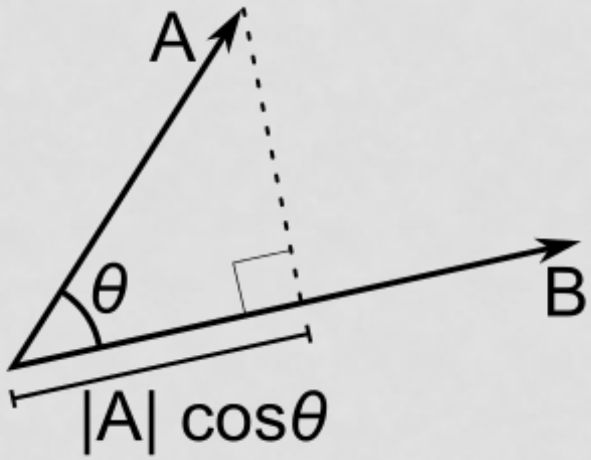
ESSENTIAL MATH FOR DATA SCIENCE

DAY 2

DOT PRODUCT

Intuition:

- directional multiplication
- applies the directional growth of one vector another
- result is how much stronger we've made the original

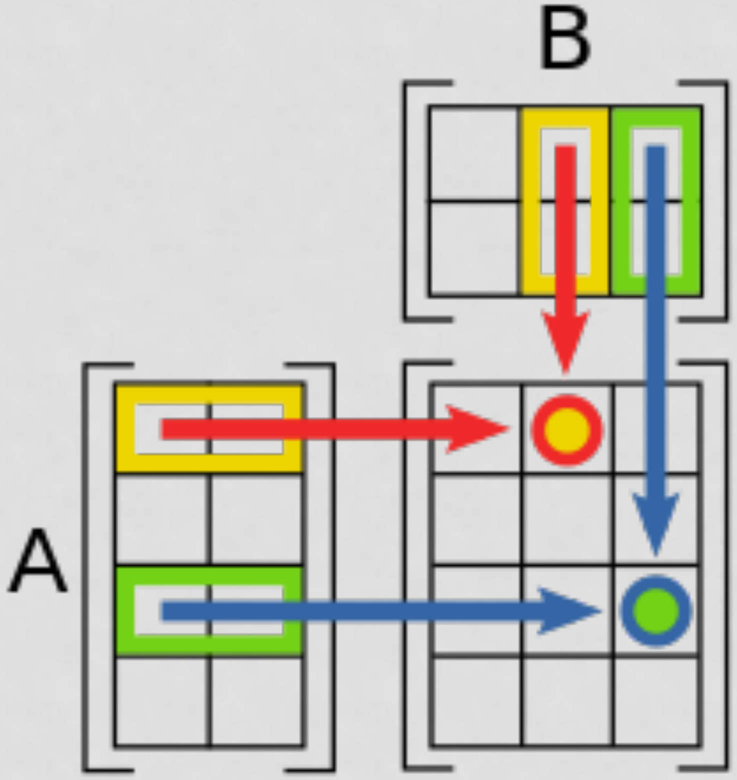


$$\vec{x} \cdot \vec{y} = \|\vec{x}\|\|\vec{y}\| \cos \theta$$

$$\vec{x} \cdot \vec{y} = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

```
>>> np.dot(np.array([[1,2,3]]),np.array([[2,3,4]]))
```

MATRIX MULTIPLICATION



- dot product between rows of first matrix and columns of second
- length of each row (i.e. num of columns) in A must equal length of each column (i.e. num of rows) in B

MATRIX TRANSPOSE

- A matrix transpose is an operation that takes an *mxn* matrix and turns it into an *nxm* matrix
- the rows of the original matrix become the columns of the resulting matrix and vice versa

$$A = \begin{bmatrix} 2 \\ 8 \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} 2 & 8 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 1 \\ 2 & 4 \\ 1 & -1 \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} 0 & 2 & 1 \\ 1 & 4 & -1 \end{bmatrix}$$

```
>>> x = np.array([[3,4,5,6]]).T
```

MATRIX INVERSE

- because there is no such thing as matrix division
- same idea as the reciprocal of a number
- note: not every matrix is invertible
  - must be a square matrix
  - determinant != 0

$$8 \cdot \frac{1}{8} = 1 \quad \rightarrow \quad A \cdot A^{-1} = I$$

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ in } \mathbb{R}^3$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{\underbrace{ad-bc}_{\text{determinant}}} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

```
>>> invA = np.linalg.inv(A)
```

PUT IT ALL TOGETHER…

Imagine yourself as a crew member on a coast guard boat, looking for evildoers. Periodically, your boat radios its position to headquarters. You expect that communication to be intercepted so you have to transform the actual position of the boat. Calculate the encoded coordinates. Use:

$$\vec{y} = A\vec{x}$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{matrix} \text{for Western longitude} \\ \text{for Northern latitude} \end{matrix} = \begin{bmatrix} 5 \\ 42 \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 & + & 3x_2 \\ 2x_1 & + & 5x_2 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = ?$$

PUT IT ALL TOGETHER…

Imagine yourself as a member of the coast guard headquarters. Periodically, crew members on a boat radio in their position. They expect that communication to be intercepted so they have transformed the actual position of the boat. How do you get their true location? Use:

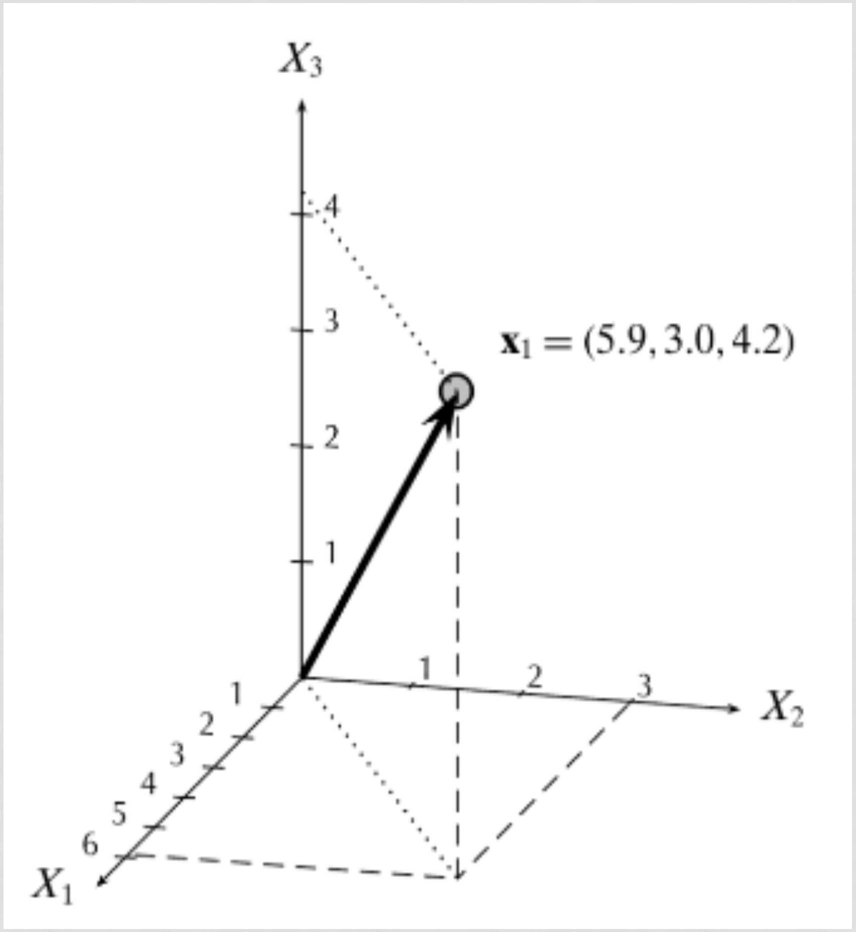
$$\vec{x} = B\vec{y} \qquad B = A^{-1} \qquad B = \begin{bmatrix} -5 & 3 \\ 2 & -1 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -5 & 3 \\ 2 & -1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} -5 & 3 \\ 2 & -1 \end{bmatrix} \begin{bmatrix} 131 \\ 220 \end{bmatrix} = ?$$

SYSTEMS OF EQUATIONS BREAKOUT!!

IT'S YOUR TURN

NORMS



- vectors are composed of both direction and magnitude
- Frobenius norm (aka Euclidean norm) is the most commonly known

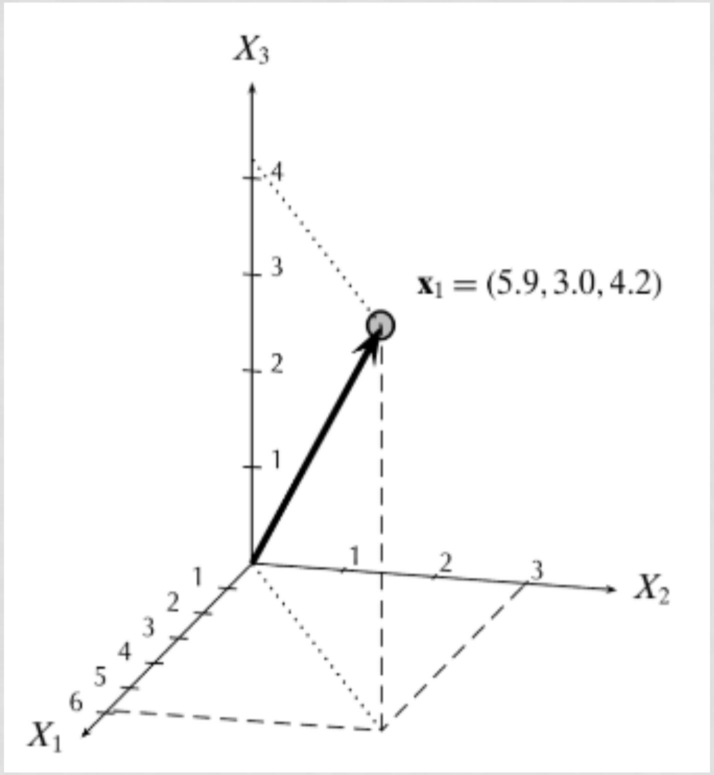
$$\|x\| = \sqrt{{x_1}^2 + {x_2}^2 + \cdots + {x_n}^2}$$

- More generally,

$$\|x\|_p = \left(\sum_i |x_i|^p \right)^{1/p}$$

```
>>> print(np.linalg.norm(x))
```

NORMS



- Note:

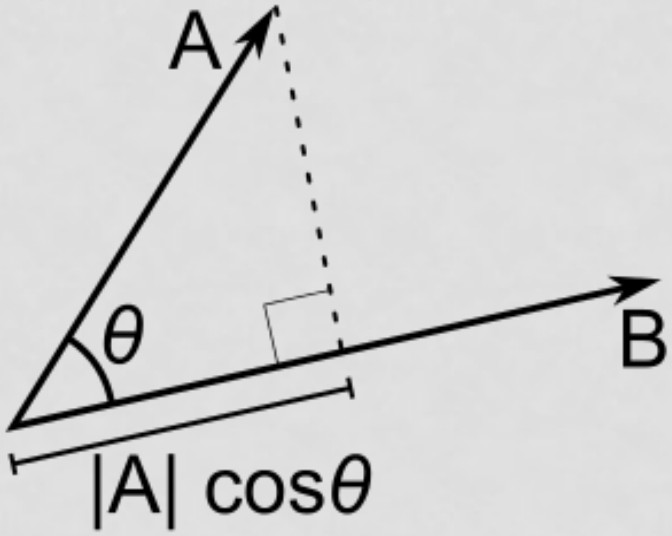
$$\|x\|^2 = x \cdot x$$

- Additionally, the norm between two vectors is the same as the distance btw two points

$$\|x - y\| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}$$

```
>>> np.linalg.norm(x-y)
```

COSINE SIMILARITY



- cosine of the angle between two vectors

$$\cos\theta = \frac{x \cdot y}{\|x\|\,\|y\|}$$

- if **x** and **y** are both zero-centered, then cosine similarity is the correlation between **x** and **y**

```
>>> np.dot(x,y)/(np.linalg.norm(x)*np.linalg.norm(y))
```

PROPERTIES OF MATRICES

- If X and Y are both $n \times p$ *matrices, then*

  $$X + Y = Y + X$$

- If X, Y, and Z are all $n \times p$ matrices, then

  $$X + (Y + Z) = (X + Y) + Z$$

- If X, Y, and Z are all conformable, then

  $$X(YZ) = (XY)Z$$

- If X is of dimension $n \times k$, and Y and Z are of dimension $k \times p$, then

  $$X(Y + Z) = XY + XZ$$

- If X is of dimension $p \times n$, and Y and Z are of dimension $k \times p$, then

  $$(Y + Z)X = YX + ZX$$

- If a and b are real numbers, and X is an $n \times p$ matrix,

  $$(a + b)X = aX + bX$$

- If a is a real number, and X and Y are both $n \times p$,

  $$a(X + Y) = aX + aY$$

- if a is a real number, and X and Y are conformable,

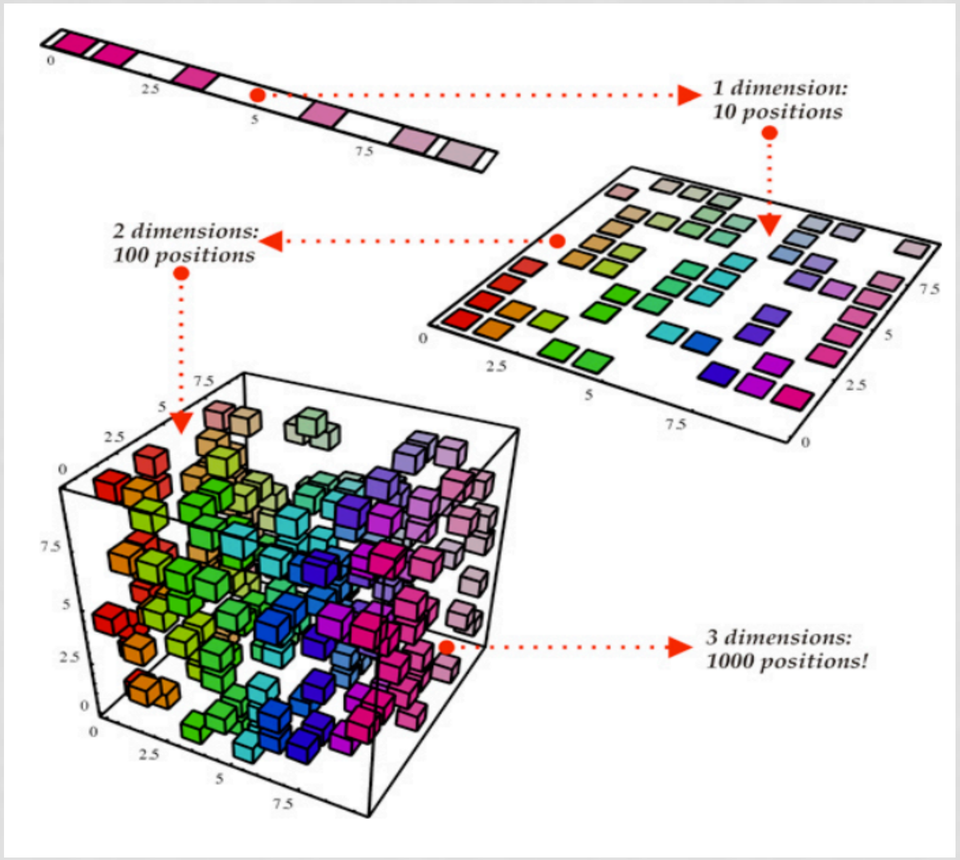
  $$X(aY) = a(XY)$$

BREAK

GET UP AND MOVE

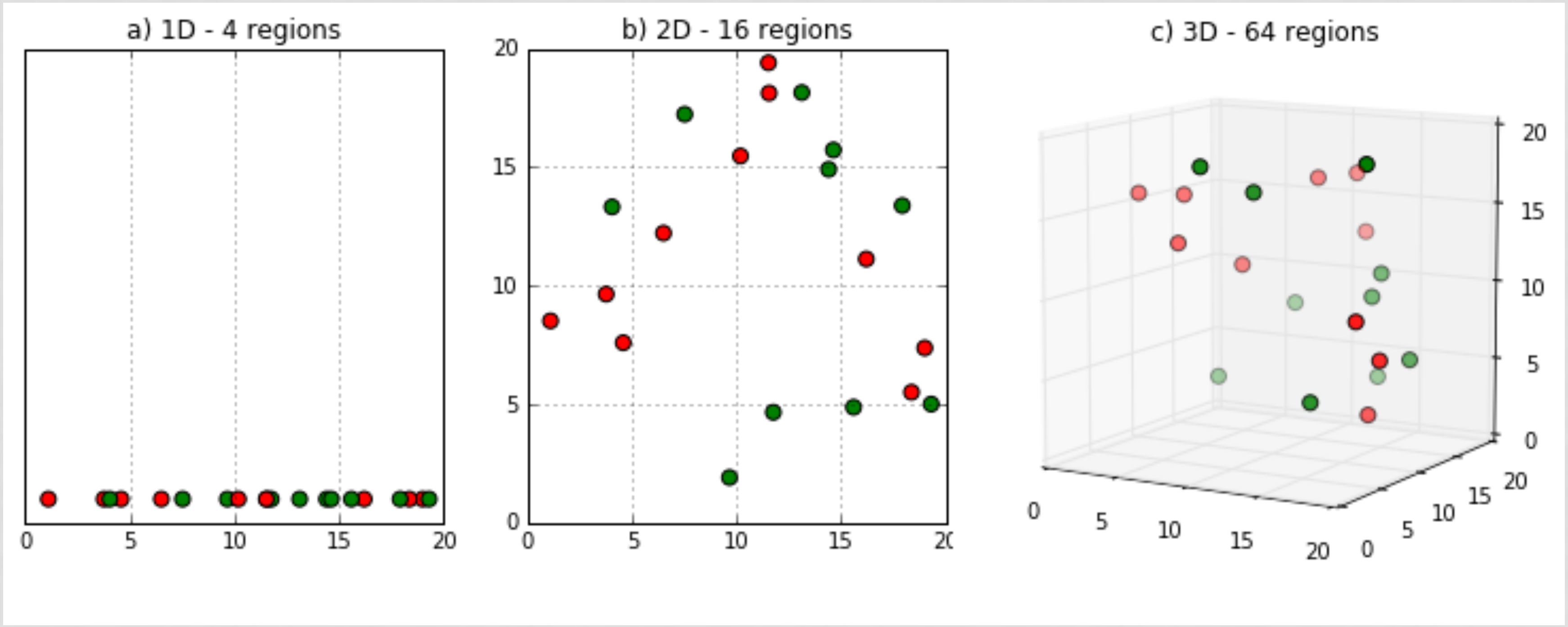
DIMENSIONALITY REDUCTION

- task of reducing the number of inputs
- preprocessing task
- why do we need it?
  - reduce computational cost
  - removes multicollinearity
  - make dataset easier to use
  - remove noise and redundant features
  - make results easier to understand/visualize



CURSE OF DIMENSIONALITY

The exponential growth of data that causes high sparsity with each increasing dimension.



MATRIX DECOMPOSITION

Numerical Factorization

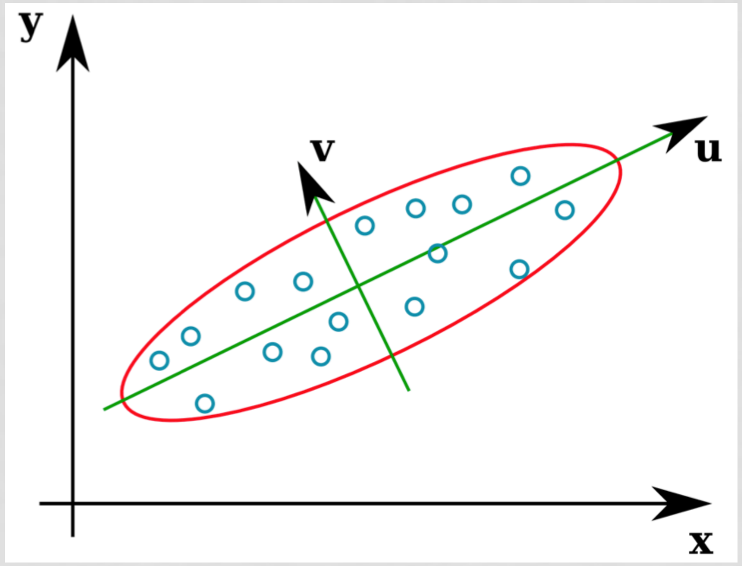
break a number up into its prime factors and their exponents

$$6,534 = 2^1 \cdot 3^3 \cdot 11^2$$

$$36,288 = 2^6 \cdot 3^4 \cdot 7^1$$

Matrix Factorization

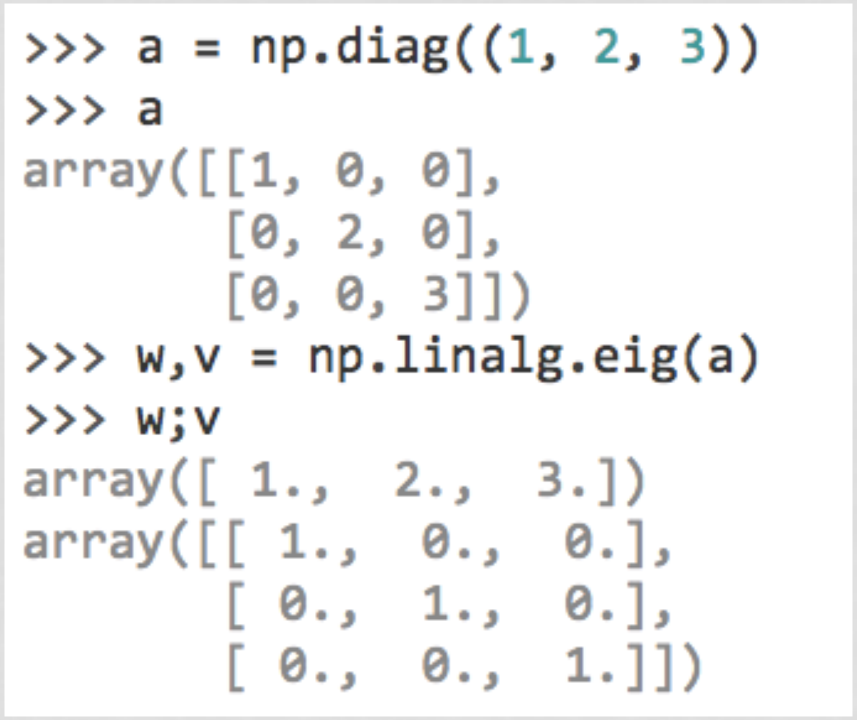
break a matrix up into its eigenvectors and their eigenvalues



EIGENVECTOR AND EIGENVALUES

- Let A be an *nxn* matrix and **x** be an *n*x1 nonzero vector
- An **eigenvalue** of A is a number λ such that

  $$A\mathbf{x} = \lambda \mathbf{x}$$



```
>>> a = np.diag((1, 2, 3))
>>> a
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
>>> w,v = np.linalg.eig(a)
>>> w;v
array([ 1.,  2.,  3.])
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

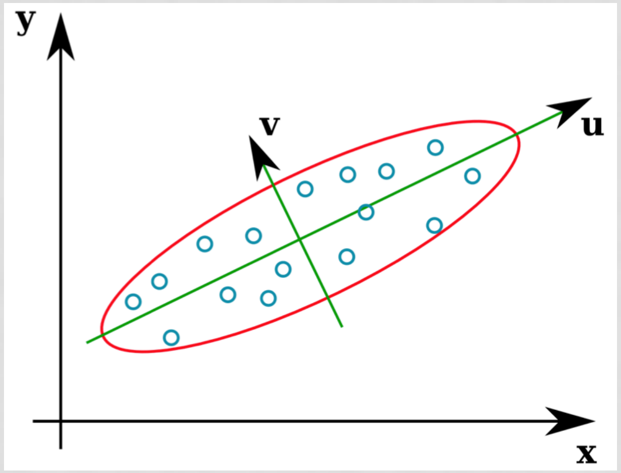
PRINCIPAL COMPONENT ANALYSIS

- Goal: to project a dataset from many correlated coordinates onto fewer uncorrelated coordinates (called principal components)

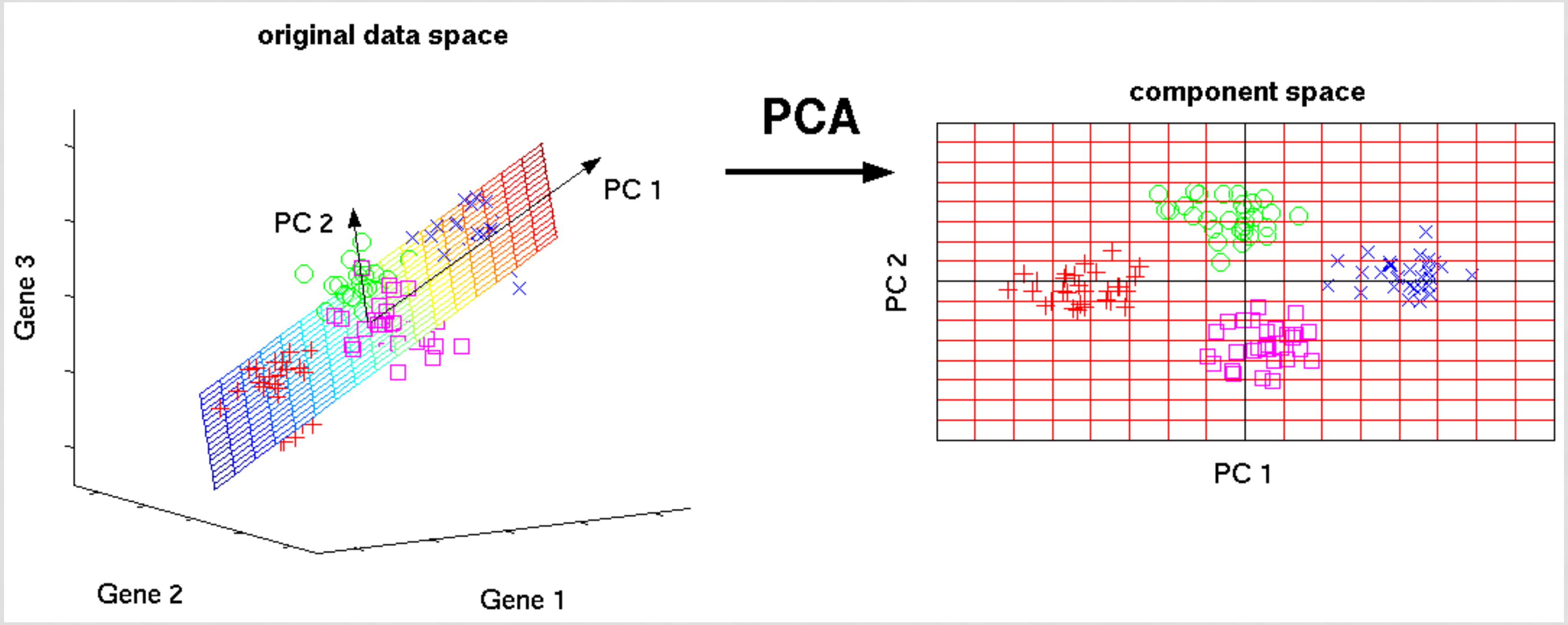
- How: realign data in the direction of most variance

uses

- dimensionality reduction
- data compression
- feature extraction
- visualization



PRINCIPAL COMPONENT ANALYSIS



PRINCIPAL COMPONENT ANALYSIS

Pros

- reduces complexity
- identifies most important features
- works in both supervised and unsupervised cases

Cons

- could throw away useful info
- may not be necessary; no new information
- requires a square matrix

SINGULAR VALUE DECOMPOSITION

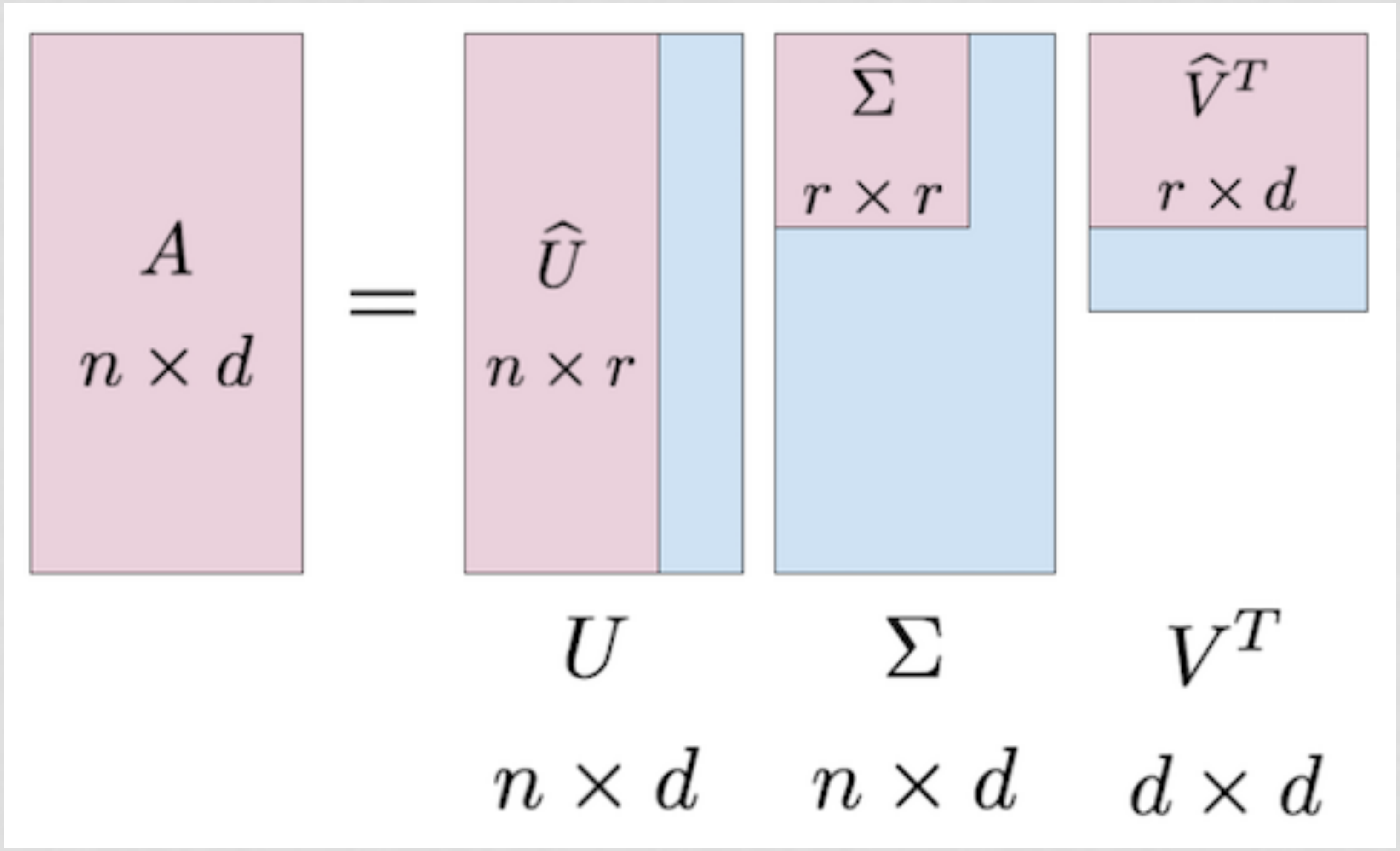
- matrix factorization technique used to approximate original data
- generalized PCA, allows for non-square matrices
- works in unsupervised case
- useful for discovering hidden topics

$$A = U\Sigma V^T$$

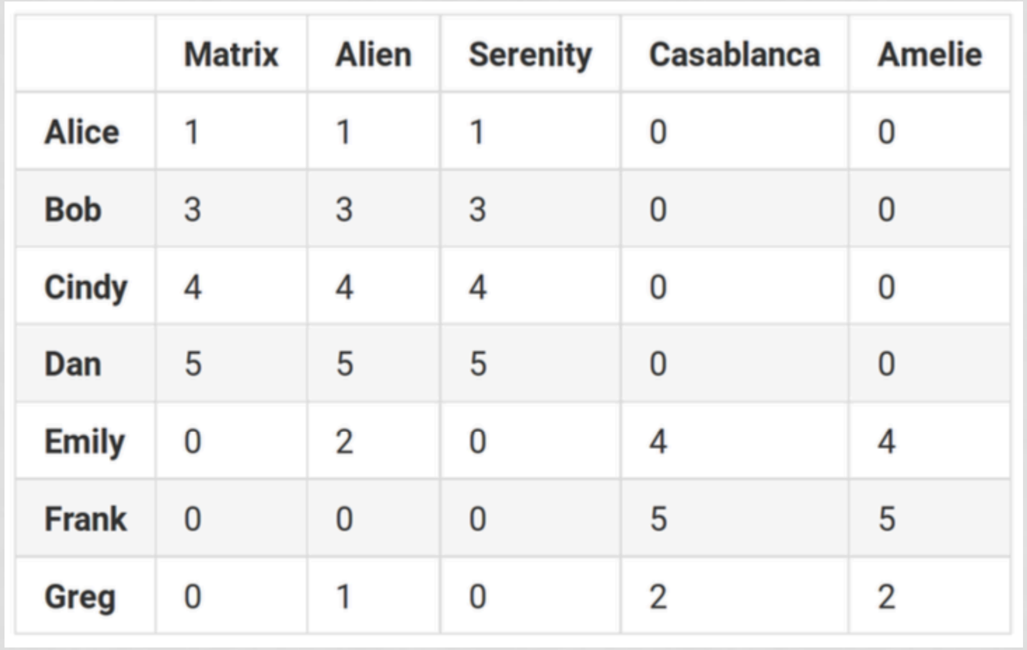
Where

- A is original matrix of shape *n* x *d*
- U has shape *n* x *d*
- VT has shape *d* x *d*
- Σ is a singular value matrix with shape *d* x *d*

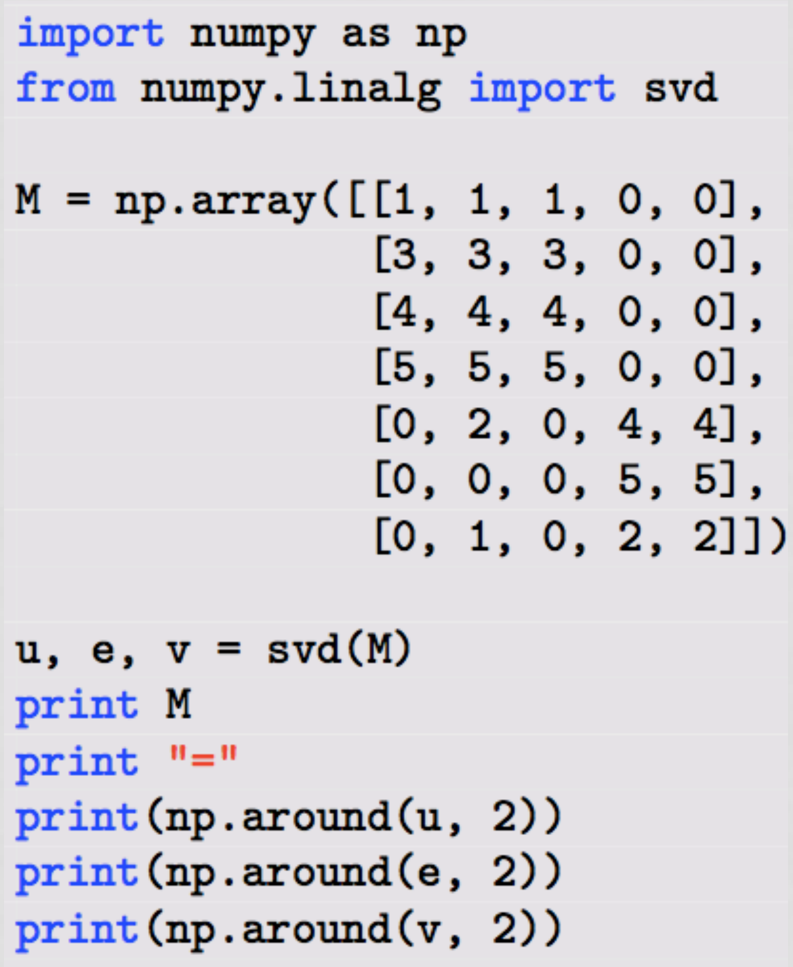
SINGULAR VALUE DECOMPOSITION



SINGULAR VALUE DECOMPOSITION



	Matrix	Alien	Serenity	Casablanca	Amelie
Alice	1	1	1	0	0
Bob	3	3	3	0	0
Cindy	4	4	4	0	0
Dan	5	5	5	0	0
Emily	0	2	0	4	4
Frank	0	0	0	5	5
Greg	0	1	0	2	2



```
import numpy as np
from numpy.linalg import svd

M = np.array([[1, 1, 1, 0, 0],
              [3, 3, 3, 0, 0],
              [4, 4, 4, 0, 0],
              [5, 5, 5, 0, 0],
              [0, 2, 0, 4, 4],
              [0, 0, 0, 5, 5],
              [0, 1, 0, 2, 2]])

u, e, v = svd(M)
print M
print "="
print(np.around(u, 2))
print(np.around(e, 2))
print(np.around(v, 2))
```

SINGULAR VALUE DECOMPOSITION

Science Fiction

- first singular value (12.4)
- first column of the U matrix
- first row of the V matrix

Romance

- second singular value (9.5)
- second column of the U matrix
- second row of the V matrix

$$\begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ \mathbf{3} & \mathbf{3} & \mathbf{3} & 0 & 0 \\ \mathbf{4} & \mathbf{4} & \mathbf{4} & 0 & 0 \\ \mathbf{5} & \mathbf{5} & \mathbf{5} & 0 & 0 \\ 0 & \mathbf{2} & 0 & \mathbf{4} & \mathbf{4} \\ 0 & 0 & 0 & \mathbf{5} & \mathbf{5} \\ 0 & \mathbf{1} & 0 & \mathbf{2} & \mathbf{2} \end{bmatrix} = \begin{bmatrix} -0.14 & -0.02 & -0.01 \\ \mathbf{-0.41} & -0.07 & -0.03 \\ \mathbf{-0.55} & -0.09 & -0.04 \\ \mathbf{-0.69} & -0.12 & -0.05 \\ -0.15 & \mathbf{0.59} & \mathbf{0.65} \\ -0.07 & \mathbf{0.73} & \mathbf{-0.68} \\ -0.08 & \mathbf{0.3} & \mathbf{0.33} \end{bmatrix} \begin{bmatrix} \mathbf{12.48} & 0.0 & 0.0 \\ 0.0 & \mathbf{9.51} & 0.0 \\ 0.0 & 0.0 & \mathbf{1.35} \end{bmatrix} \begin{bmatrix} \mathbf{-0.56} & \mathbf{-0.59} & \mathbf{-0.56} & -0.09 & -0.09 \\ -0.13 & 0.03 & -0.13 & \mathbf{0.7} & \mathbf{0.7} \\ \mathbf{-0.41} & \mathbf{0.8} & \mathbf{-0.41} & -0.09 & -0.09 \end{bmatrix}$$

With $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, U is the user-to-topic matrix and V is the movie-to-topic matrix

SINGULAR VALUE DECOMPOSITION

The third singular value is relatively small. We can exclude it with little loss of the data. General rule of thumb: keep at least 90% of data

$$
\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} \approx \begin{bmatrix} -0.14 & -0.02 \\ -0.41 & -0.07 \\ -0.55 & -0.09 \\ -0.69 & -0.12 \\ -0.15 & 0.59 \\ -0.07 & 0.73 \\ -0.08 & 0.3 \end{bmatrix} \begin{bmatrix} 12.48 & 0.0 \\ 0.0 & 9.51 \end{bmatrix} \begin{bmatrix} -0.56 & -0.59 & -0.56 & -0.09 & -0.09 \\ -0.13 & 0.03 & -0.13 & 0.7 & 0.7 \end{bmatrix}
$$

$$
= \begin{bmatrix} 0.99 & 1.01 & 0.99 & -0.0 & -0.0 \\ 2.98 & 3.04 & 2.98 & -0.0 & -0.0 \\ 3.98 & 4.05 & 3.98 & -0.01 & -0.01 \\ 4.97 & 5.06 & 4.97 & -0.01 & -0.01 \\ 0.36 & 1.29 & 0.36 & 4.08 & 4.08 \\ -0.37 & 0.73 & -0.37 & 4.92 & 4.92 \\ 0.18 & 0.65 & 0.18 & 2.04 & 2.04 \end{bmatrix}
$$

SINGULAR MATRIX DECOMPOSITION

NOW ITS YOUR TURN

REFERENCES

- Galvanize OpenSource: https://github.com/GalvanizeOpenSource/math-essentials-for-data-science
- Linear Algebra with Applications (4th Ed) by Otto Bretscher
- Machine Learning in Action by Peter Harrington
- Dot product: https://betterexplained.com/articles/vector-calculus-understanding-the-dot-product/
- more PCA and SVD: https://intoli.com/blog/pca-and-svd/