

# Unsupervised Learning, Recommenders, Reinforcement Learning

## K-means algorithm

Randomly initialize  $K$  cluster centroids  $\mu_1, \mu_2, \dots, \mu_K$

Repeat {

# Assign points to cluster centroids

for  $i = 1$  to  $m$

$c^{(i)} :=$  index (from 1 to  $K$ ) of cluster centroid closest to  $x^{(i)}$

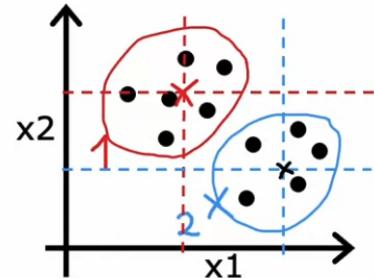
# Move cluster centroids

for  $k = 1$  to  $K$

$\mu_k :=$  average (mean) of points assigned to cluster  $k$

}

$$\begin{array}{l} \mu_1, \mu_2 \\ x^{(1)}, x^{(2)}, \dots, x^{(m)} \\ n=2 \\ K=2 \end{array}$$



## K-means optimization objective

$c^{(i)}$  = index of cluster ( $1, 2, \dots, K$ ) to which example  $x^{(i)}$  is currently assigned

$\mu_k$  = cluster centroid  $k$

$\mu_{c^{(i)}}$  = cluster centroid of cluster to which example  $x^{(i)}$  has been assigned

### Cost function

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

$\min_{c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K} J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

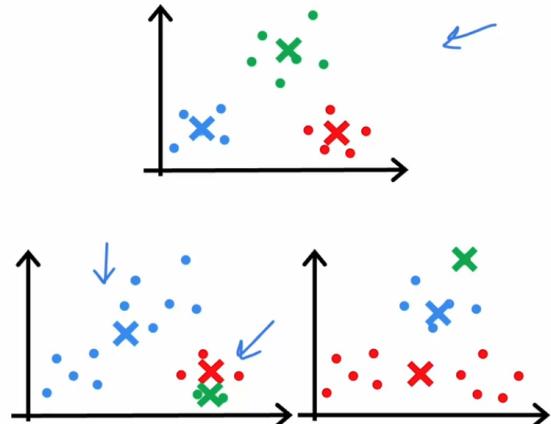
Distortion

## Random initialization

Choose  $K < m$

Randomly pick  $K$  training examples.

Set  $\mu_1, \mu_2, \dots, \mu_k$  equal to these  $K$  examples.



With random initialization using some of the data points themselves, depending on which points you choose, we can end up with different clusters. Local minima of the cost function may be found. We can run it with multiple random initializations and then pick the one with lowest final cost function.

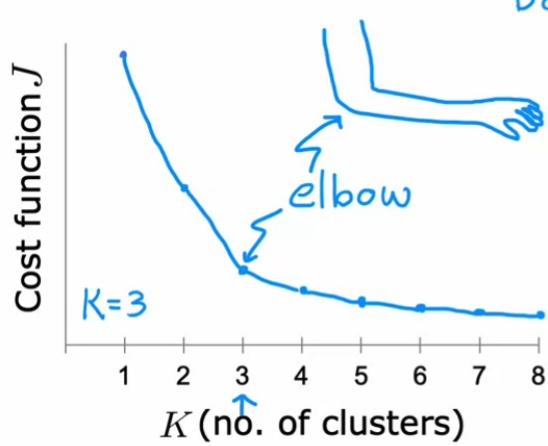
## Random initialization

```
For i = 1 to 100 {  
    Randomly initialize K-means.  
    Run K-means. Get  $c^{(1)}, \dots, c^{(m)}, \mu_1, \mu_2, \dots, \mu_k \leftarrow$   
    Computer cost function (distortion)  
     $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \mu_2, \dots, \mu_k) \leftarrow$   
}
```

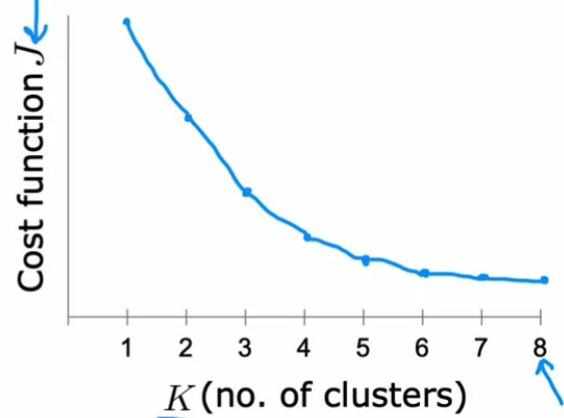
Pick set of clusters that gave lowest cost  $J$

## Choosing the value of K

Elbow method

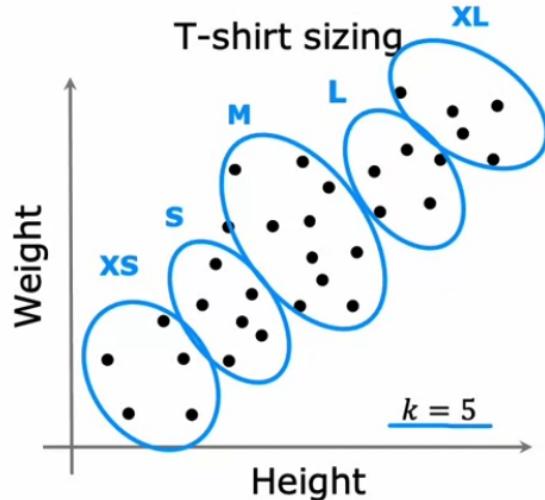
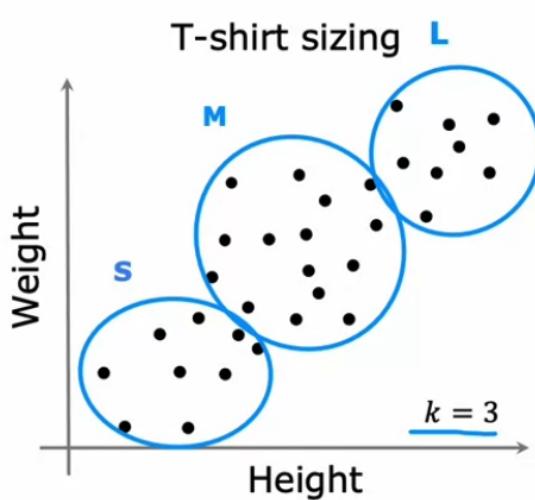


the right "k" is often ambiguous  
Don't choose K just to minimize cost  $J$



## Choosing the value of K

Often, you want to get clusters for some later (downstream) purpose.  
Evaluate K-means based on how well it performs on that later purpose.



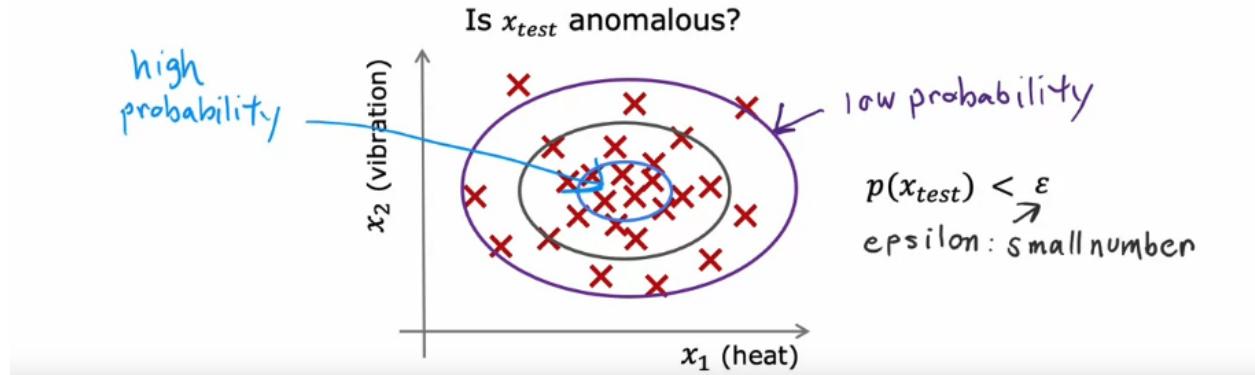
## Anomaly Detection

Majority of training examples are normal. Does the test example look like them?

Uses: fraud (e.g. financial), manufacturing defects, computer monitoring in data centers

## Density estimation

Dataset:  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$  probability of  $x$  being seen in dataset  
Model  $p(x)$



## Density estimation

Training set:  $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(m)}\}$   
Each example  $\vec{x}^{(i)}$  has  $n$  features

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$p(\vec{x}) = p(x_1; \mu_1, \sigma_1^2) * p(x_2; \mu_2, \sigma_2^2) * p(x_3; \mu_3, \sigma_3^2) * \dots * p(x_n; \mu_n, \sigma_n^2)$$

$$= \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) \quad \sum \quad \prod$$

"add"      "multiply"

$$\begin{aligned} p(x_1 = \text{high temp}) &= 1/10 \\ p(x_2 = \text{high vibra}) &= 1/20 \\ p(x_1, x_2) &= p(x_1) * p(x_2) \\ &= \frac{1}{10} \times \frac{1}{20} = \frac{1}{200} \end{aligned}$$

We fit a Gaussian distribution for each feature that could be indicative of anomaly. The intuition is that if any feature is too small or large, then it's likely an anomaly...

## Anomaly detection algorithm

1. Choose  $n$  features  $x_i$  that you think might be indicative of anomalous examples.

2. Fit parameters  $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$$

$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$$

Vectorized formula

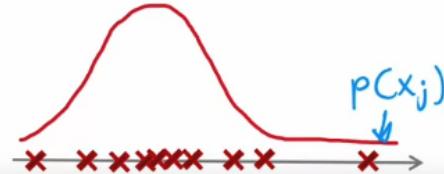
$$\vec{\mu} = \frac{1}{m} \sum_{i=1}^m \vec{x}^{(i)}$$

$$\vec{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix}$$

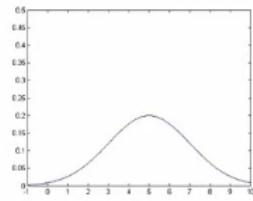
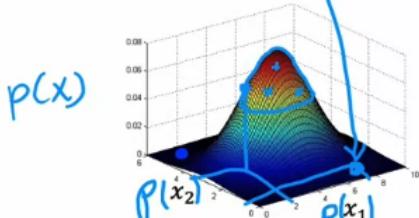
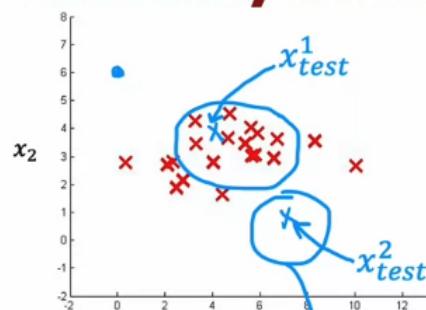
3. Given new example  $x$ , compute  $p(x)$ :

$$p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if  $p(x) < \varepsilon$

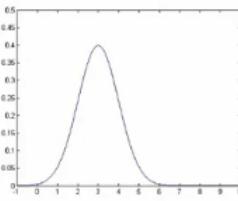


## Anomaly detection example



$$\mu_1 = 5, \sigma_1 = 2$$

$$p(x_1; \mu_1, \sigma_1^2)$$



$$\mu_2 = 3, \sigma_2 = 1$$

$$p(x_2; \mu_2, \sigma_2^2)$$

$$\varepsilon = 0.02$$

$$p(x_{test}^{(1)}) = 0.0426 \longrightarrow "ok"$$

$$p(x_{test}^{(2)}) = 0.0021 \longrightarrow \text{anomaly}$$

## The importance of real-number evaluation

When developing a learning algorithm (choosing features, etc.), making decisions is much easier if we have a way of evaluating our learning algorithm.

Assume we have some labeled data, of anomalous and non-anomalous examples.

$$y=1 \quad y=0$$

Training set:  $x^{(1)}, x^{(2)}, \dots, x^{(m)}$  (assume normal examples/not anomalous)

$y=0$  for all training examples

Cross validation set:  $(x_{cv}^{(1)}, y_{cv}^{(1)}), \dots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$   
Test set:  $(x_{test}^{(1)}, y_{test}^{(1)}), \dots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

} include a few anomalous examples  
 $y=1$       mostly normal examples  
 $y=0$

## Aircraft engines monitoring example

10000 good (normal) engines  
20 flawed engines (anomalous)  
2

Training set: 6000 good engines       $y=0$       2 to 50  
CV: 2000 good engines ( $y=0$ )      10 anomalous ( $y=1$ )  
Test: 2000 good engines ( $y=0$ ), 10 anomalous ( $y=1$ )

use cross validation set      tune  $\epsilon$       tune  $x_j$

Alternative: No test set      use if very few labeled anomalous examples

Training set: 6000 good engines 2  
CV: 4000 good engines ( $y=0$ ), 20 anomalous ( $y=1$ )  
tune  $\epsilon$       tune  $x_j$

We use the cross validation to tune parameters of the model and the test set to see how well it's doing. If we have very few anomalous examples, one alternative is to only have a cross validation set, although the chances of overfitting are higher. The algorithm is still unsupervised since the training examples are unlabeled data. We use labels only on the cross validation to evaluate the model that was already fit. We can then get metrics on the cross validation and test sets. We can calculate TP, FP, FN, TN, precision, recall, f1 score, specially if the dataset is really skewed.

## Anomaly detection vs. Supervised learning

Very small number of positive examples ( $y = 1$ ). (0-20 is common).

Large number of negative ( $y = 0$ ) examples.

P(x)

y=1

Many different “types” of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like; future anomalies may look nothing like any of the anomalous examples we've seen so far.

Fraud

Large number of positive and negative examples.

20 positive examples

Enough positive examples for algorithm to get a sense of what positive examples are like, future positive examples likely to be similar to ones in training set.

Spam

### Anomaly detection

- Fraud detection
- Manufacturing - Finding new previously unseen defects in manufacturing.(e.g. aircraft engines)
- Monitoring machines in a data center
- ⋮

### vs. Supervised learning

- Email spam classification
- Manufacturing - Finding known, previously seen defects scratches  $y=1$
- Weather prediction (sunny/rainy/etc.)
- Diseases classification
- ⋮

Carefully choosing the right features is more important for anomaly detection than supervised learning.

- Pick Gaussian features or adjust non-gaussian features to gaussian (e.g. take the log or the sqrt)
- Error analysis (which anomalies is the algorithm not detecting?) to try and identify a new feature that would help it label the anomaly correctly

# Recommender Systems

e.g. Given set of movies that a user has rated, what movies could we recommend next from the ones the user hasn't watched yet?

If we get access to features about the movies, we can fit a linear regression model. We can use regularization to prevent overfitting.

## What if we have features of the movies?

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)	$x_1$ (romance)	$x_2$ (action)	$n_u = 4$	$n_m = 5$	$n = 2$
Love at last	5	5	0	0	0.9	0			
Romance forever	5	?	?	0	1.0	0.01			
→ Cute puppies of love	?	4	0	?	0.99	0			
Nonstop car chases	0	0	5	4	0.1	1.0			
Swords vs. karate	0	0	5	?	0	0.9			

For user 1: Predict rating for movie  $i$  as:  $w^{(1)} \cdot x^{(i)} + b^{(1)}$  ← just linear regression

$$w^{(1)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}, b^{(1)} = 0, x^{(3)} = \begin{bmatrix} 0.99 \\ 0 \end{bmatrix}, w^{(1)} \cdot x^{(3)} + b^{(1)} = 4.95$$

→ For user  $j$ : Predict user  $j$ 's rating for movie  $i$  as  $w^{(j)} \cdot x^{(i)} + b^{(j)}$

## Cost function

Notation:

- $r(i,j) = 1$  if user  $j$  has rated movie  $i$  (0 otherwise)
- $y^{(i,j)}$  = rating given by user  $j$  on movie  $i$  (if defined)
- $w^{(j)}, b^{(j)}$  = parameters for user  $j$
- $x^{(i)}$  = feature vector for movie  $i$

For user  $j$  and movie  $i$ , predict rating:  $w^{(j)} \cdot x^{(i)} + b^{(j)}$

- $m^{(j)}$  = no. of movies rated by user  $j$
- To learn  $w^{(j)}, b^{(j)}$

$$\min_{w^{(j)}, b^{(j)}} J(w^{(j)}, b^{(j)}) = \frac{1}{2m^{(j)}} \sum_{i: r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2m^{(j)}} \sum_{k=1}^n (w_k^{(j)})^2$$

## Cost function

To learn parameters  $w^{(j)}, b^{(j)}$  for user  $j$  :

$$J(w^{(j)}, b^{(j)}) = \frac{1}{2} \sum_{i:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^n (w_k^{(j)})^2$$

To learn parameters  $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \dots, w^{(n_u)}, b^{(n_u)}$  for all users :

$$\frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2$$

If we don't have the features, we'd have to learn them. We're now given a set of parameters of the model and minimize a cost function to get the features.

## Cost function

Given  $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \dots, w^{(n_u)}, b^{(n_u)}$

to learn  $x^{(i)}$  :

$$J(x^{(i)}) = \frac{1}{2} \sum_{j:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^n (x_k^{(i)})^2$$

→ To learn  $x^{(1)}, x^{(2)}, \dots, x^{(n_m)}$ :

$$J(x^{(1)}, x^{(2)}, \dots, x^{(n_m)}) = \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

## Collaborative filtering

Cost function to learn  $w^{(1)}, b^{(1)}, \dots, w^{(n_u)}, b^{(n_u)}$ :

$$\min_{w^{(1)}, b^{(1)}, \dots, w^{(n_u)}, b^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2$$

	$j=1$	$j=2$	$j=3$
Alice	5	5	?
Bob	?	2	3

Cost function to learn  $x^{(1)}, \dots, x^{(n_m)}$ :

$$\min_{x^{(1)}, \dots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

Put them together:

$$\min_{\substack{w^{(1)}, \dots, w^{(n_u)} \\ b^{(1)}, \dots, b^{(n_u)} \\ x^{(1)}, \dots, x^{(n_m)}}} J(w, b, x) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

## Gradient Descent

*collaborative filtering*

Linear regression (course 1)

repeat {

$$\begin{aligned} \underline{w_i} &= w_i - \alpha \frac{\partial}{\partial w_i} J(w, b) \\ \underline{b} &= b - \alpha \frac{\partial}{\partial b} J(w, b) \end{aligned}$$

$$\begin{aligned} w_i^{(j)} &= w_i^{(j)} - \alpha \frac{\partial}{\partial w_i^{(j)}} J(w, b, x) \\ b^{(j)} &= b^{(j)} - \alpha \frac{\partial}{\partial b^{(j)}} J(w, b, x) \end{aligned}$$

$$x_k^{(i)} = x_k^{(i)} - \alpha \frac{\partial}{\partial x_k^{(i)}} J(w, b, x)$$

}

parameters  $w, b, x$

$\times$  is also a parameter

Collaborative filtering naming is in the sense that multiple users have helped come up with good values for the features of the model and then we can use these to predict rating for other users in the future.

For binary labels, (e.g. likes, favs, clicks) we can use logistic regression and the corresponding loss function we've seen before.

## Cost function for binary application

Previous cost function:

$$\frac{1}{2} \sum_{(i,j):r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2$$

Loss for binary labels  $y^{(i,j)} : f_{(w,b,x)}(x) = g(w^{(j)} \cdot x^{(i)} + b^{(j)})$

$$L(f_{(w,b,x)}(x), y^{(i,j)}) = -y^{(i,j)} \log(f_{(w,b,x)}(x)) - (1 - y^{(i,j)}) \log(1 - f_{(w,b,x)}(x))$$

Loss for single example

$$J(w, b, x) = \sum_{(i,j):r(i,j)=1} L(f_{(w,b,x)}(x), y^{(i,j)})$$

cost for all examples

## Users who have not rated any movies

Movie	Alice(1)	Bob (2)	Carol (3)	Dave (4)	Eve (5)	
Love at last	5	5	0	0	?	5 5 0 0 ?
Romance forever	5	?	?	0	?	5 ? ? 0 ?
Cute puppies of love	?	4	0	?	?	? 4 0 ? ?
Nonstop car chases	0	0	5	4	?	0 0 5 4 ?
Swords vs. karate	0	0	5	?	?	0 0 5 0 ?

$$\min_{\substack{w^{(1)}, \dots, w^{(n_u)} \\ b^{(1)}, \dots, b^{(n_u)} \\ x^{(1)}, \dots, x^{(n_m)}}} \frac{1}{2} \sum_{(i,j):r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

$w^{(s)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad b^{(s)} = 0 \quad w^{(s)} \cdot x^{(i)} + b^{(s)}$

Initial guess for a new user is the average of the ratings for all other users. This makes the algorithm run faster and predictions be more reasonable.

## Mean Normalization

$$\begin{array}{c}
 \xrightarrow{\quad} \boxed{5 \ 0 \ 0 \ ? \ 2.5} \\
 \xrightarrow{\quad} \boxed{5 \ ? \ ? \ 0 \ ? \ 2.5} \\
 ? \ 4 \ 0 \ ? \ ? \ 2 \\
 \boxed{0 \ 0 \ 5 \ 4 \ ? \ 2.25} \\
 \boxed{0 \ 0 \ 5 \ 0 \ ? \ 1.25}
 \end{array}
 \quad \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix}
 \quad \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

For user  $j$ , on movie  $i$  predict:

$$w^{(j)} \cdot x^{(i)} + b^{(j)} + \mu_i$$

User 5 (Eve):

$$w^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad b^{(5)} = 0 \quad \underbrace{w^{(5)} \cdot x^{(1)} + b^{(5)}}_0 + \mu_1 = 2.5$$

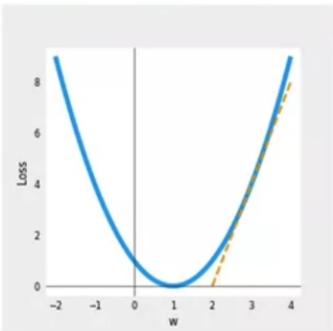
With tensorflow, we only need to tell it how to compute the function and the related cost function. It computes the gradients for us ("auto diff").

$$J = (\mathbf{w}\mathbf{x} - \mathbf{y})^2$$

Gradient descent algorithm  
Repeat until convergence

$$\mathbf{w} = \mathbf{w} - \alpha \frac{\partial J(\mathbf{w}, \mathbf{b})}{\partial \mathbf{w}}$$

Fix  $\mathbf{b} = 0$  for this example



## Custom Training Loop

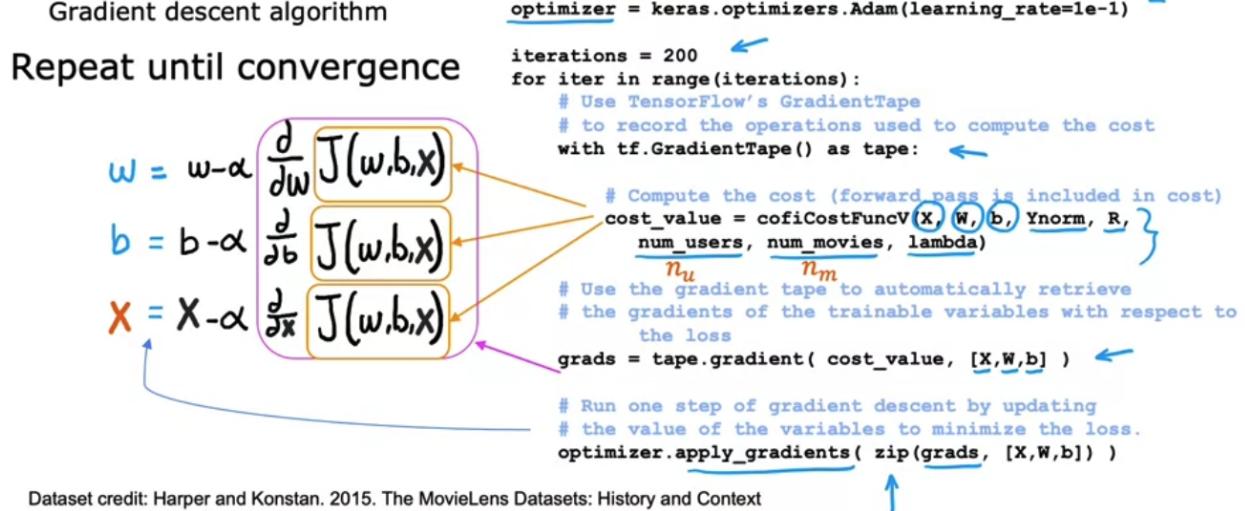
```
w = tf.Variable(3.0)
x = 1.0
y = 1.0 # target value
alpha = 0.01

iterations = 30
for iter in range(iterations):
    # Use TensorFlow's Gradient tape to record the steps
    # used to compute the cost J, to enable auto differentiation.
    with tf.GradientTape() as tape:
        fwb = w*x
        costJ = (fwb - y)**2

    # Use the gradient tape to calculate the gradients
    # of the cost with respect to the parameter w.
    [dJdw] = tape.gradient(costJ, [w])

    # Run one step of gradient descent by updating
    # the value of w to reduce the cost.
    w.assign_add(-alpha * dJdw)
```

## Implementation in TensorFlow



The learned features might be difficult to interpret but nevertheless, we can find similar items if we get the items with smallest distances to any given item.

## Finding related items

The features  $x^{(i)}$  of item  $i$  are quite hard to interpret.

To find other items related to it,

find item  $k$  with  $x^{(k)}$  similar to  $x^{(i)}$

i.e. with smallest distance

$$\sum_{l=1}^n (x_l^{(k)} - x_l^{(i)})^2$$

$$\|x^{(k)} - x^{(i)}\|^2$$

Collaborative filtering has a cold start problem:

- rank new items that few users have rated?

- show something reasonable to new users who have rated few items?

## Collaborative filtering vs Content-based filtering

→ Collaborative filtering:

Recommend items to you based on ratings of users who gave similar ratings as you

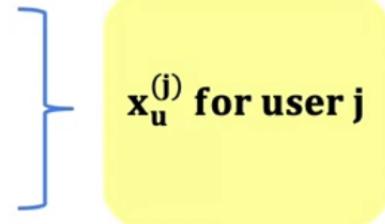
→ Content-based filtering:

Recommend items to you based on features of user and item to find good match

## Examples of user and item features

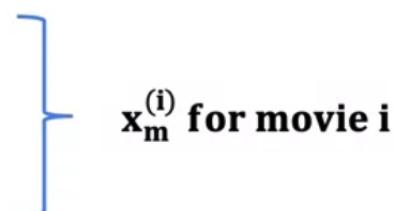
### User features:

- • Age
- • Gender (1 hot)
- • Country (1 hot, 200)
- • Movies watched (1000)
- • Average rating per genre
- ...



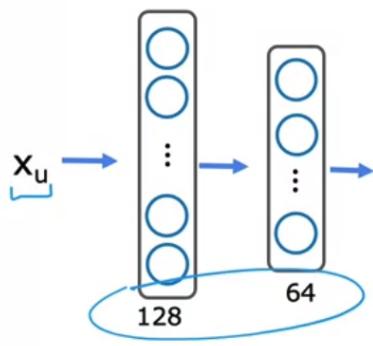
### Movie features:

- • Year
- • Genre/Genres
- • Reviews
- • Average rating
- ...

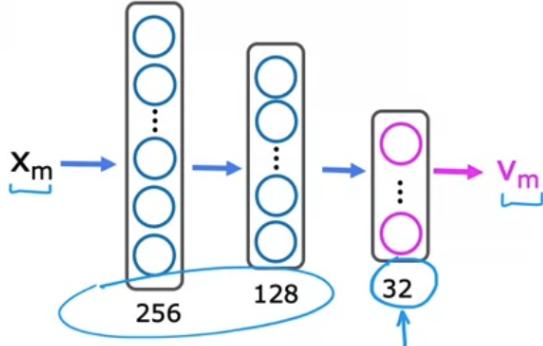


## Neural network architecture

$x_u \rightarrow v_u$  User network



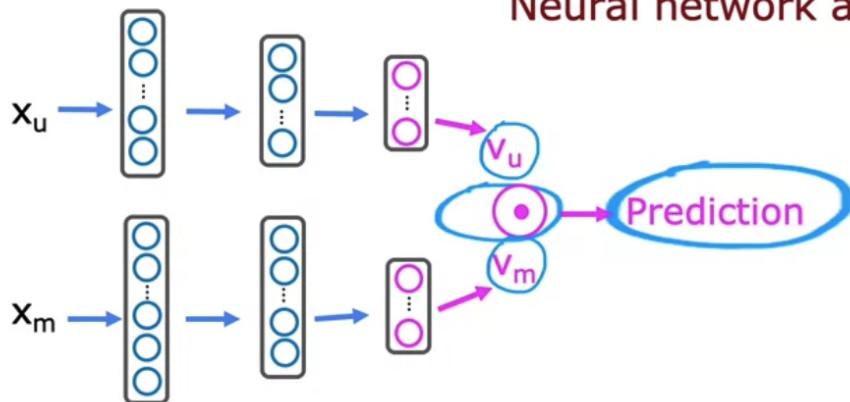
$x_m \rightarrow v_m$  Movie network



**Prediction :**  $v_u \cdot v_m$

$g(v_u \cdot v_m)$  to predict the probability that  $y^{(i,j)}$  is 1

## Neural network architecture



Cost function  $J = \sum_{(i,j):r(i,j)=1} (v_u^{(j)} \cdot v_m^{(i)} - y^{(i,j)})^2 + \text{NN regularization term}$

## Learned user and item vectors:

- $v_u^{(j)}$  is a vector of length 32 that describes user j with features  $x_u^{(j)}$
- $v_m^{(i)}$  is a vector of length 32 that describes movie i with features  $x_m^{(i)}$

To find movies similar to movie i:

$$\|v_m^{(k)} - v_m^{(i)}\|^2 \text{ small}$$

$$\|x^{(k)} - x^{(i)}\|^2$$

Note: This can be pre-computed ahead of time

## Two steps: Retrieval & Ranking

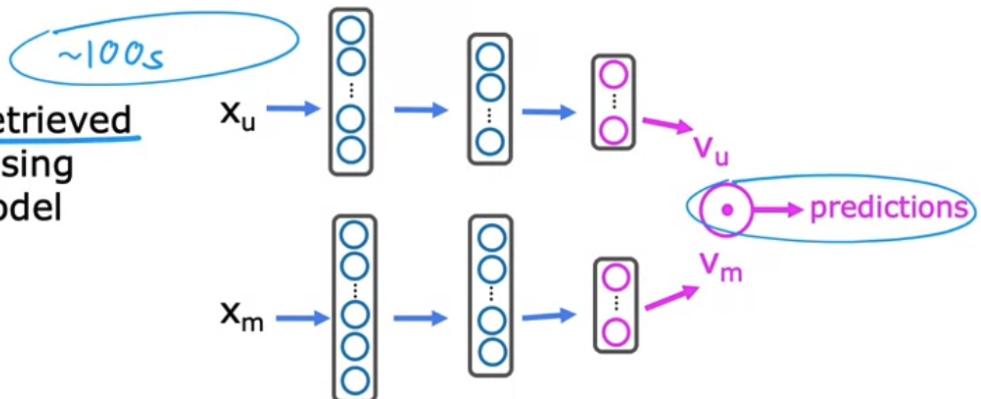
Retrieval:

- • Generate large list of plausible item candidates       $\sim 100s$ 
  - e.g.
  - 1) For each of the last 10 movies watched by the user, find 10 most similar movies  
 $\|v_m^{(k)} - v_m^{(i)}\|^2$
  - 2) For most viewed 3 genres, find the top 10 movies
  - 3) Top 20 movies in the country
- Combine retrieved items into list, removing duplicates and items already watched/purchased

## Two steps: Retrieval & ranking

Ranking:

- Take list retrieved and rank using learned model



- Display ranked items to user

## Retrieval step

- Retrieving more items results in better performance, but slower recommendations.
- To analyse/optimize the trade-off, carry out offline experiments to see if retrieving additional items results in more relevant recommendations (i.e.,  $p(y^{(i,j)}) = 1$  of items displayed to user are higher).

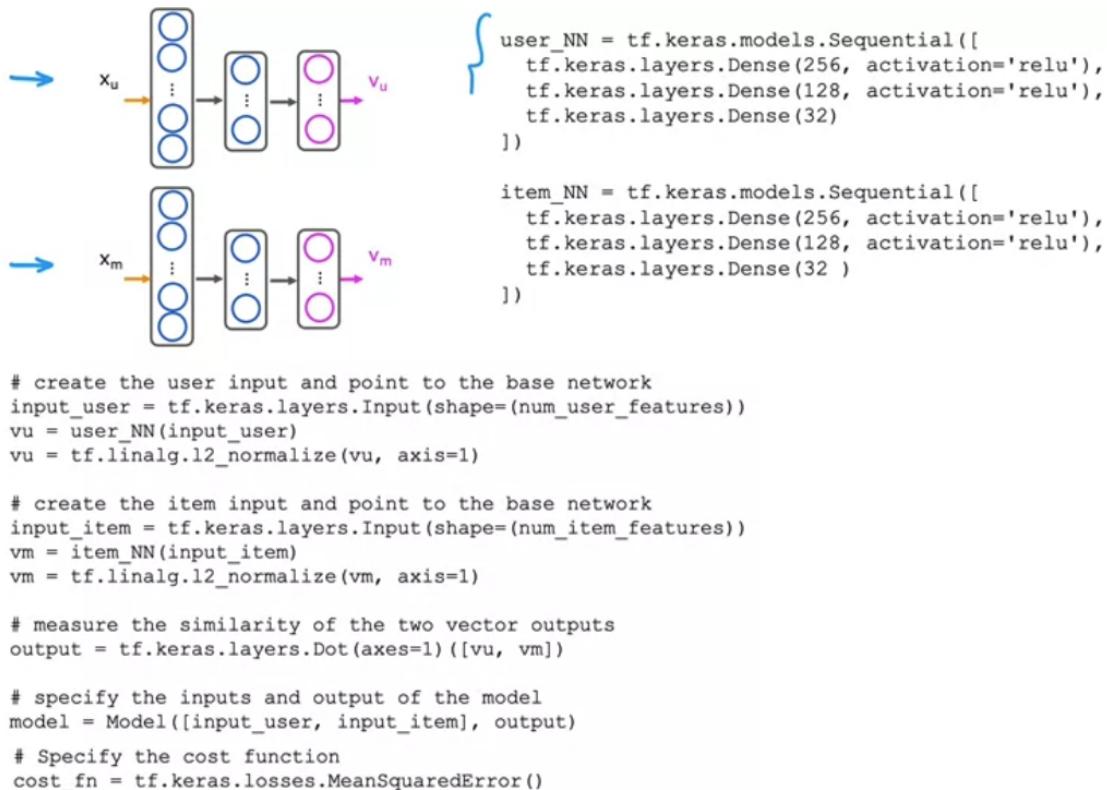
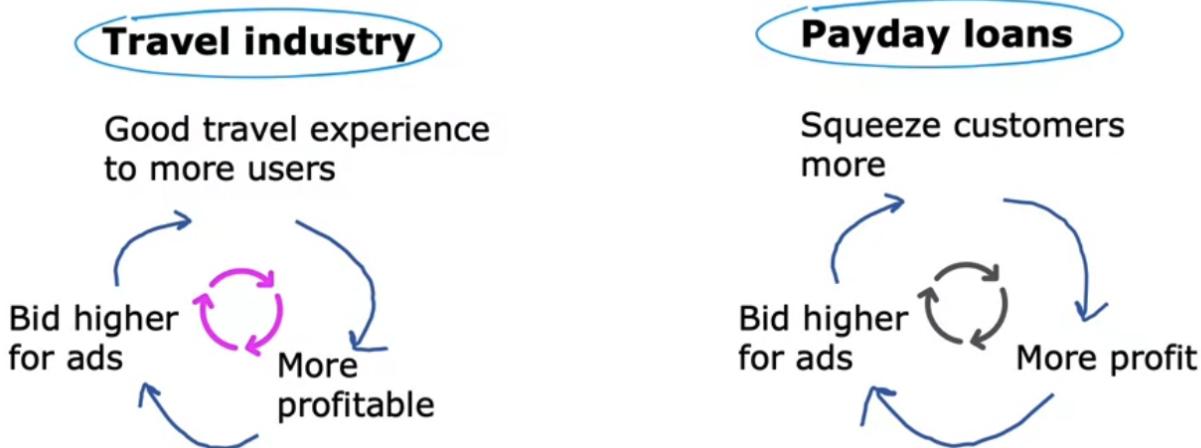
## What is the goal of the recommender system?

Recommend:

- Movies most likely to be rated 5 stars by user
- Products most likely to be purchased
- Ads most likely to be clicked on *+high bid*
- Products generating the largest profit
- Video leading to maximum watch time



## Ethical considerations with recommender systems



## Principal Component Analysis

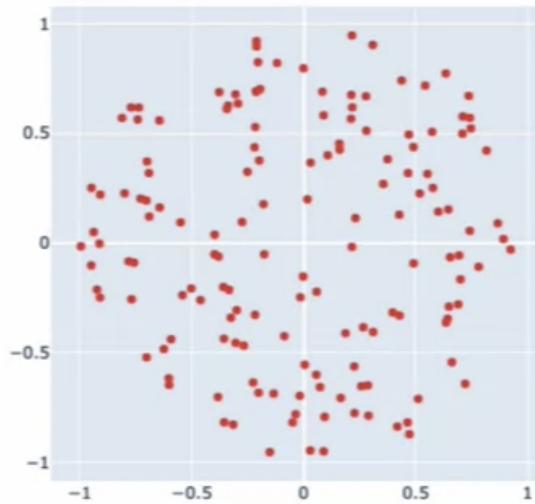
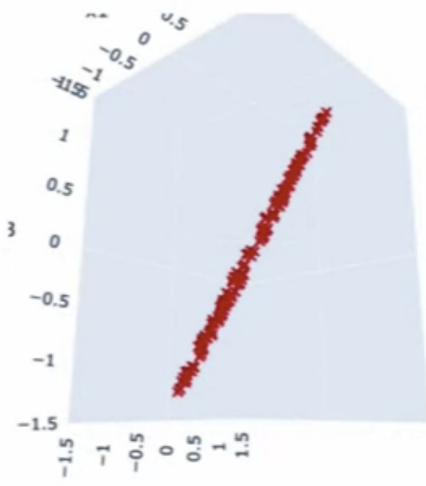
Used for: **visualization**, data compression (reduce storage or network), speed up training of supervised learning model

Reduces number of features by finding new axis and coordinates to keep useful information but reduce the number of features to describe that information

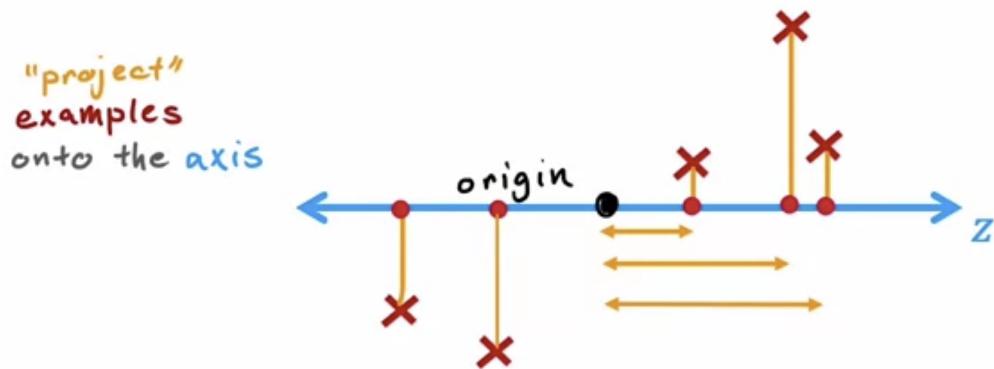
e.g. below all the points lie on a plane in 3D, so we can find new coordinates to represent them in 2D

The principal component is the axis that would give more variance to the data points, preserving the most amount of information of the original data.

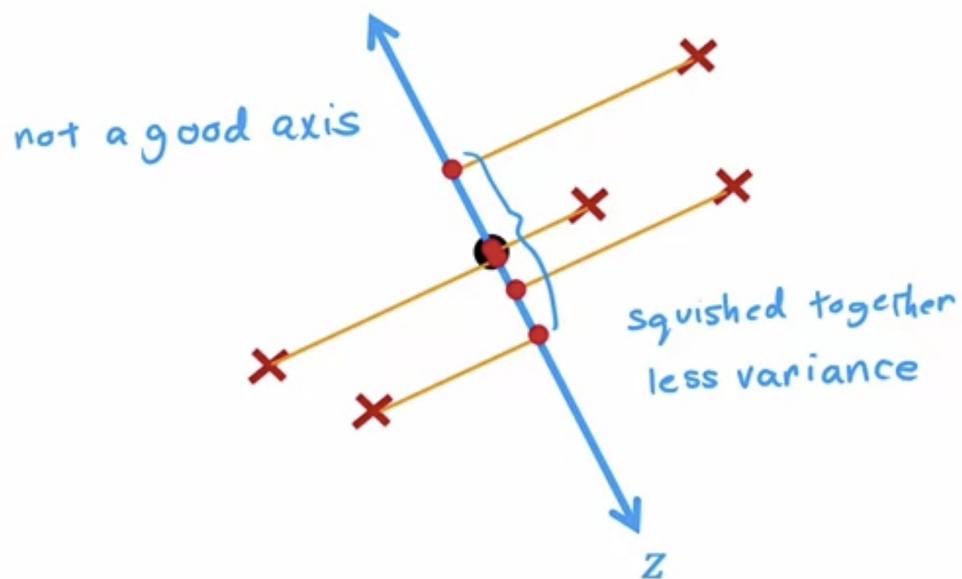
## From 3D to 2D



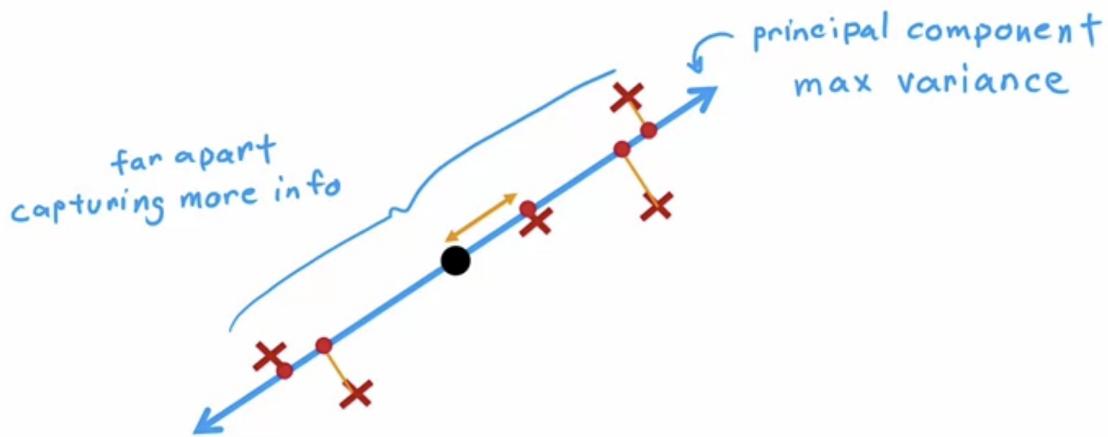
# Choose an axis



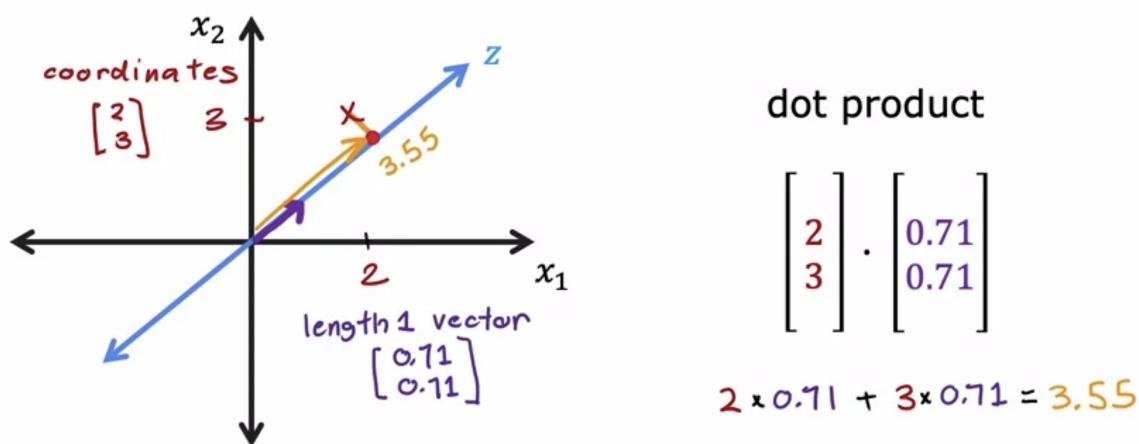
# Choose an axis



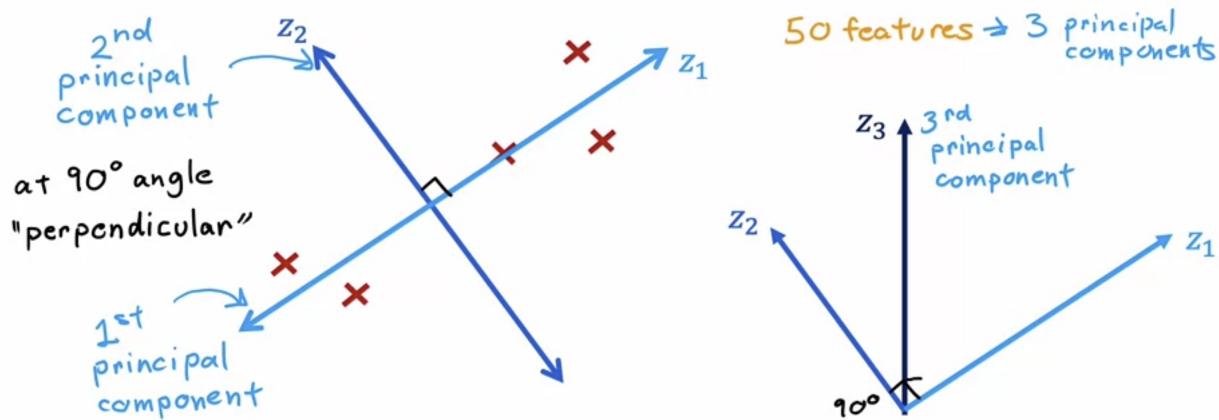
# Choose an axis



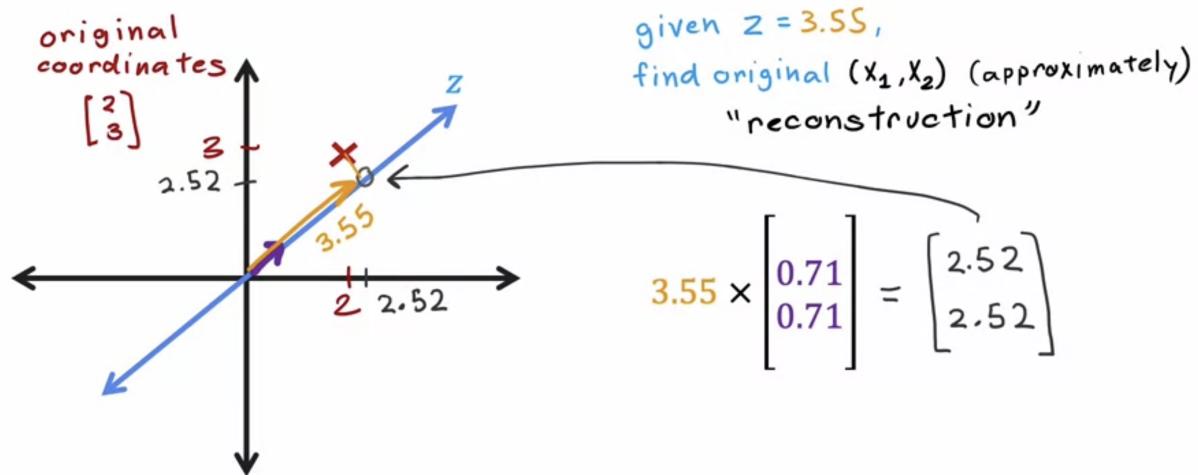
## Coordinate on the new axis



# More principal components



## Approximation to the original data



# PCA in scikit-learn

Optional pre-processing: Perform feature scaling



for visualization

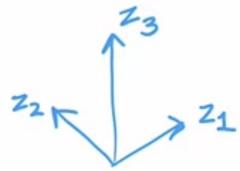
1. "fit" the data to obtain 2 (or 3) new axes (principal components)

fit includes mean normalization



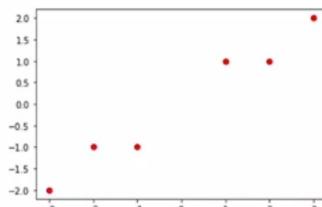
2. Optionally examine how much variance is explained by each principal component.

explained\_variance\_ratio



3. Transform (project) the data onto the new axes

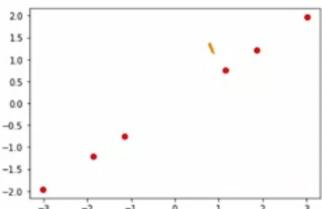
transform



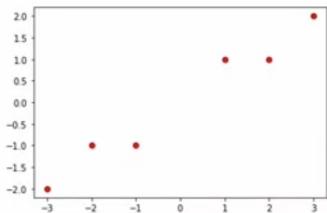
## Example

```
X = np.array([[1, 1], [2, 1], [3, 2],  
[-1, -1], [-2, -1], [-3, -2]])
```

```
pca_1 = PCA(n_components=1)  
pca_1.fit(X)  
pca_1.explained_variance_ratio_ 0.992  
X_trans_1 = pca_1.transform(X)  
X_reduced_1 = pca_1.inverse_transform(X_trans_1)
```



```
array([  
[ 1.38340578],  
[ 2.22189802],  
[ 3.6053038 ],  
[-1.38340578],  
[-2.22189802],  
[-3.6053038 ]])
```

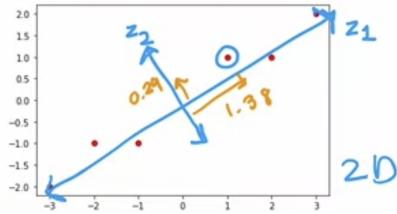


## Example

```
X = np.array([[1, 1], [2, 1], [3, 2],
              [-1, -1], [-2, -1], [-3, -2]])
```

2D

```
pca_2 = PCA(n_components=2)
pca_2.fit(X)
pca_2.explained_variance_ratio_
X_trans_2 = pca.transform(X)
X_reduced_2 = pca.inverse_transform(X_trans_2)
```



$z_1$        $z_2$

```
array([
    [ 1.38340578,  0.2935787 ],
    [ 2.22189802, -0.25133484],
    [ 3.6053038 ,  0.04224385],
    [-1.38340578, -0.2935787 ],
    [-2.22189802,  0.25133484],
    [-3.6053038 , -0.04224385]])
```