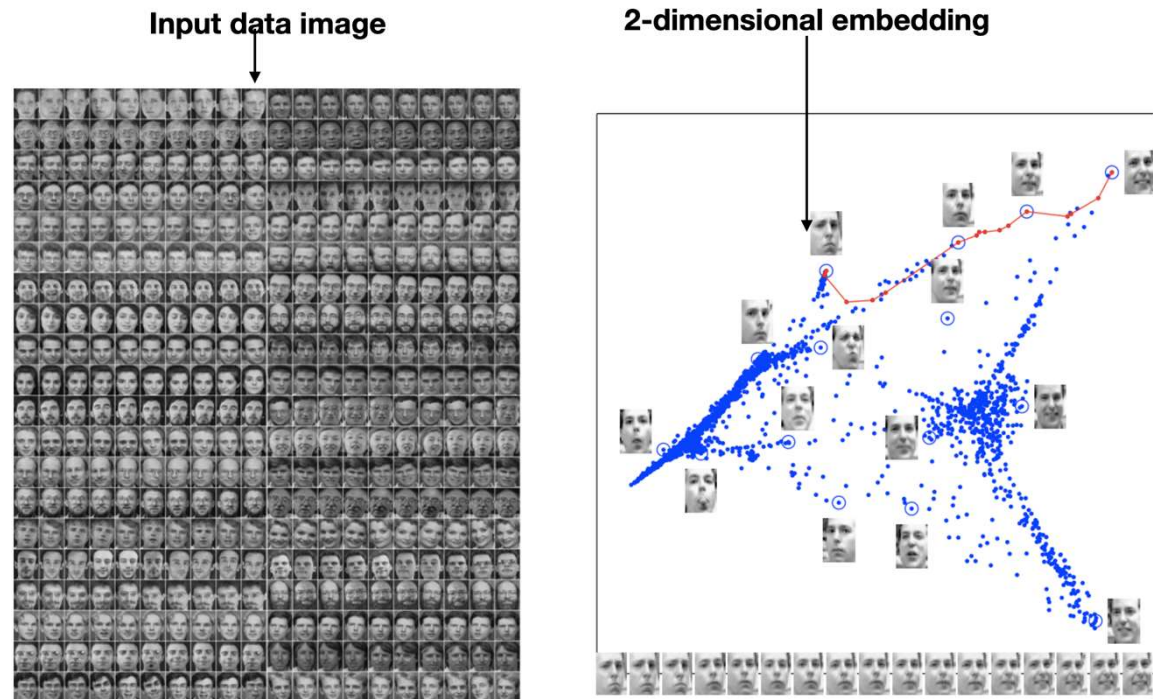




# Embedding Pictures

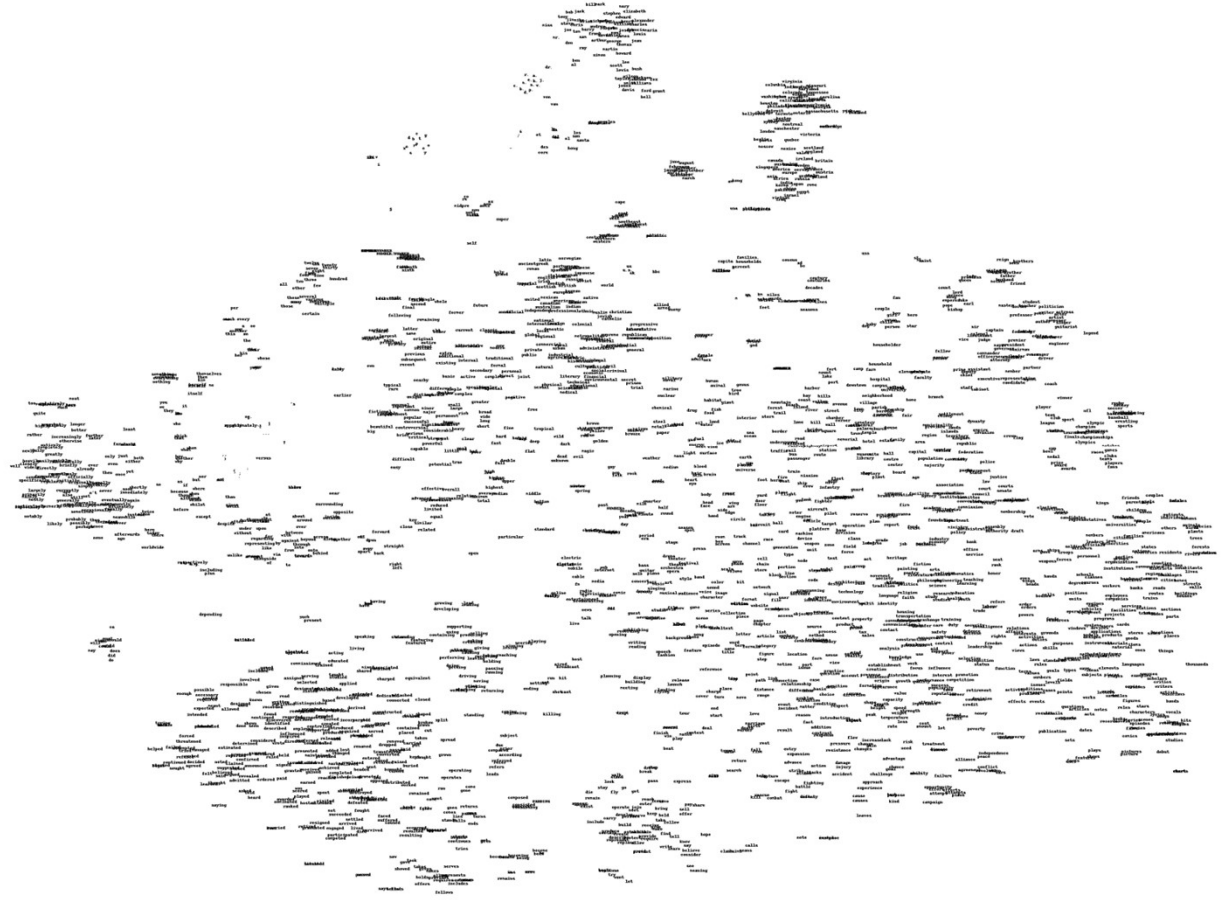
Example: Embed high dimensional data in low dimensions to visualize the data

- Goal: Similar images should be near each other.



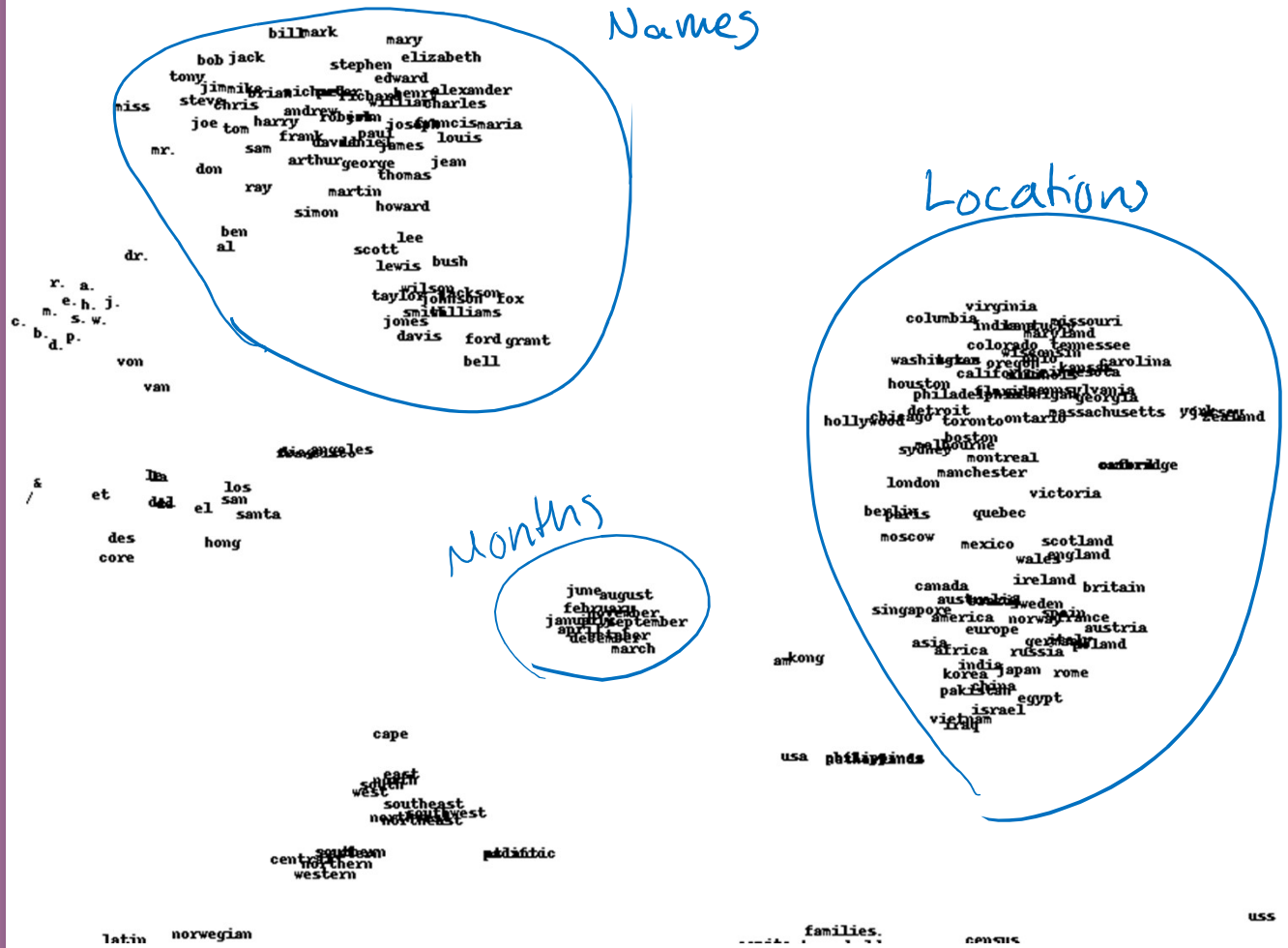
# Embedding Words

Feat 2



Feat 1

# Embedding Words



# Dimensionality Reduction

Input data might have thousands or millions of dimensions!

**Dimensionality Reduction** is the the task of representing the data with a fewer number of dimensions, while keeping meaningful relations between data

- **Easier Learning:** fewer parameters, no curse of dimensionality
- **Visualization:** Hard to visualize more than 3D
- Discover “**intrinsic dimensionality**” of the data
  - High dimensional data is truly lower dimensional (i.e. redundant information)

# Projections

$$x_i = (x_i[1], x_i[2], \dots, x_i[d])$$

Could do something like feature importance and find the subset of features with most meaningful information.

A popular approach is to **create new features** that are **combinations of existing features**

$$z_i[1] = 2x_i[1] + 3x_i[2] - 7x_i[3] + \dots + 12x_i[d]$$

$$z_i[2] = \dots$$

⋮

$$z_i[k] = \dots$$

We can do this in the unsupervised setting when we only know  $x$  and not  $y$ !

# PCA

One very popular dimensionality reduction algorithm is called **Principal Component Analysis (PCA)**.

Idea: Use a linear projection from  $d$ -dimensional data to  $k$ -dimensional data

- E.g. 1000 dimension word vectors to 3 dimensions

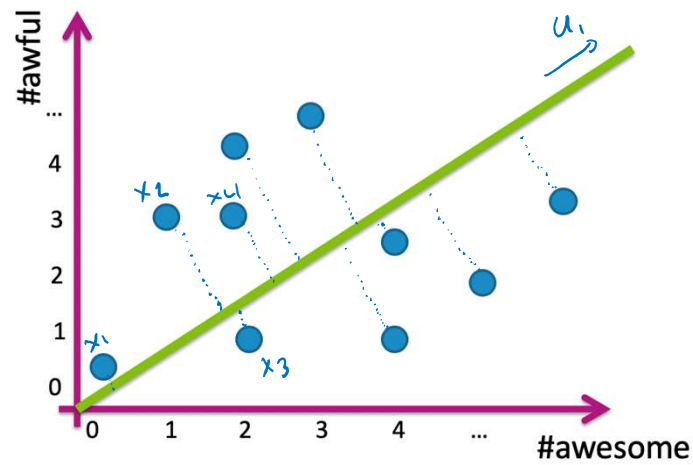
Choose the projection that minimizes **reconstruction error**

- Idea: The information lost if you were to "undo" the projection

# Linear Projection

$$z_i = u_i^T x_i = u_i[1]x_i[1] + \dots + u_i[d]x_i[d]$$

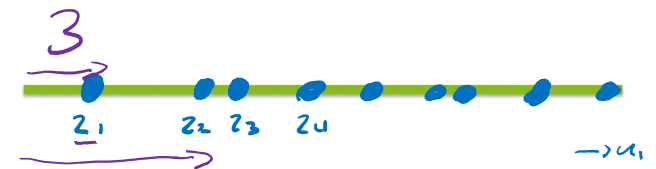
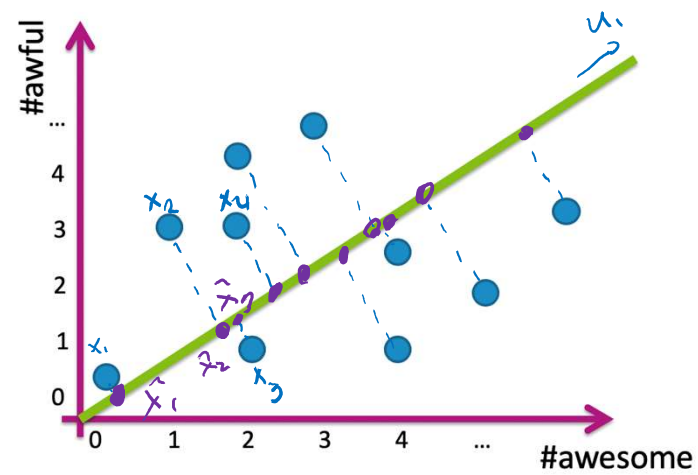
Project data into 1 dimension along a  $u_i$





# Reconstruction

Reconstruct original data only knowing the projection



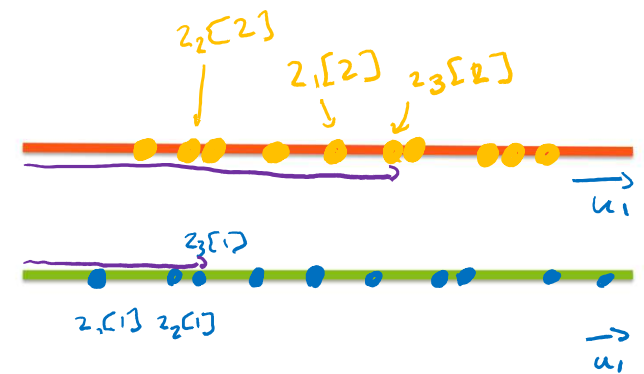
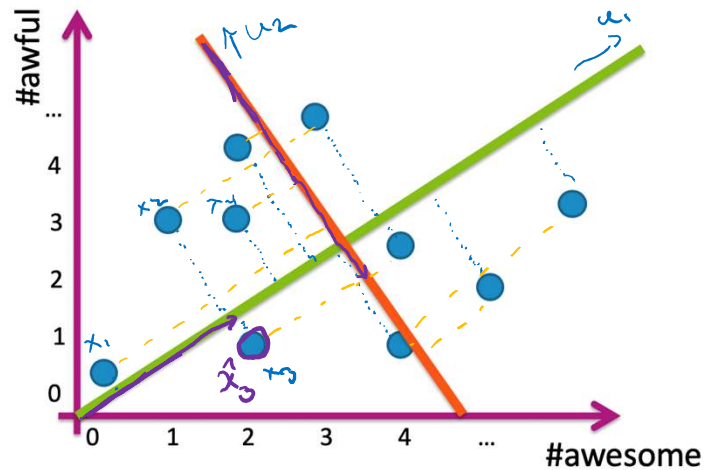
$$\hat{x}_i = z_i u_1$$

# Projection w/ 2 Dimensions

What if projected onto  $d$  orthogonal vectors?

$$z_i[1] = u_1^T x_i \quad \leftarrow \text{projection of } x_i \text{ on } u_1$$

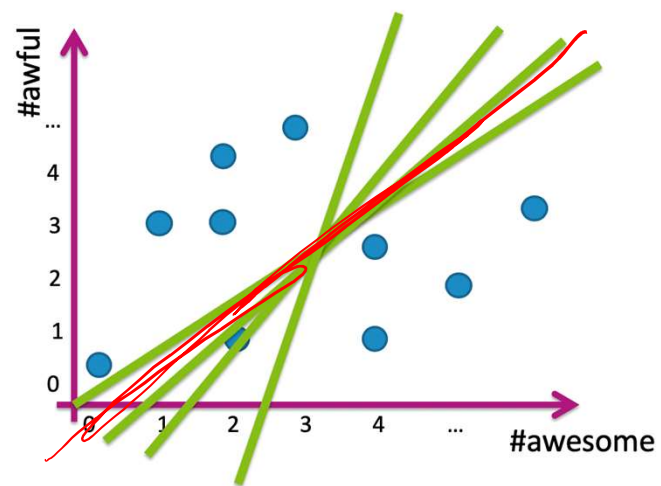
$$z_i[2] = u_2^T x_i \quad \leftarrow \text{projection of } x_i \text{ on } u_2$$



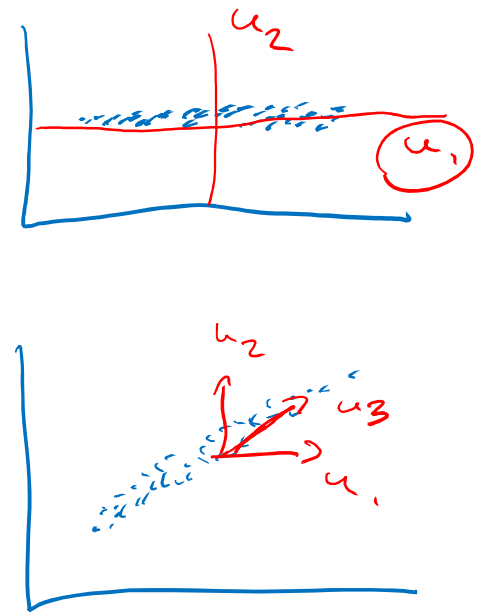
$$\hat{x}_i = z_i[1]u_1 + z_i[2]u_2$$

# Which Line?

Given a dataset, how do we choose which line to project on?



Extreme Example



# PCA Visualized

This process can work with any dimension data (go from  $d$  dimensions to  $k$  dimensions).

PCA Explained Visually:

<http://setosa.io/ev/principal-component-analysis/>

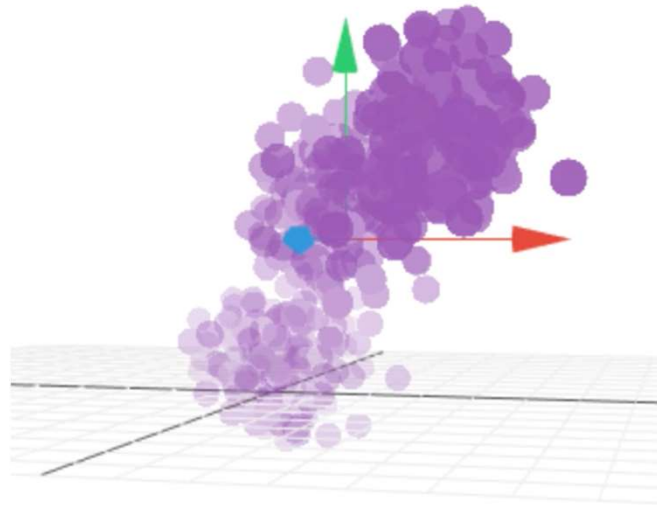
## Poll Everywhere

Think 

1 min

[pollev.com/cse416](https://pollev.com/cse416)

Assume we were trying to reduce the dimension of 3d data to 2d data, which of the following would be the result of PCA?  
Assume we are using the data from the visual demo and take pc1 and pc2 as the principal components.



**1:00**

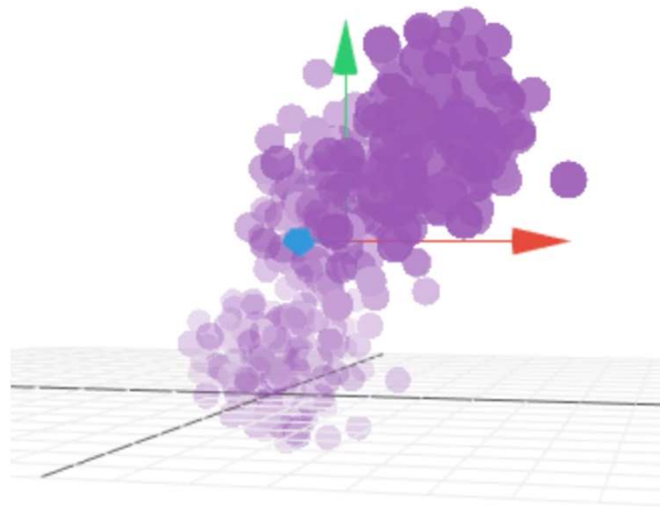
## Poll Everywhere

Pair 

2 min

[pollev.com/cse416](https://pollev.com/cse416)

Assume we were trying to reduce the dimension of 3d data to 2d data, which of the following would be the result of PCA?  
Assume we are using the data from the visual demo and take pc1 and pc2 as the principal components.



**2:00**



Brain Break

10:43



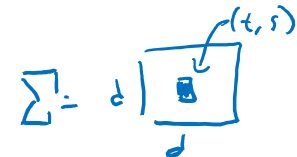
# PCA Algorithm

Input Data: An  $n \times d$  data matrix  $X$   $X = \begin{matrix} n \\ \text{---} x_i \text{---} \\ d \end{matrix}$

- Each row is an example

1. Recenter Data: Subtract mean from each row

$$X_c \leftarrow X - \bar{X}[1:d]$$



2. Compute spread/orientation: Compute covariance matrix  $\Sigma$

$$\Sigma[t, s] = \frac{1}{n} \sum_{i=1}^n x_{c,i}[t] x_{c,i}[s]$$

3. Find basis for orientation: Compute eigenvectors of  $\Sigma$

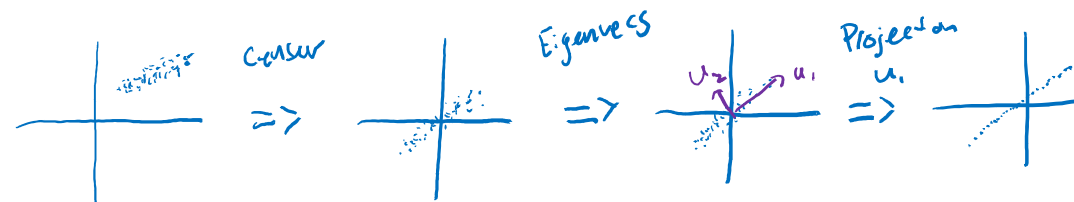
- Select  $k$  eigenvectors  $u_1, \dots, u_k$  with largest eigenvalues

4. Project Data: Project data onto principal

$$z_i[1] = u_1^T x_{c,i} = u_1[1]x_{c,i}[1] + \dots + u_1[d]x_{c,i}[d]$$

...

$$z_i[k] = u_k^T x_{c,i} = u_k[1]x_{c,i}[1] + \dots + u_k[d]x_{c,i}[d]$$





## Reconstructing Data

Using principal components and the projected data, you can reconstruct the data in the original domain.

$$\hat{x}_i[1:d] = \bar{X}[1:d] + \sum_{j=1}^k z_i[j] u_j$$

Choose  $u_1, \dots, u_k$  such that it minimizes

$$|\hat{X}_i[1:d] - X_i|$$


# Example: Eigenfaces

Apply PCA to face data

$64 \times 256$

Input Data




$x_i$    $256$

$u_1, u_2, \dots, u_{25}$

Principal Components



$u_i =$    $256 \Rightarrow 64 \times 25$

## Reconstructing Faces

Depending on context, it may make sense to look at either original data or projected data.

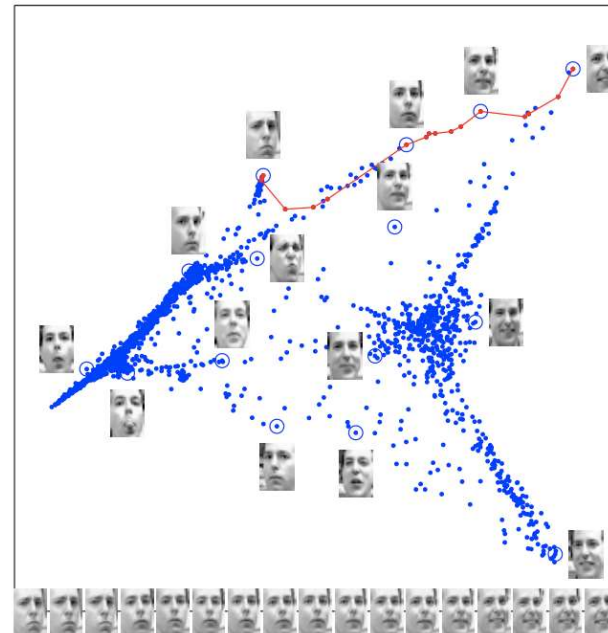
In this case, let's see how the original data looks after using more and more principal components for reconstruction.

- Each image shows additional 8 principal components



# Embedding Images

Other times, it does make sense to look at the data in the projected space! (Usually if  $k \leq 3$ )



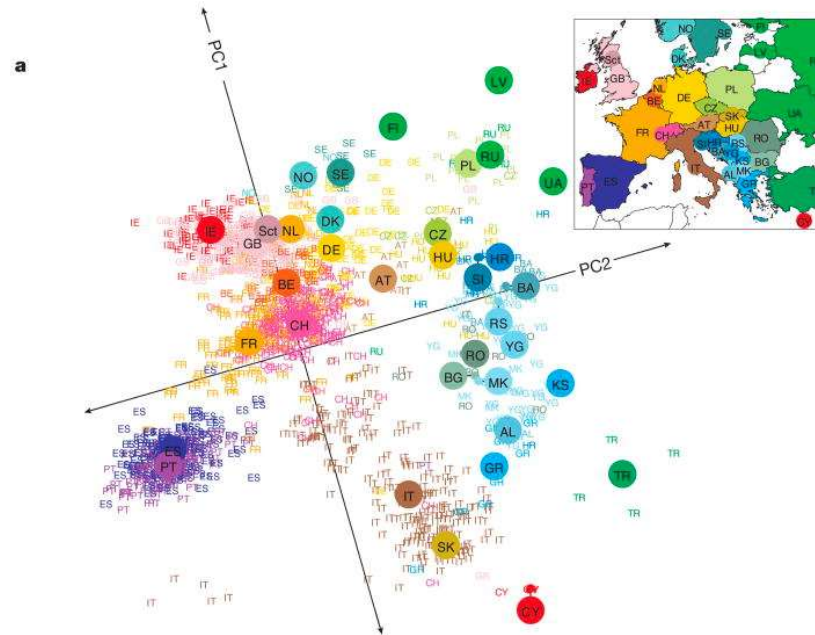
## Example: Genes

3192  
people

500,568 genes

country

Dataset of genes of Europeans (3192 people; 500,568 loci) and their country of origin, ran PCA on the data and plotted 2 principal components.



## Poll Everywhere

Think 

1 min

[pollev.com/cse416](https://pollev.com/cse416)

Using the PCA algorithm described in class on this gene data is an example of a:

- Supervised Learning Problem
- Unsupervised Learning Problem
- Neither
- I'm not sure

**1:00**

## Poll Everywhere

Pair 

2 min

[pollev.com/cse416](https://pollev.com/cse416)

Using the PCA algorithm described in class on this gene data is an example of a:

- Supervised Learning Problem
- Unsupervised Learning Problem
- Neither
- I'm not sure

**2:00**

# Scaling Up

Covariance matrix  $\Sigma$  can be very large with high-dimensional data

- $\Sigma \in \mathbb{R}^{d \times d}$  which can be quite large for 10,000 features
- This means finding the principal components will be slow

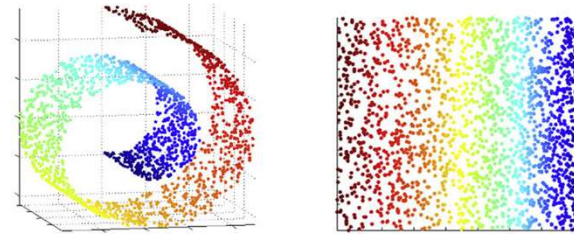
In practice, you can use the Singular Value Decomposition (SVD)

- Can be used to find the  $k$  eigenvectors with largest eigenvalue
- Very fast implementations available
- Don't care if you know what SVD is, but you will likely use it in practice if you are doing anything with PCA.



## PCA Failure Modes

PCA assumes there is a lower dimensional **linear subspace** that represents the data well. Works some times, but can fail in practice.



May want to look into non-linear dimensionality reduction

- Manifold learning
- Popular: SDD Maps, Isomap, LLE, t-SNE

# PCA Recap

## Dimensionality Reduction

- Why and when it's important

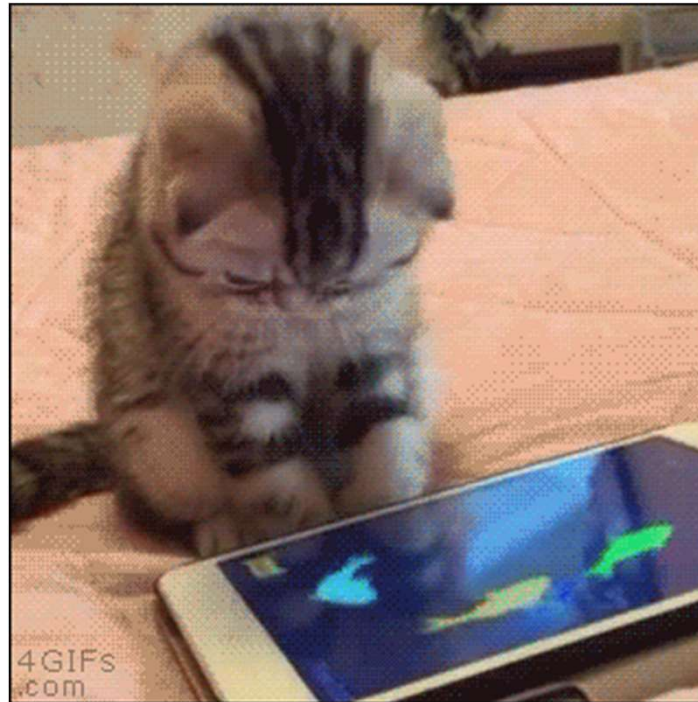
## Principal Component Analysis (PCA)

- High level intuition for what the algorithm is doing
- Goal: Minimizing reconstruction error



Brain Break

11:23





# Personalization

Personalization is transforming our experience of the world

Youtube

Netflix 

Amazon

Spotify

Facebook

Many more...

Almost all have share a common trait where there are users that use the system and items that we want the user to look at.

A recommender system recommends items to a user based on what we think will be the most “useful” for the user.

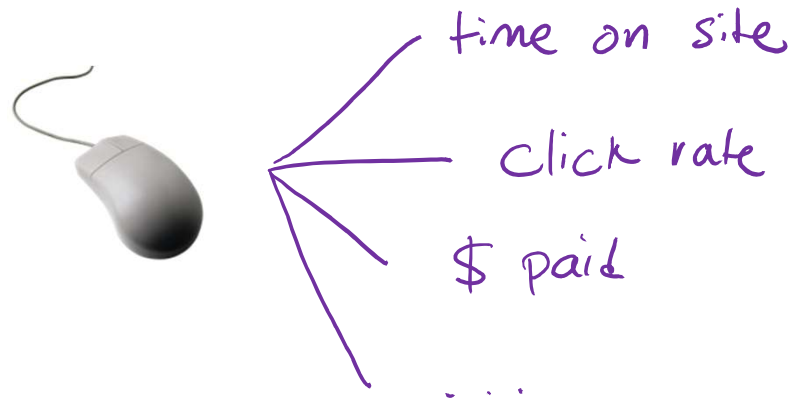
## Recommender System Challenges

## Types of Feedback

Explicit - User tells us what she likes



Implicit - We try to infer what she likes from usage data



## Top-k vs Diverse Outputs

Top-k recommendations might be very redundant

- Someone who likes Rocky I also will likely enjoy Rocky II, Rocky III, Rocky IV, Rocky V

Diverse Recommendations

- Users are multi-faceted & we want to hedge our bets
- Maybe recommend: Rocky II, Always Sunny in Philadelphia, Robin Hood



## Cold Start

When a new movie comes into our system, we don't know who likes it! This is called the **cold start** problem.

Generally, to solve we will need “side information”

- Genre, actors, if it's a sequel

Could also try to test users to see if they like it to learn quickly

## That's So Last Year

Interests change over time

- Is it 1967?
- Or 1977?
- Or 1998?
- Or 2011?

Models need flexibility to adapt to users

- Macro scale
- Micro scale (fads)



# Scalability

For  $N$  users and  $M$  movies, some approaches take  $\mathcal{O}(N^3 + M^3)$  time. This can be prohibitively slow for billions of users.

Big focus has been on:

- Efficient implementations
- Exact or faster approximate methods as needed

# Popularity

*Solution 0*

# Popularity

Simplest Approach: Recommend whatever is popular

- Rank by global popularity (i.e. Avengers Endgame)

## Limitations

- No personalization
- Feedback loop

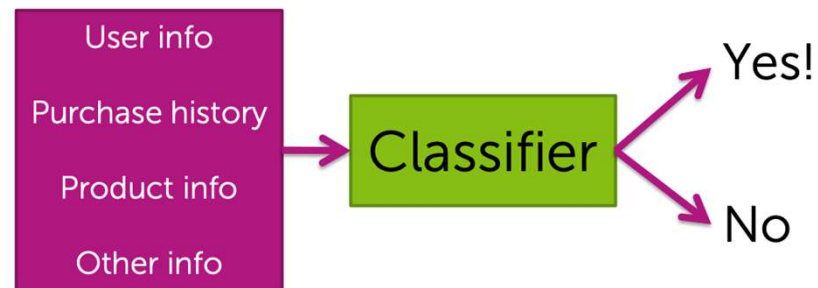
# Classification Model

*Solution 1*

## Learn a Classifier

Common to use something like logistic Regression so that you have probability predictions

Train a classifier to learn whether or not someone will like an item

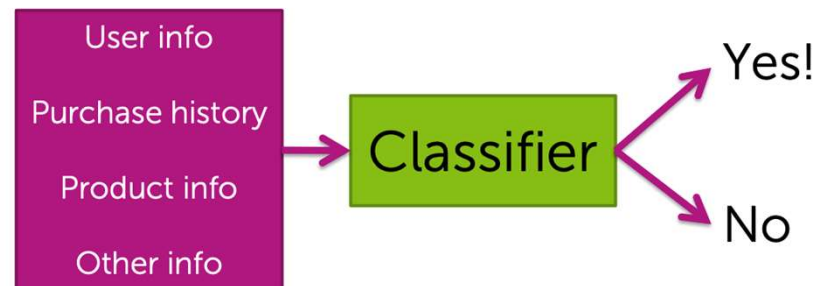


### Pros

- Personalized
- Features can capture context (time of day, recent history, ...)
- Can even handle limited user history (age of user, location, ...)

# Learn a Classifier

Train a classifier to learn whether or not someone will like an item



## Cons

- Features might not be available or hard to work with
- Often doesn't perform well in practice when compared to more advanced techniques like **collaborative filtering**