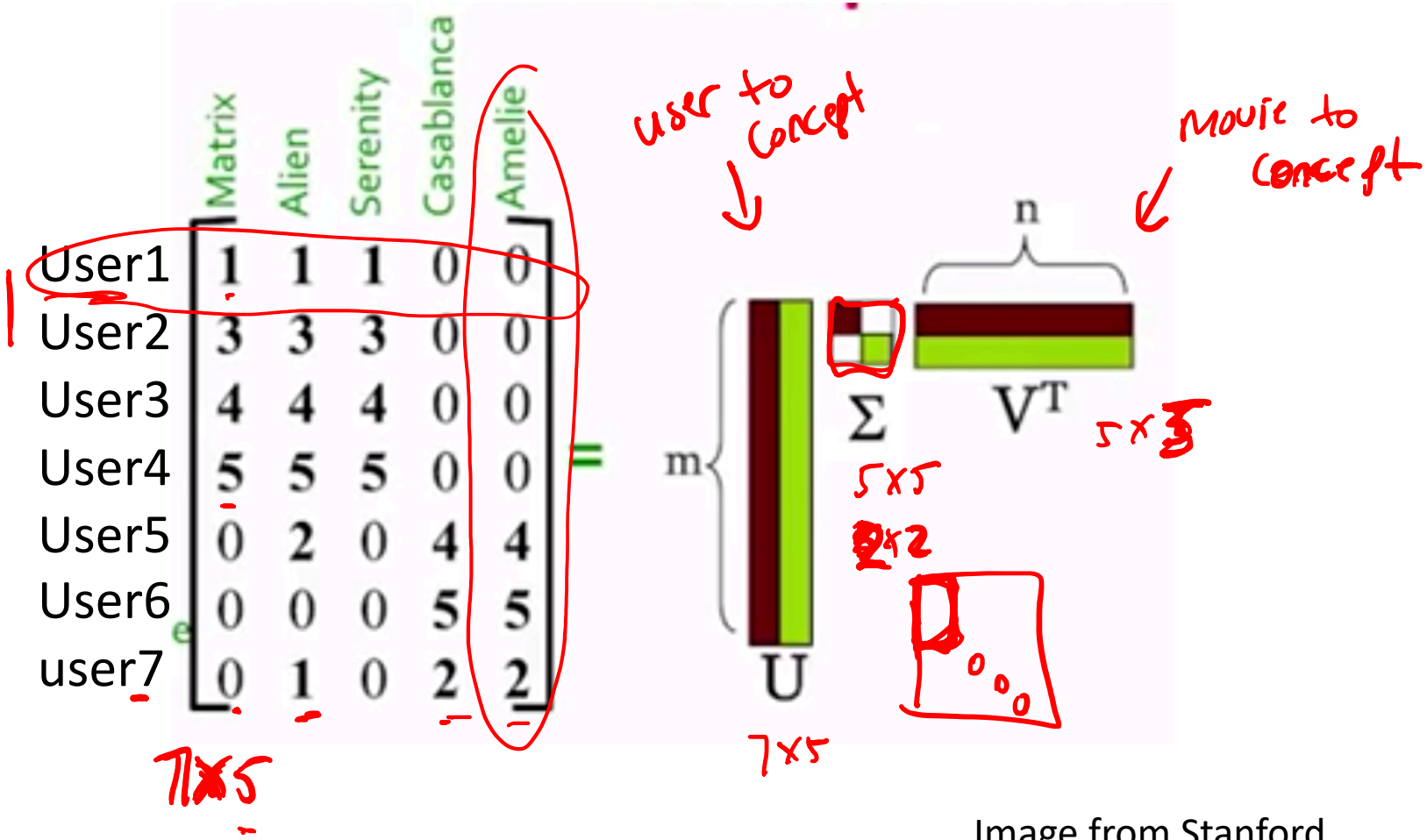




Image from Stanford

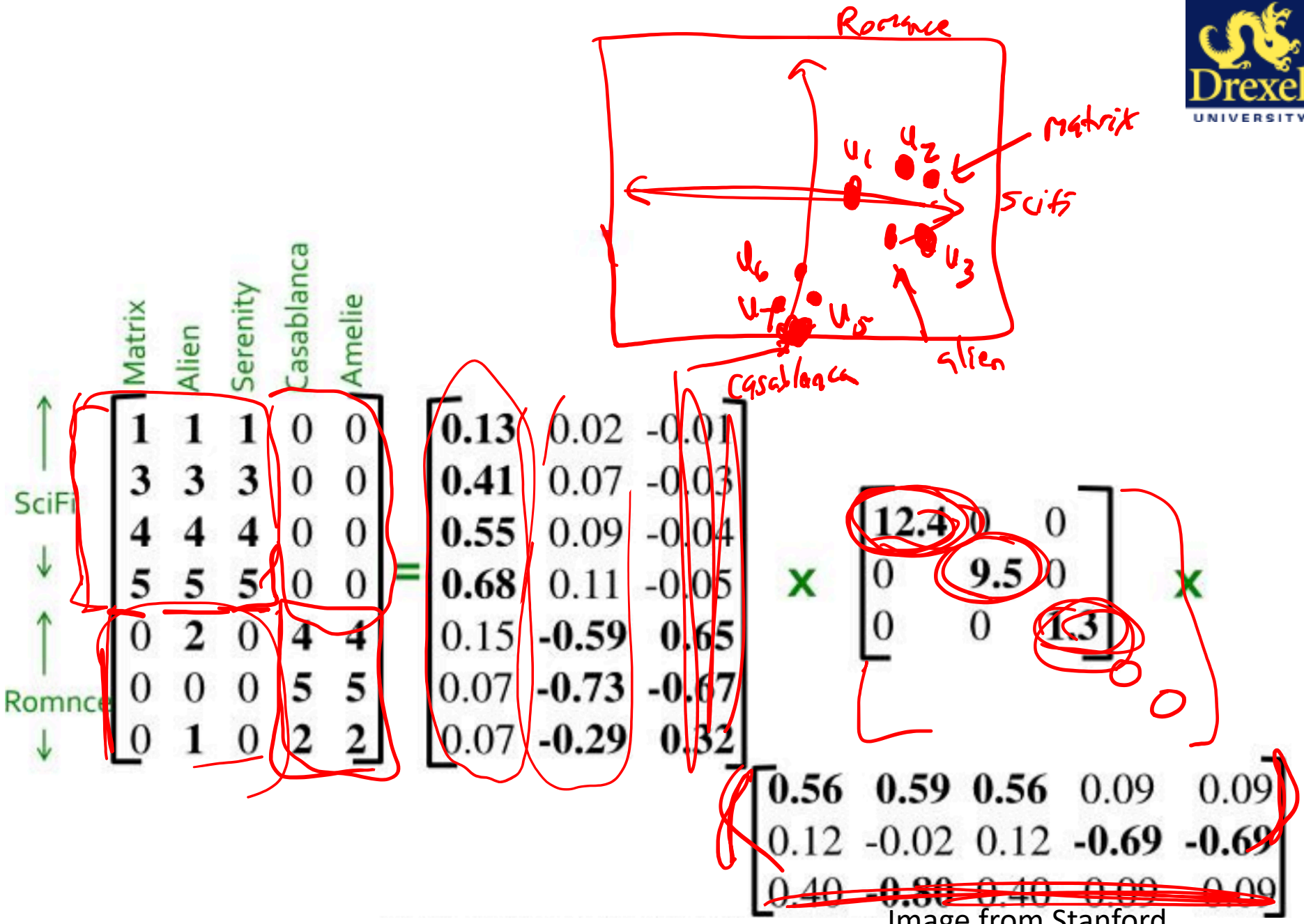



Image from Stanford

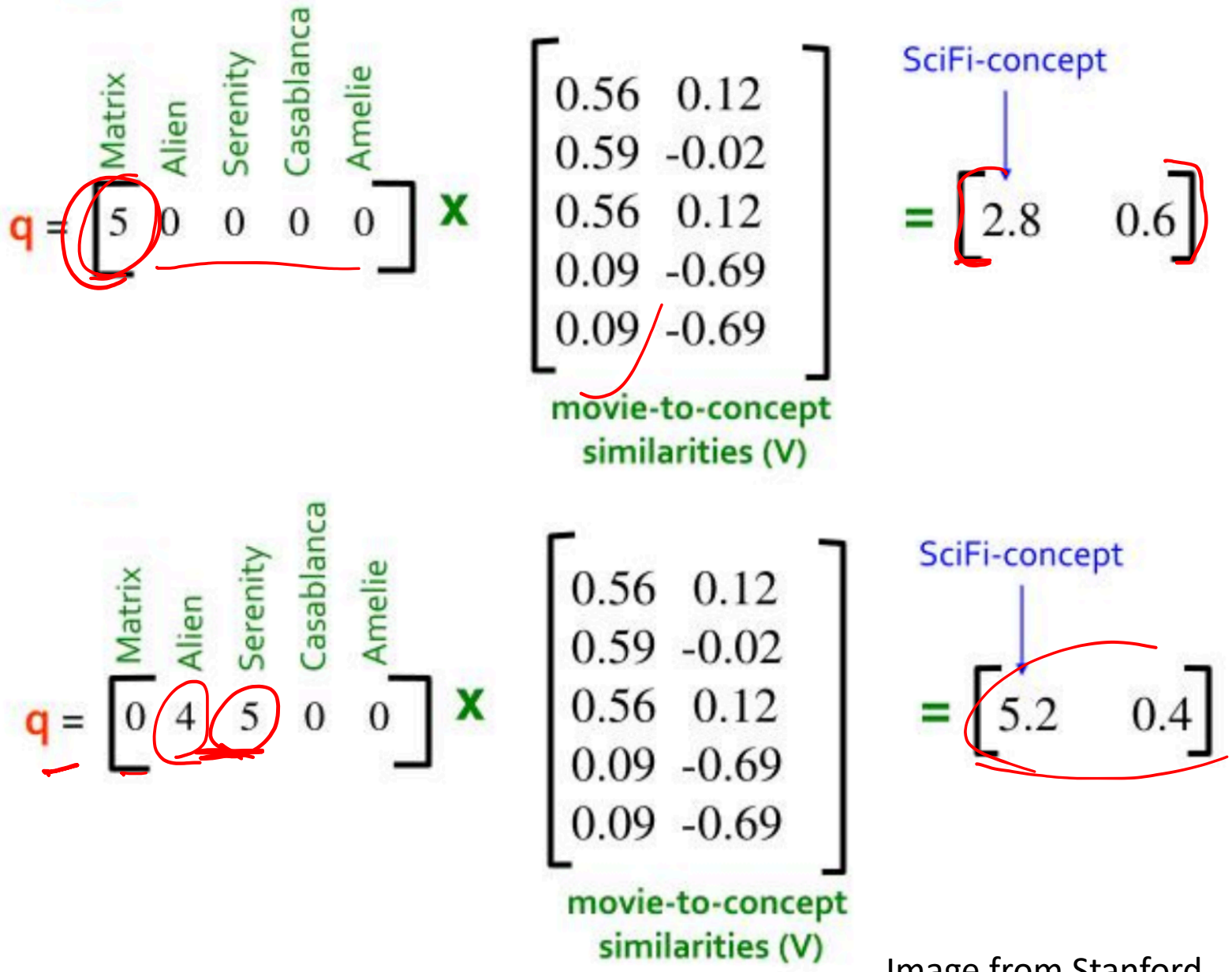



Image from Stanford

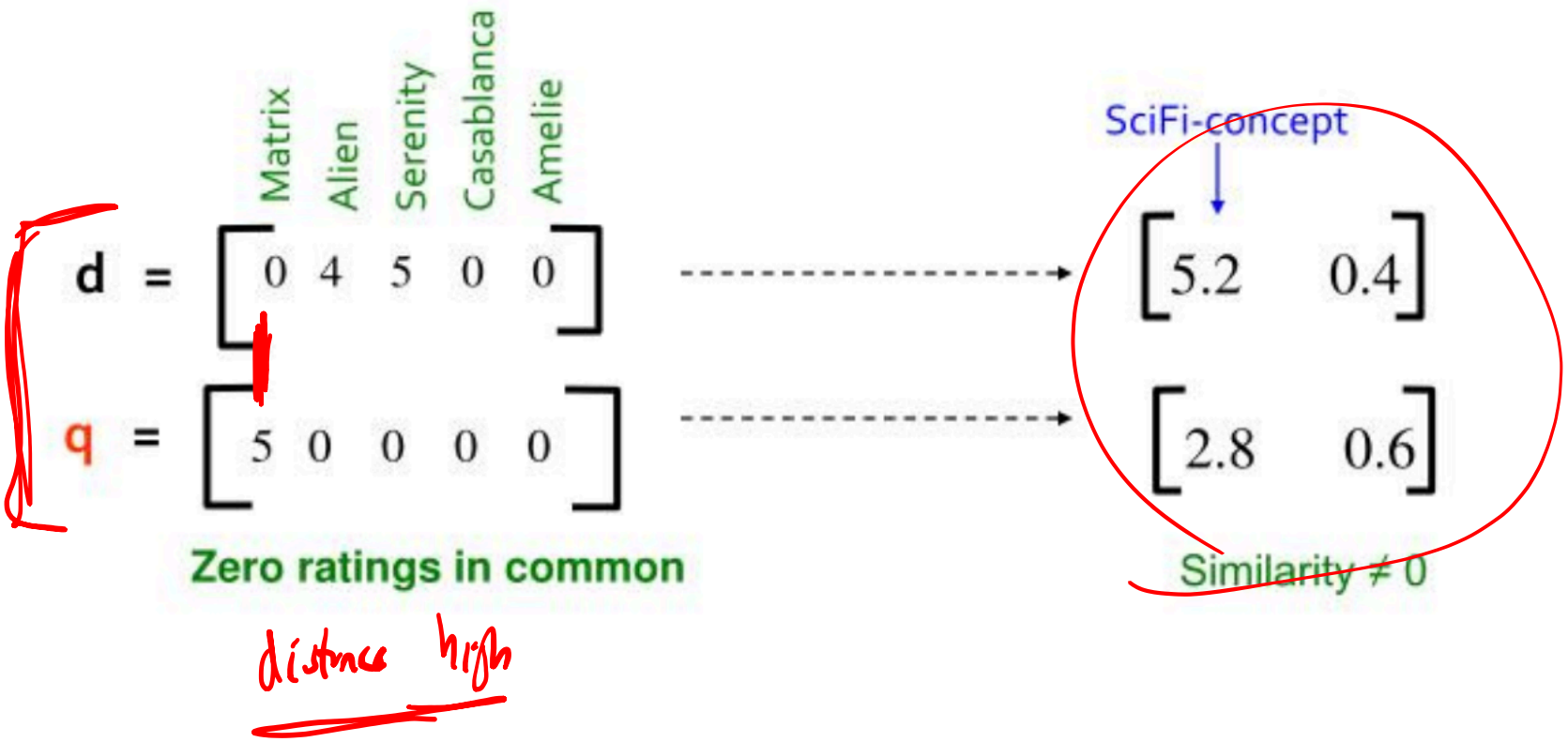


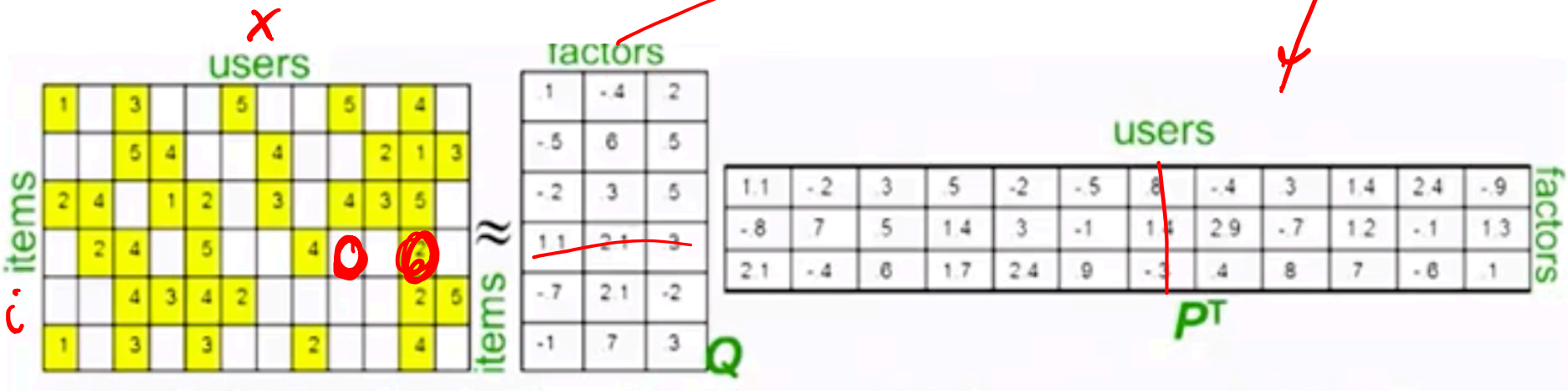

Image from Stanford

# Modify SVD

$R = Q P^T$

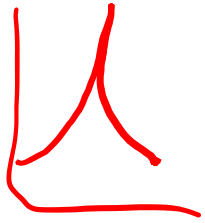
$U \Sigma V^T$

$\sum_{i,x \in R} \left( r_{xi} - q_i p_x^T \right)^2$



# ICA

independent component analysis



independent → $s_1$ $s_2$ —mix→ $x_1$ $x_2$

$x_1 = a_{11} s_1 + a_{12} s_2$

$x_2 = a_{21} s_1 + a_{22} s_2$

mixing matrix

$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = X = A s$

$A^{-1} A s = A^{-1} x$

$s = A^{-1} x$

$(U \Sigma V^T)^+$

$s = V \Sigma^{-1} U^T x$

covariance $x$

$(As)(As)^T$

$(U \Sigma V^T s)(U \Sigma V^T s)^T$

$U \Sigma V^T s s^T V \Sigma^T U^T$

$s s^T \sim I$

$U \Sigma^2 U^T$ ← eigen decomp