# Search Engines

COSC 494/594: Human-AI Interaction
Spring 2021 (CRN: 44874)

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

# Logistics

**Next Week: AI + Creativity + Interfaces**
→ Entirely online.

**Thanksgiving Week + Nov 30th**
→ No Class.

**Project 4: Releases Tomorrow**
→ Deadline: Nov 30th @ 11:59pm.

**Reminder: Nov 30th (CS523)**
→ Project Reports Due
  → Submission: Canvas (Applied AI Project)
  → Late Assignment Policy applies.

**Final Exam: Online**
→ Similar format to midterm.

| | | | |
|---|---|---|---|
| 9 | Oct 12 | *Reasoning Under Uncertainty* | |
| | Oct 14 | *Bayes Nets* | P3 Released |
| 10 | Oct 19 | Engineer's Day | |
| | Oct 21 | *Hidden Markov Models* | |
| | | *Guest Lecturer: Zhixiu Lu* | |
| 11 | Oct 26 | *Statistical Learning* | |
| | Oct 28 | *Markov Decision Processes* | |
| 12 | Nov 2 | *Reinforcement Learning* | P3 Due |
| | Nov 4 | *Adversarial Attacks* | |
| | | *Guest Lecturer: Zhuohang Li* | |
| 13 | Nov 9 | *Recommender Systems* | |
| | Nov 11 | *Search Engines* | P4 Released |
| 14 | Nov 16 | *Human-AI Interaction* | |
| | Nov 18 | *AI and Creativity* | |
| | | *Guest Lecturer: Rhema Linder* | |
| 15 | Nov 23 | Thanksgiving Week | |
| | Nov 25 | Thanksgiving Week | |
| 16 | Nov 30 | *AI, Eth nd Society* | P4 Due |
| | - | | |
| 17 | Dec 7 | Final Exam | |

# Revisiting “Professor’s Tune”

TRAINING

**Listening Feed:**

1. JS Bach (Like)
2. Whiplash (Like)
3. Owl City (Dislike)
4. Bo Burnham (Like)
5. Kanye West (Dislike)
6. Maylene (Like)

TEST

**Predictions**

1. Yeah Yeah Yeahs (Dislike)
2. Polyphia (Like)
3. Lil Nas X (Like)

**Let’s talk constraints about our knowledge.**

1. The data is a limited window of activity.
   → Six songs probably isn’t enough.

Dr. Williams’ Real Interests
→ Instrumental, e.g. Hans Zimmer
→ Strings, e.g. guitar, violin, lute, etc.
→ Technical, e.g. progressive

*“Your interests are kind-of soft”*
- Unknown, 2021.

# Revisiting "Professor's Tune"

TRAINING

**Listening Feed:**

1. JS Bach (Like)
2. Whiplash (Like)
3. Owl City (Dislike)
4. Bo Burnham (Like)
5. Kanye West (Dislike)
6. Maylene (Like)

TEST

**Predictions**

1. Yeah Yeah Yeahs (Dislike)
2. Polyphia (Like)
3. Lil Nas X (Like)

**Let's talk constraints about our knowledge.**

2. The data may not be generated by Dr. Willliams
→ Was Dr. Williams in control of the machine?
→ Were the "likes" implicitly collected?

3. The data may reflect contextual interests.
→ Christmas? Recommend Mariah Carey.

→ Programming? Recommend classical music.

→ Socializing? Recommending light jazz.

# What makes us trust the recommendations we receive?


"Buy GME!"
robinhood
9:41
Overview
Free Stock
Money Investing: $5,000
$50.000 (1.00%)
1D 1W 1M 3M 1Y ALL
Total $ SPY PINS BAC
Watchlist
SPY $287.54
PINS $26.50
QQQ $180.57


"Buy GME!"
robinhood
9:41
Overview
Free Stock
Money Investing: $5,000
$50.000 (1.00%)
1D 1W 1M 3M 1Y ALL
Total $ SPY PINS BAC
Watchlist
SPY $287.54
PINS $26.50
QQQ $180.57

# Learning Objectives

## 1. Introduce Recommender Systems

→ Purpose, issues, usage.

## 2. Understand User Modelling Approaches

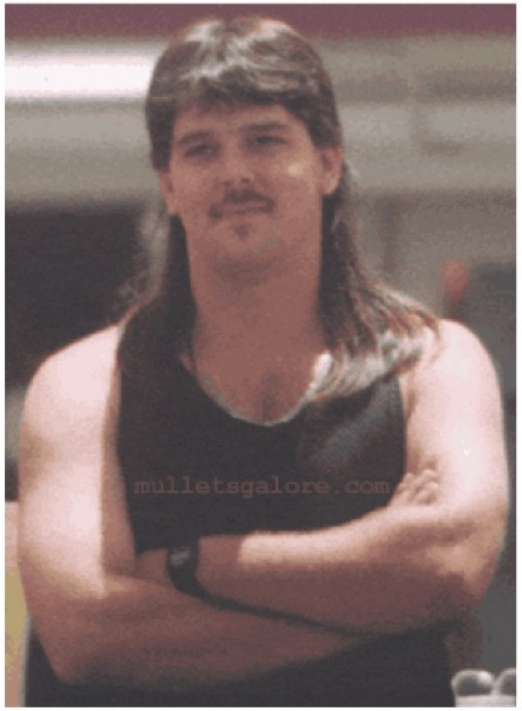
→ Content-based Filtering

→ Collaborative Filtering

## 3. Search Engine Architecture

→ Introducing Project 4

# Recommender Systems




**Customer A**

→ Buys Metallica CD

→ Buys Megadeth CD


**Customer B**

→ Does search on Metallica …

→ Recommender system suggests Megadeth from data collected about Customer A

# Recommendations


amazon

NETFLIX

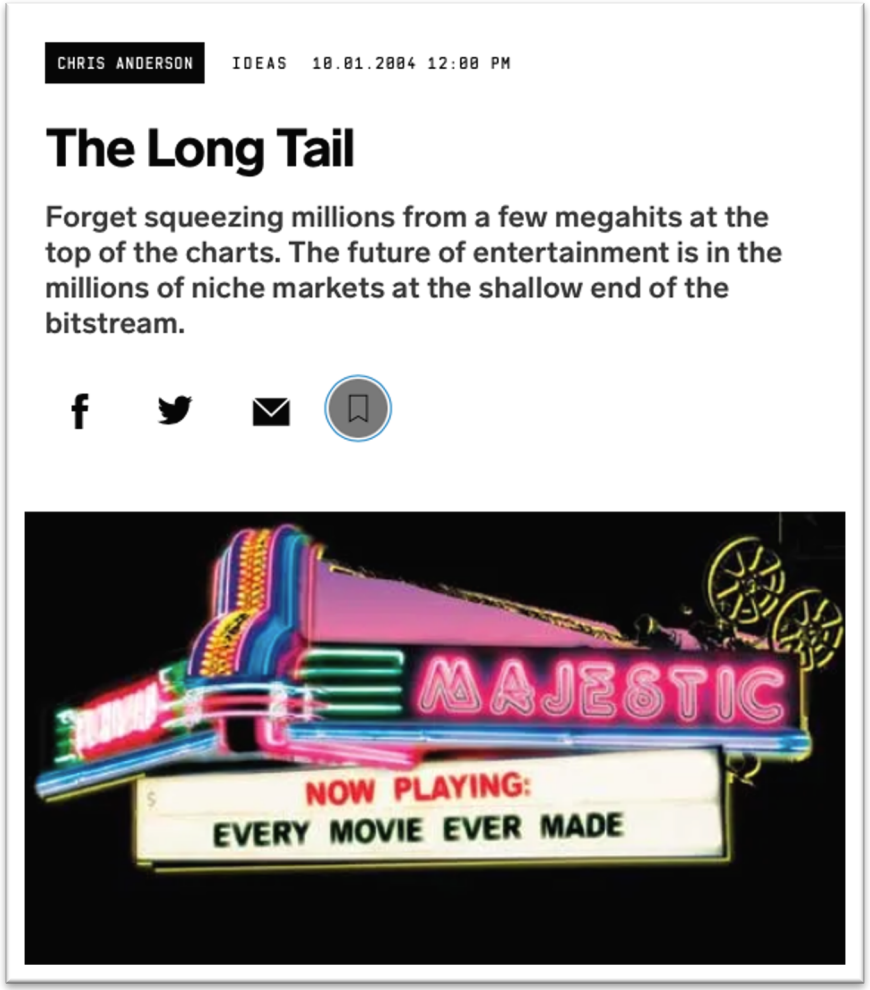
# Toward the Future

1. **Shelf space is a scarce commodity for traditional retailers.**
→ Also: TV networks, movie theaters

2. **The Web enables near-zero-cost dissemination of information.**
→ From scarcity to abundance.

3. **More choice leads to better filters.**
→ When we have more options, we can better discern boundaries
    → Recommendation engines

# The Long Tail



CHRIS ANDERSON IDEAS 10.01.2004 12:00 PM

## The Long Tail

Forget squeezing millions from a few megahits at the top of the charts. The future of entertainment is in the millions of niche markets at the shallow end of the bitstream.



**Thought Exercise:**

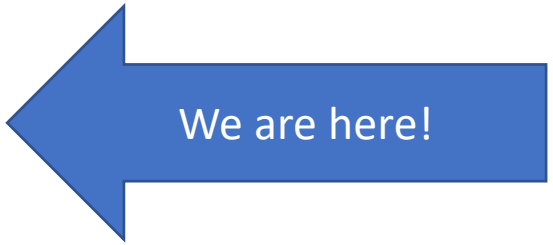
How do you find out about new things you might enjoy?

# Toward the Future

**Editorial and hand-curated recommendations**
→ List of favorites / essential items

**Simple Aggregates**
→ Top 10, Most Popular, Recent Uploads

**Tailored to Individual Users**
→ Amazon, Netflix



# A Formal Approach to User Modeling

***X*** = set of Customers
***S*** = set of Items

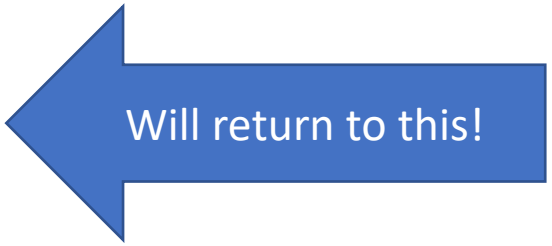
- **Utility Function**: ***u*: *X* × *S* → *R***
  - R = set of ratings
  - R is an ordered set
  - e.g., 0-5 stars, real number in [0,1]

# Utility Matrix

| | Avatar | LOTR | Matrix | Pirates |
|---|---|---|---|---|
| Alice | 1.0 | | 0.2 | |
| Bob | | 0.5 | | 0.3 |
| Carol | 0.2 | | 1.0 | |
| David | | | | 0.4 |

# Key Problems

**(1) Gathering "known" ratings for the matrix**
→ How to collect the data in the utility matrix.




**(2) Extrapolate unknown ratings from known ones**
→ Mainly interested in high unknown ratings

**(3) Evaluating extrapolation methods**
→ How to measure success/performance of recommendation methods.

# Gathering Ratings

**(1) Explicit Techniques**

→ Ask people to rate items.

→ Doesn't work well in practice… people can't be bothered.

→ Actual Alternative: You pay people to rate items. (Crowdsource it.)

**(2) Implicit Techniques**

→ Learn ratings from user actions, e.g. purchase implies high rating.

→ … What about low ratings?

# Extrapolating Utilities

**Key Problem:** Utility matrix ***U*** is sparse.

→ Most people have not rated most items.

→ The Cold Start Problem

- → New items have no ratings.
- → New users have no history.

$$\begin{pmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{pmatrix}$$

**Three Approaches to Recommender Systems:**

1. Content-based Recommenders
2. Collaborative Recommenders
3. Latent-factor Based Recommenders

# Extrapolating Utilities

**Key Problem:** Utility matrix ***U*** is sparse.

→ Most people have not rated most items.

→ The Cold Start Problem

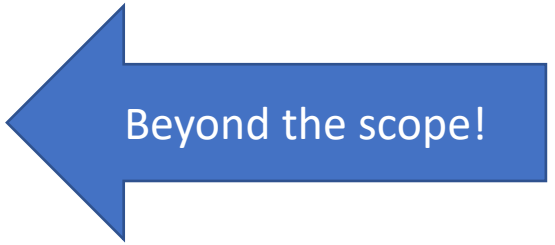
  → New items have no ratings.

  → New users have no history.

$$\begin{pmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{pmatrix}$$

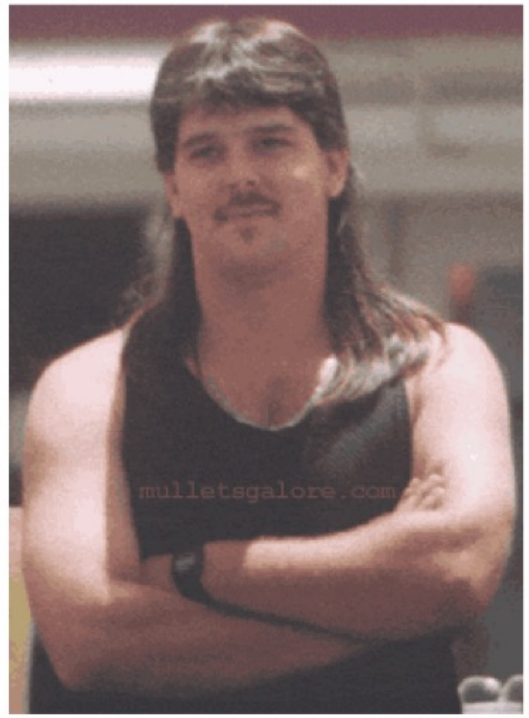
**Three Approaches to Recommender Systems:**

1. Content-based Recommenders
2. Collaborative Recommenders
3. ~~Latent-factor Based Recommenders~~



# 1. Content-based Recommender Systems

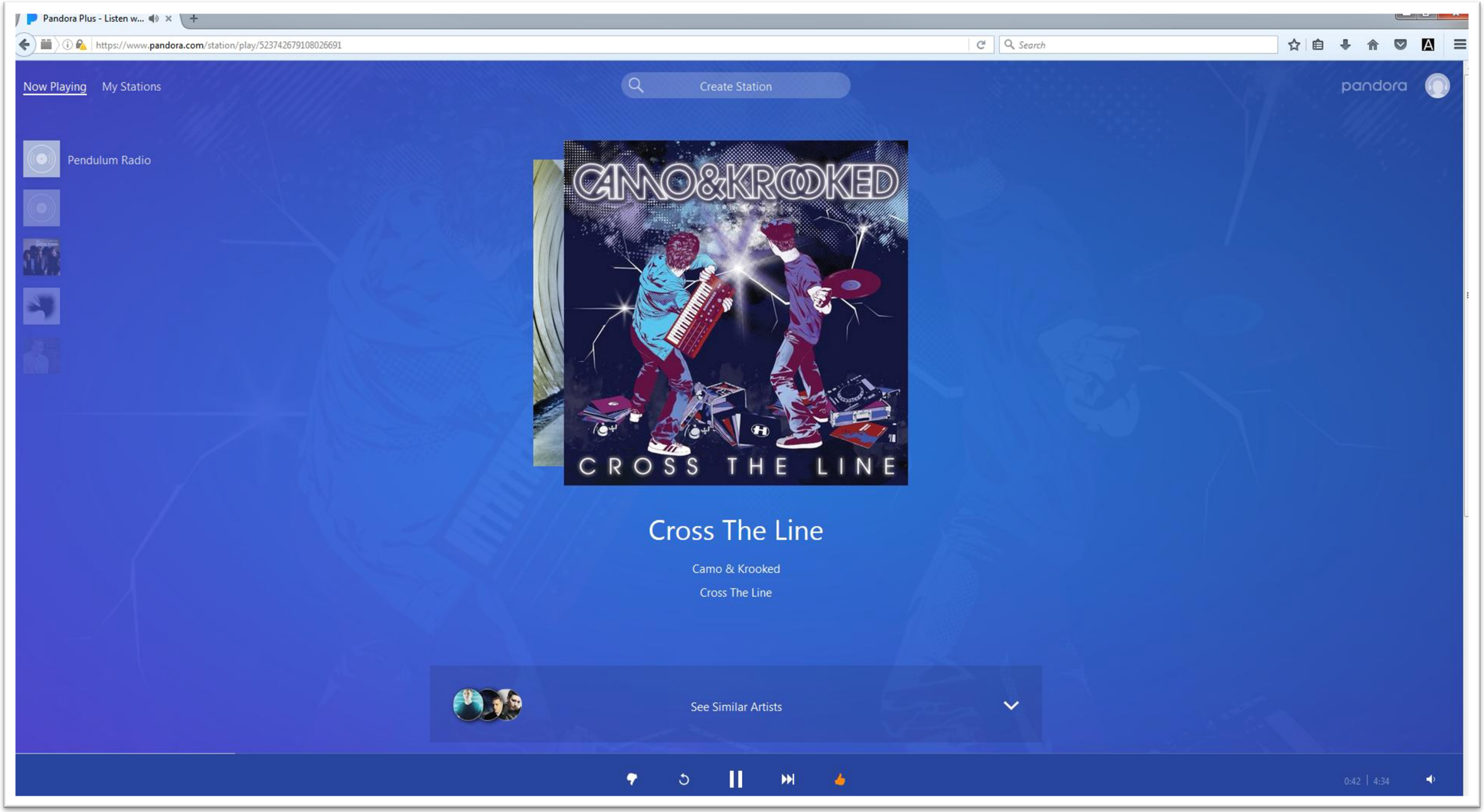
# Content-based Recommendations



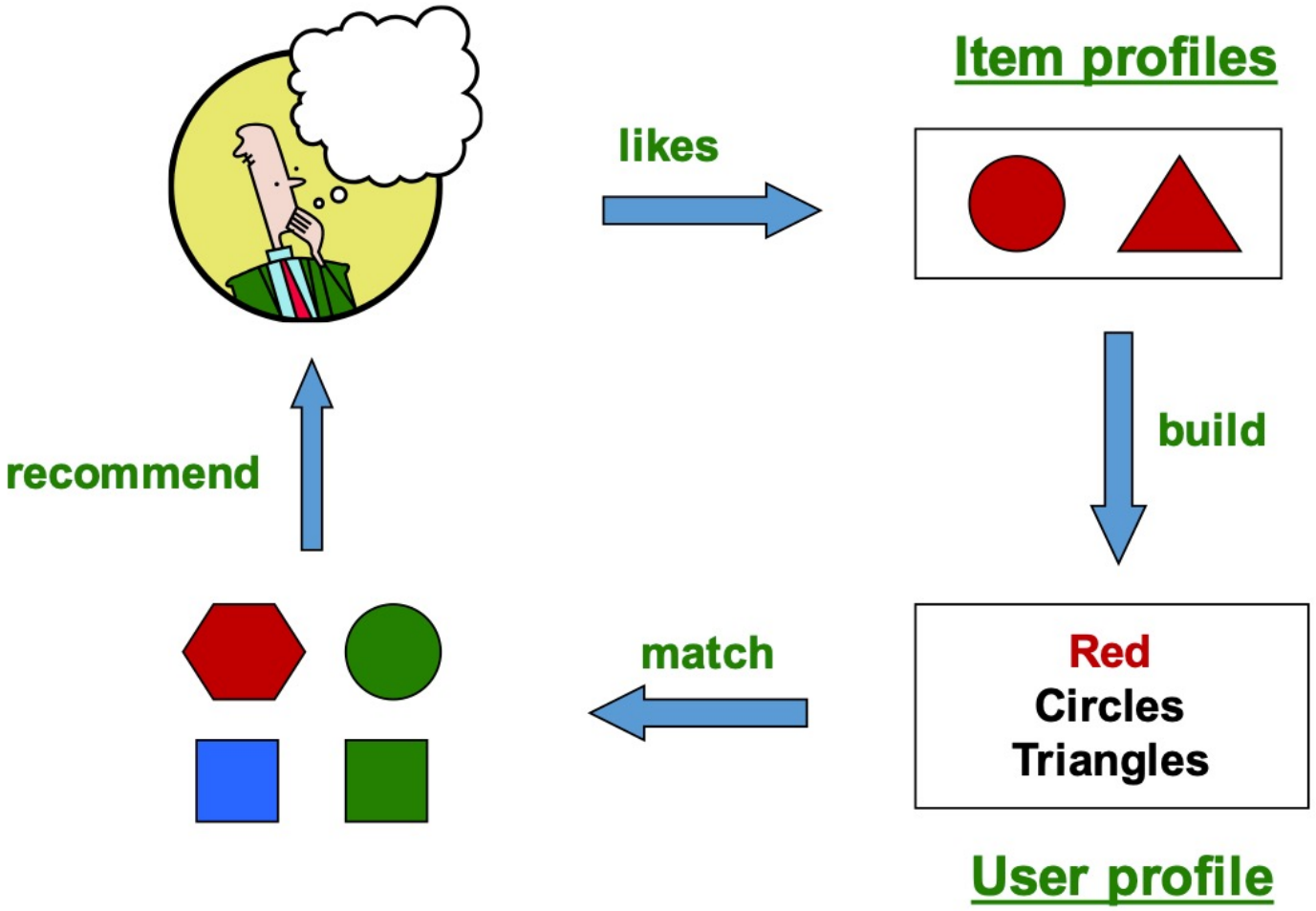

**Main idea:** Recommend items to customer x similar to previous items rated highly by x.

**Example:**

- Movie Recommendations
  - Recommend movies with the same actor(s), directors, genre, …
- Websites, blogs, news
  - Recommend other sites with “similar content”


Pandora Plus - Listen w...
https://www.pandora.com/station/play/523742679108026691
Search
Now Playing
My Stations
Create Station
pandora
Pendulum Radio
CAMO&KROOKED
CROSS THE LINE
Cross The Line
Camo & Krooked
Cross The Line
See Similar Artists
0:42 | 4:34

# Content-based Recommendations



# Item Profiles

**Main idea:** For each item, create a profile.

**Profile is a set (vector) of features.**

- **Movies:** author, genre, director, actors, year
- **Text**: Set of "important" words in document.

| | Actor A | Actor B | ... | Johnny Depp | Comic Genre | Spy Genre | Pirate Genre | Avg Rating |
|---|---|---|---|---|---|---|---|---|
| Movie X | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 3 |
| Movie Y | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 4 |

**How do you pick important features?**

- **Similarity Metrics**

# User Profiles

**We want a vector with the same components/dimensions as items.**

- Could be 1's representing user purchases.
- Or arbitrary numbers from a rating.

**User profile is an *aggregate* of items.**

- Average (weighted?) of rating item profiles.

| | Natalie Portman | Actor A | Actor B | ... | |
|---|---|---|---|---|---|
| User U | 0.2 | .005 | 0 | 0 | ... |

# Using Profiles to Predict and Recommend

**User and item vectors have the same components/dimensions.**

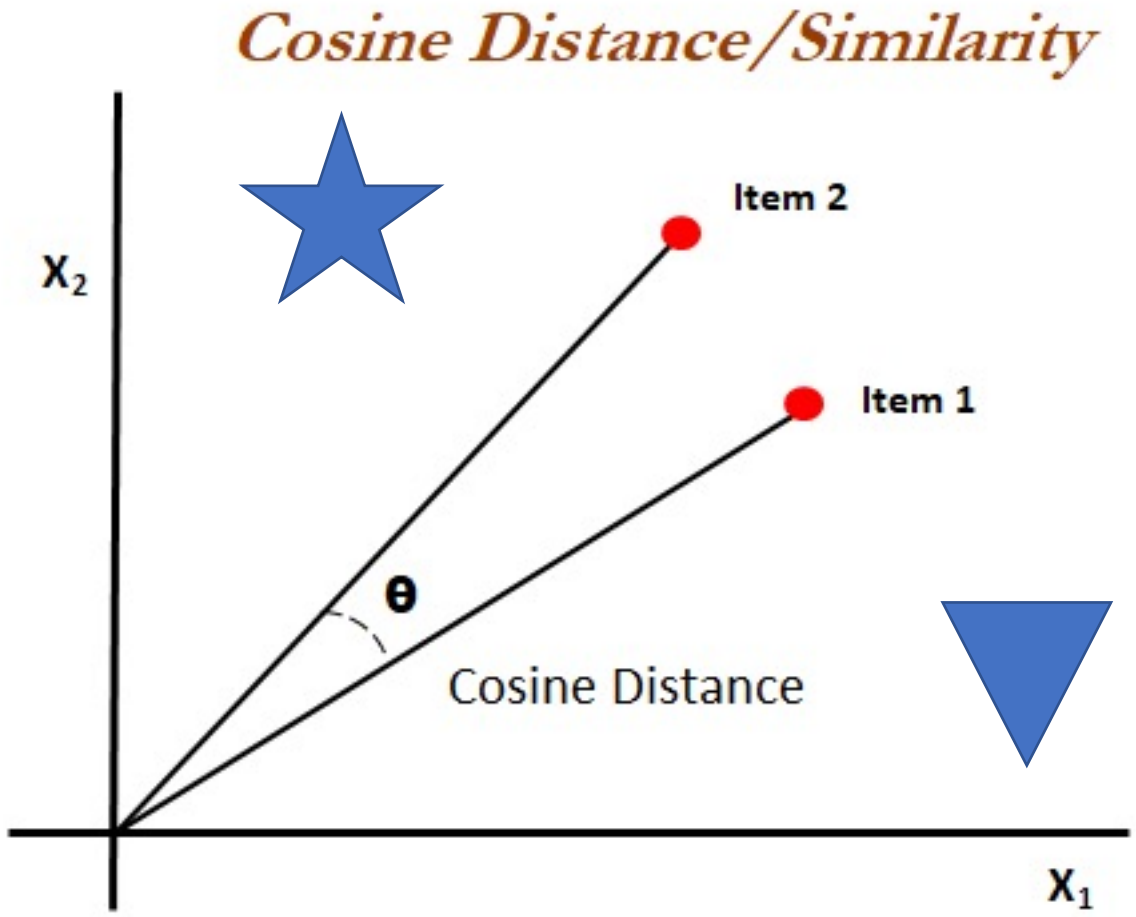
- Action: Recommend the items whose vectors are most similar to the user vector.

**Surprisingly easy to compute with Cosine Distance.**

- Given user profile x and item profile i, :

$$\text{estimate } u(\boldsymbol{x}, \boldsymbol{i}) = \cos(\boldsymbol{x}, \boldsymbol{i}) = \frac{\boldsymbol{x} \cdot \boldsymbol{i}}{||\boldsymbol{x}|| \cdot ||\boldsymbol{i}||}$$

# Cosine Similarity




**Intuition:** Further apart = larger angle = less similar.

$$u(x, i) = \cos(x, i) = \frac{x \cdot i}{||x|| \cdot ||i||}$$

# Cosine Similarity

User and Item Vectors

x = {3, 2 , 0,  5}

i = {1, 0 , 0,  0}

1. Calculate x · i

→ (3 * 1) + (2 * 0) + (0 * 0) + (5 * 0) = 3

2. Calcualte || x ||

→ sqrt ( ($3^2$) + ($2^2$) + ($0^2$) + ($5^2$) ) = 6.16

3. Calculate || y ||

→ sqrt ( ($1^2$) + ($0^2$) + ($0^2$) + ($0^2$) ) = 1

4. Calculate cos(x,i)|

→ 3 / (6.16 * 1) = 0.49

# Pros and Cons

**+: No need for data on other users.**

- No cold start or sparsity issues.

**+: Able to recommend to unique tastes.**

**+: Able to recommend to unpopular items.**

- No first-rater problem.

**+: Able to provide explanations.**

- Can provide explanations of recommended items by listing content-features that caused an item to be recommended.

**-: Finding the appropriate features is hard.**

- eg. Images, movies, music

**-: Recommendations for New Users**

- How to build a user profile?

**-: Overspecialization**

- Never recommends items outside user's content profile.
- People might have multiple interests

- **Unable to exploit quality judgements of other users.**

# 2. Collaborative Filtering with Recommender Systems

# Collaborative Filtering

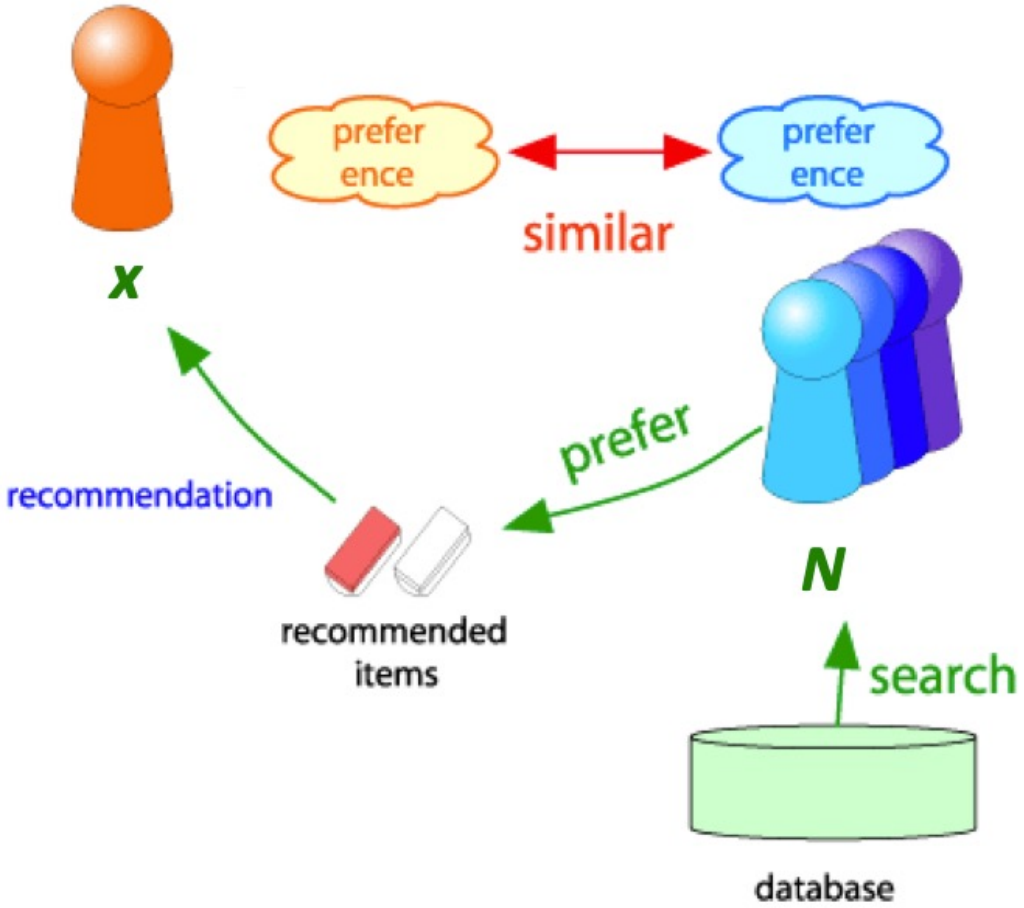

**Main idea:** Expand recommendation based on ratings from other users.

**Consider user *X*.**

- Find set of N of other users who ratings are "similar" to X's ratings.
  - Estimate X's ratings based on ratings of users in *N*.

In other words, …
Make a prediction about possible items X might like.

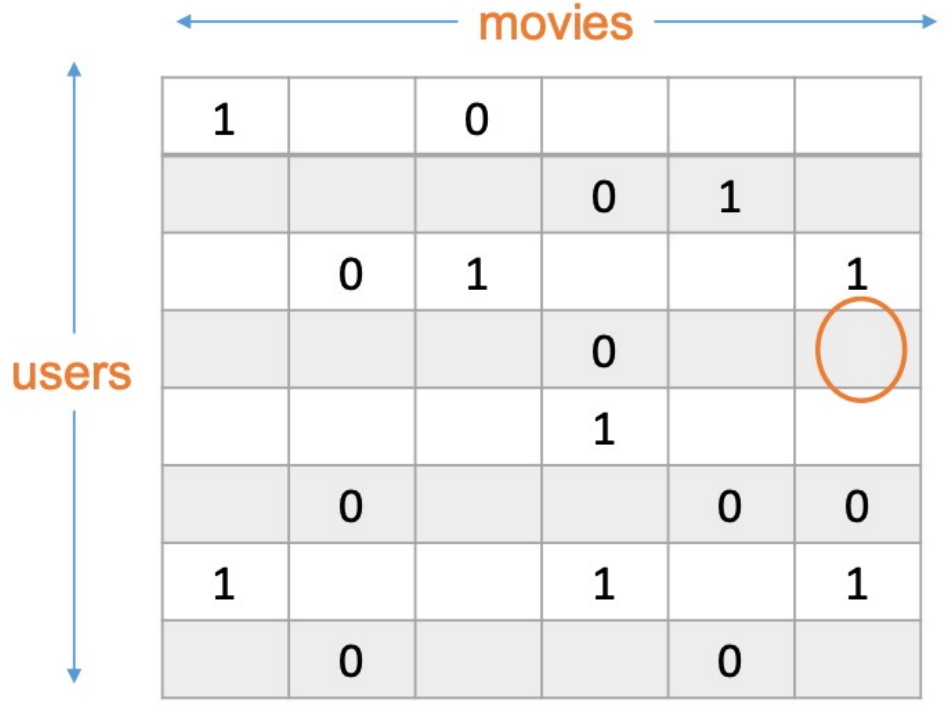
# Collaborative Filtering




Modeling User Preferences

# Collaborative Filtering

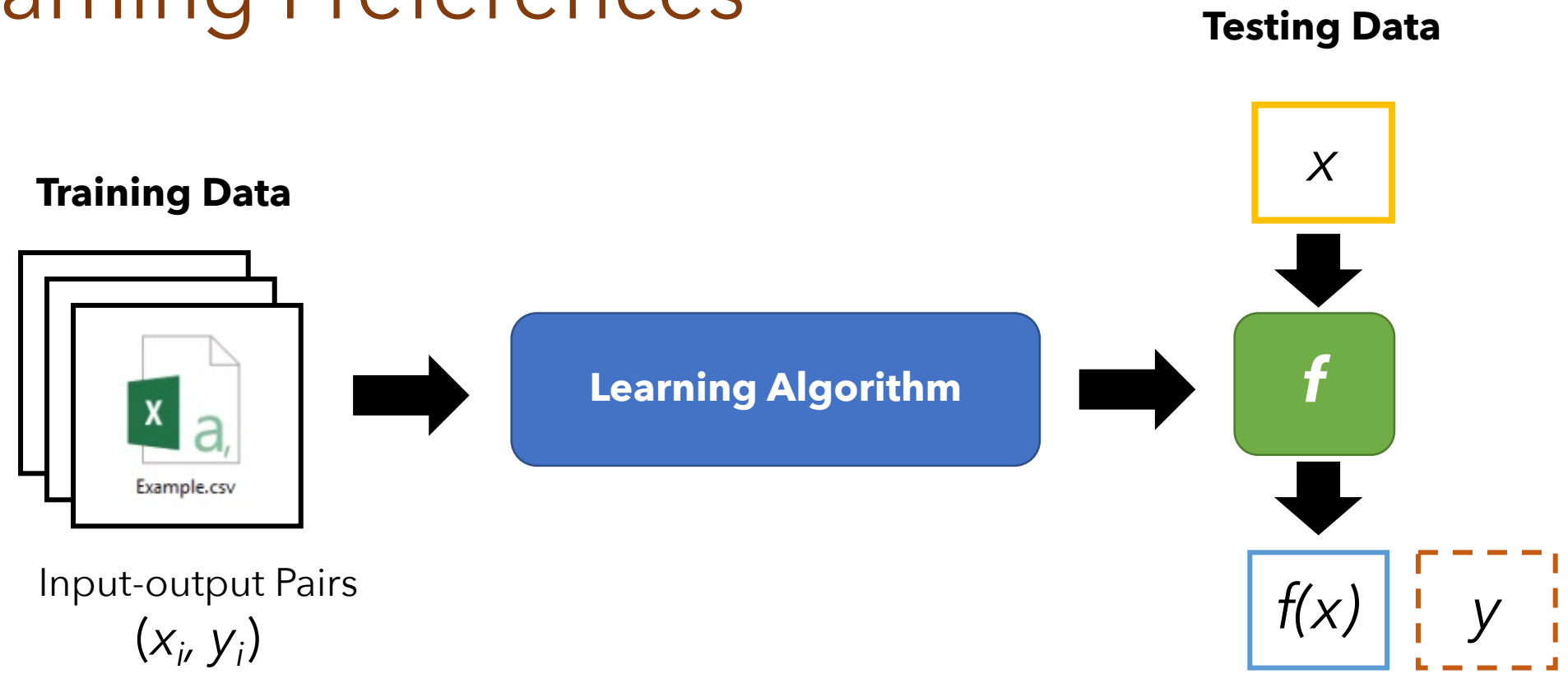
**Goal**: Predict possible items that will be rated positively.




Say we have an existing set of movie ratings for each user. 1 = Good, 0 = Bad

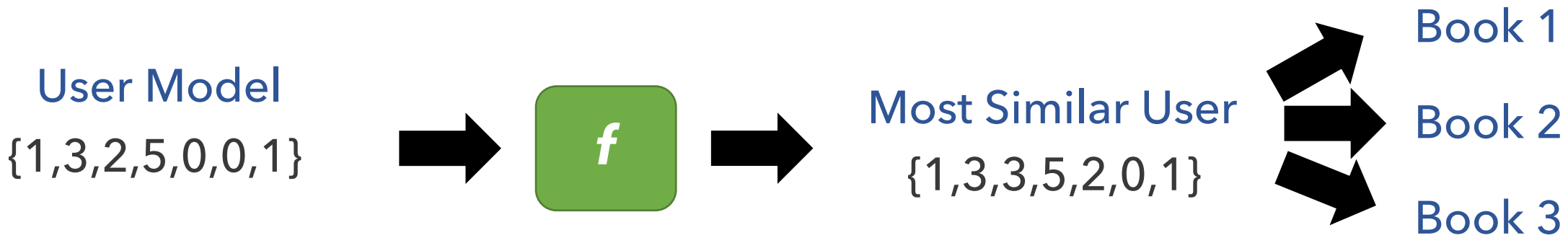
How many factors determine preference?
→ Key challenge of user modelling.

# Learning Preferences



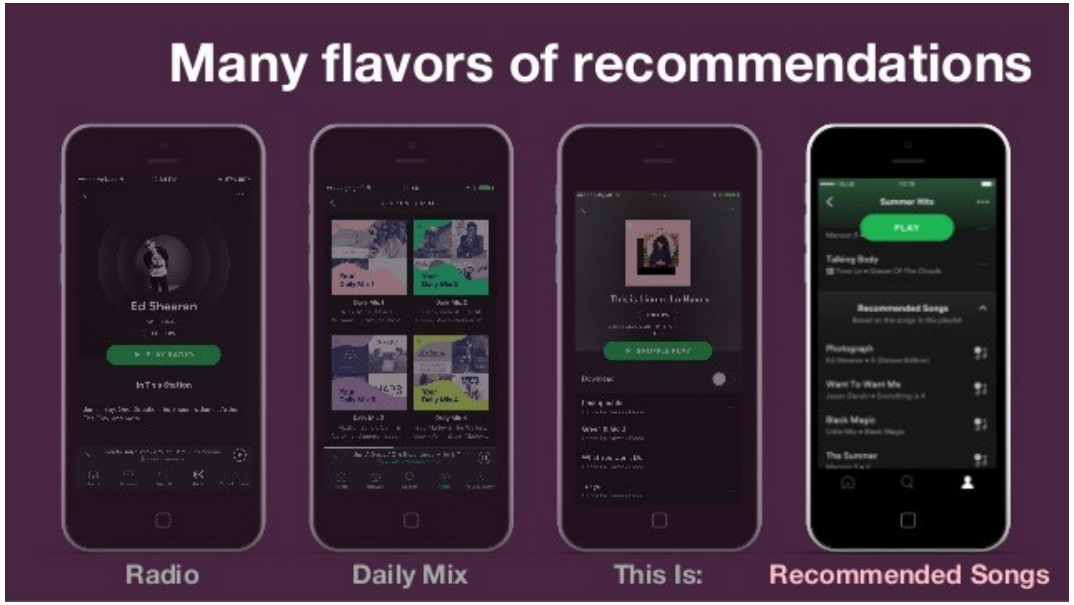

**Major Assumption:** You have access to $y_i$, (i.e. output variables).

# Using a Trained Model




Learning techniques learn **a function** that maps relations between users to make ties between preferences.

# Collaborative Filtering




**Summary**

→ They surround us!

→ Devices, applications, appliances …

→ Explaining recommendations.

→ Primarily an issue with collaborative filtering.

→ Recommenders are People-Driven.

→ What do you do when you don't have a userbase?

→ Alternatives to crowdsourcing.

# Search Engines as Recommenders

# Search Engines

**General Overview**

→ Search engines are arguably the most widely used AI system in society.

→ The only reason people don't them? Lack of internet access.

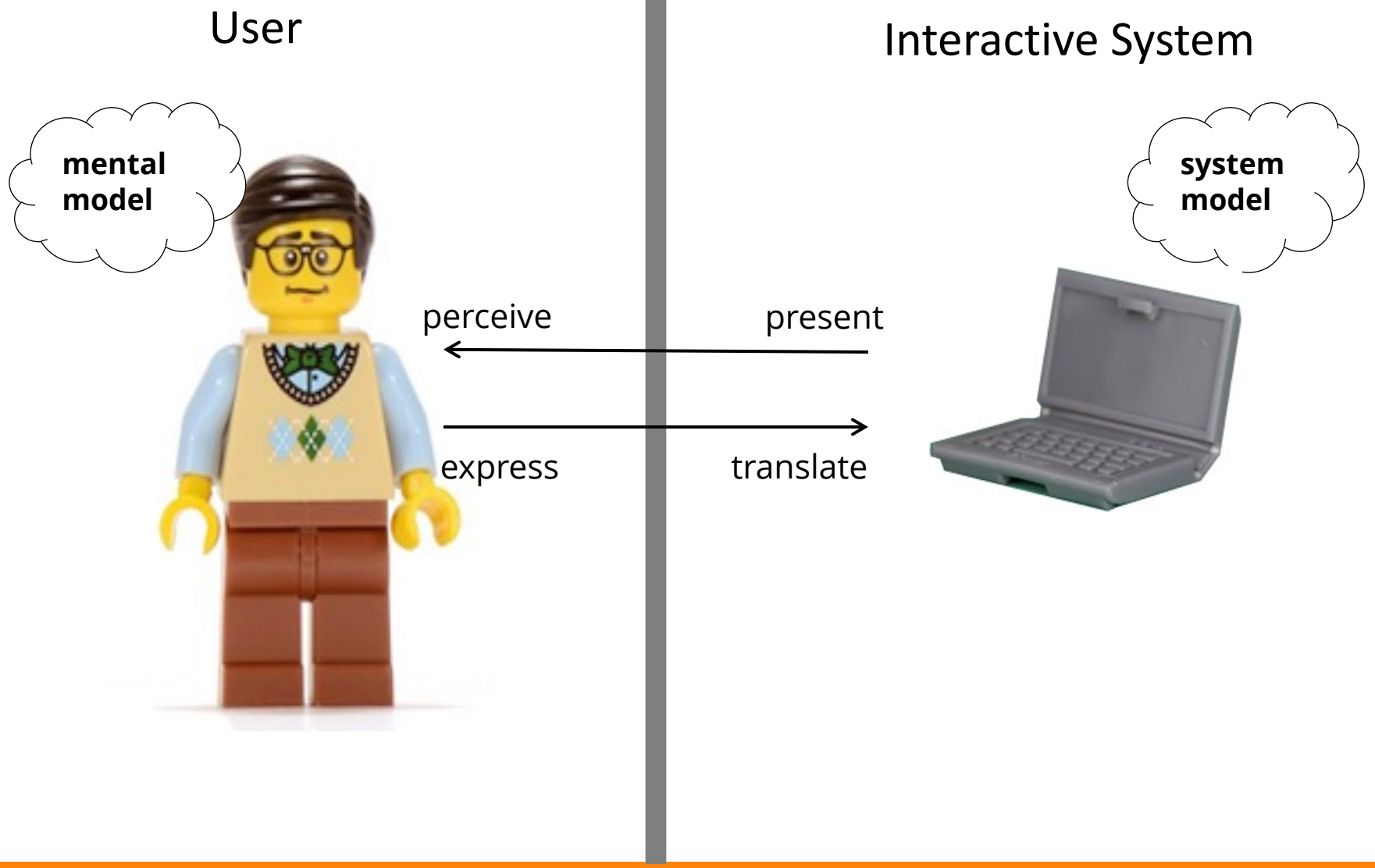
→ Search engines alleviate the pain of finding information.

→ Often text focused, but caters to media, generally, e.g. images.

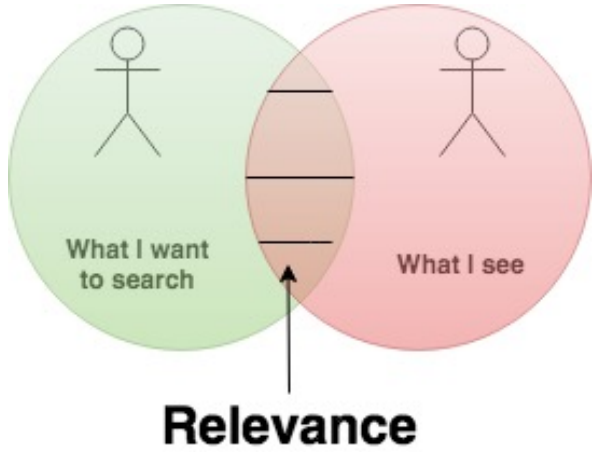
→ Search engines come in multiple flavors.

Accurate searches for general queries? → Google, Bing, etc.
Privacy-preserving search? → Duckduckgo.
Voice-enabled search? → Amazon's Alexa

# Mental Models



# Search Engines: Relevance




All search engines operate on the basis of three key steps:

1. Extract information from webpages.
2. Determine similarity between webpages.
3. Facilitate a ranking system for future queries.

Search engines can be thought of as learning models that predict relevance.

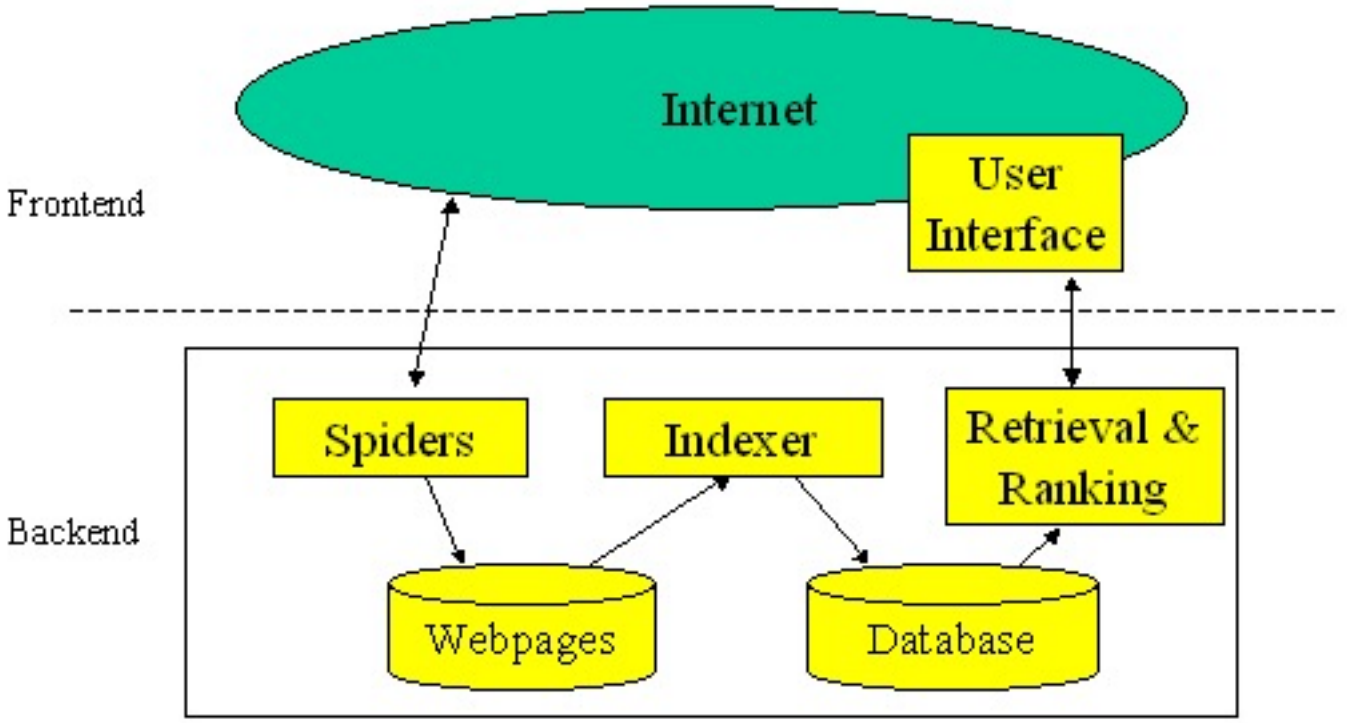
→ Can use similar metrics for measuring performance.

→ Precision, recall, accuracy, sensitivity, etc.

**Example:** Use "EECS UTK" as a query for Google.

→ We might have some notion of what is *truly* relevant and *truly* irrelevant.

# Search Architectures




**Components**

Frontend (Visible)

→ Search Interface

Backend (Invisible)

1. Web Crawler (Spider)
2. Ranking System
3. Retrieval System

**Note:** Database is ignored for simplicity.

# Component 1: Web Crawler

## Understanding Web Crawlers

Programs that automatically navigate the web in an intentional fashion.

→ The purpose of a crawler is to:
  → (1) extract content from webpages (e.g., text).
  → (2) collect links from the webpage to facilitate additional crawling.

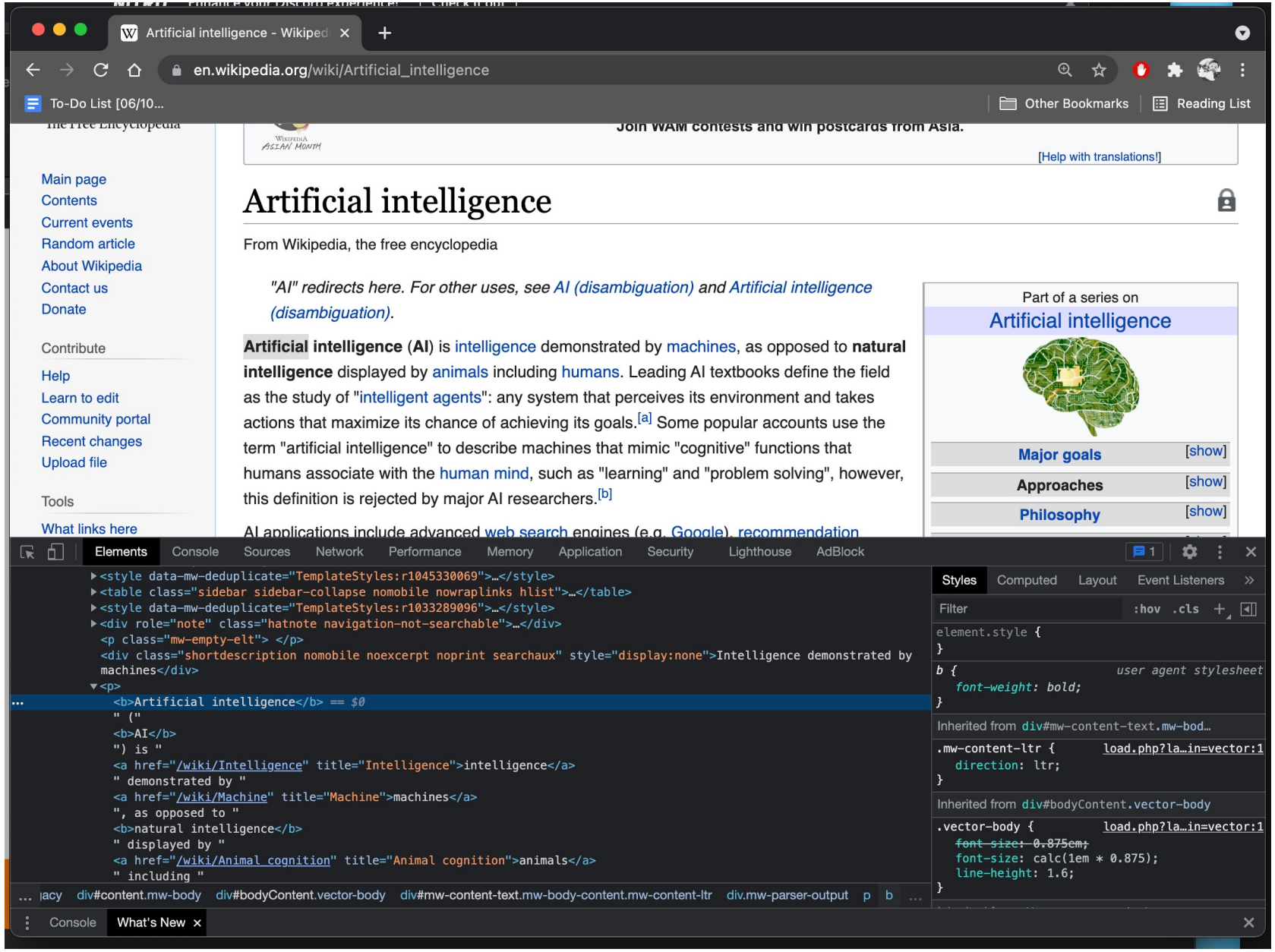
All search engines have crawlers.

→ Google's "Googlebot"
→ Microsoft's "Bingbot"

Crawlers have configurable parameters.

→ What site do you start crawling from? (i.e., the root)
→ How many times do you hop from the root? (i.e., depth)

**Example Task:**
Extract links and text information from Wikipedia.



**BeautifulSoup4 (4.8.0)**

→ A library for crawling in Python.

```python
# Requests + BeautifulSoup
import requests
from bs4 import BeautifulSoup
from urllib.request import Request, urlopen
from urllib.error import HTTPError

# Define a site, header request object, and create a request.
site = 'https://en.wikipedia.org/wiki/Artificial_intelligence'
hdr = {'User-Agent': 'Mozilla/5.0'}
req = Request(site, headers=hdr)

# The following try statement catches and prints any HTTP error that BeautifulSoup encounters when opening the website at the URL.
try:
    page = urlopen(req)
except HTTPError as err:
    print(err.code)
    # if err.code == 403:
    #     print(err.code)
    # else:

# Create an object to parse the HTML format
soup = BeautifulSoup(page, 'html.parser')

# Retrieve all popular news links (Fig. 1)
paragraphs = []
links = []
#for i in soup.find('div', {'class':'outer-wrap'}).f
for j in soup.find_all('div', {'class':'mw-parser-ou
    for i in j.find_all('p'):
        #print (i.text)
        paragraphs.append(i.text)

        # find each of the <a> elements to identify links.
        for k in i.find_all('a'):
            #print (k['href'])
            links.append(k['href'])

# Print a summary of findings.
print('Number of Paragraphs: ' + str(len(paragraphs)))
print('Number of Links: ' + str(len(links)))
```

```
(base) alex@MacBook-Pro-5:~/UTK-GDrive/UTK-Te
ts/Project5/examples$ python crawl.py
Number of Paragraphs: 100
Number of Links: 789
```

# Component 2: Ranking System

## Understanding Ranking System

Components that (in our case) post-process collected data and rank it.

→ Data on the web is visually structured, but computationally messy.

### Step 1: Cleaning the Data

→ Most of the time, this means removing characters that are meaningless to the task of assessing relevance.

### Step 2: Assessing the Similarity of the Data

→ Once cleaned, the data can be used to compute similarity metrics.

→ **In order to compute similarity, the data must be vectorized.**

→ **To facilitate vectorization, we will now introduce TF-IDF vectorization.**

Note: This is *one* way of calculating similarity.

Term Frequency (TF) = Measure of frequency.
→ `the number of observed occurrences.`

Inverse Document Frequency (IDF) = Measure of how much information is provided.
→ `log ( Number of Documents / Number of Observed Word)`

TF-IDF = TF x IDF.

## Example

**Collected Data**
$D_1$ = The sky is blue
$D_2$ = The sky is not blue

| | TF | | IDF | TF-IDF | |
|---|---|---|---|---|---|
| **Word** | **$D_1$** | **$D_2$** | | **$D_1$** | **$D_2$** |
| The | 1 | 1 | log(2/2) | 0 | 0 |
| sky | 1 | 1 | log(2/2) | 0 | 0 |
| is | 1 | 1 | log(2/2) | 0 | 0 |
| blue | 1 | 1 | log(2/2) | 0 | 0 |
| not | 0 | 1 | log(2/1) | 0 | log(2) |

**Scikit-Learn (0.15.0)**
**Pandas (0.25.1)**
**NumPy (1.18.5)**

Project5 > examples > rank_and_retrieve.py > ...

```python
from sklearn.feature_extraction.text import TfidfVectorizer
import re, string
import pandas as pd
import numpy as np

# each of the folowing statements comes from one of the UTK EECS webpages.
docs=[
    "Min H. Kao, born in Chu-Shan, Taiwan, is Executive Chairman of Garmin Ltd., which designs and manufactures navigation and communication equipment. Kao co-founded Garmin in 1989 with the vision of enriching people's lives by bringing Global Positioning System (GPS) technology to the consumer market.",

    "Curious to learn more about where you would be living? Learn more about our vibrant and thriving community just minutes from hundreds of miles of natural greenways, the Great Smoky Mountain National Park, and a bustling downtown with new restaurants and shops popping up every year.",

    "The Min H. Kao Electrical Engineering and Computer Science Building is located on the corner of Cumberland Avenue and Estabrook Road. The 150,000-square-feet building holds offices, classrooms, laboratories, conference rooms, a 147-seat auditorium, and a sixth-floor terrace with stunning views of downtown Knoxville. It is also home to the Center for Ultra-wide-area Resilient Electric Energy Transmission Networks (CURENT), an NSF Engineering Research Center.",

    "By 1896, the program was gaining steam with 85 students pursuing electrical engineering degrees. The influx of students and the addition of new equipment expanded the university's electrical laboratory to its limits in Science Hall. The addition of Estabrook Hall to campus in 1899 made room for a new power plant in addition to experimental labs.",

    "Sophomores through seniors are assigned a professional advisor and faculty mentor in EECS. Students will need to meet with their professional advisor once a year to plan their schedule and get cleared for course registration. Faculty mentors provide guidance and advice on career and employment opportunities, graduate school, and EECS elective courses."
]

```

**Scikit-Learn (0.15.0)**
**Pandas (0.25.1)**
**NumPy (1.18.5)**

```
19  # [ RANKING STAGE ]
20
21  # Step 1: Data Cleaning
22  # Perform data cleaning on the dataset to text information that complicates relevance.
23
24  documents_clean = []
25  for d in docs:
26      # Lowercase all letters in the text.
27      d_temp = d.lower()
28      # Remove punctuations in the text.
29      d_temp = re.sub(r'[%s]' % re.escape(string.punctuation), ' ', d_temp)
30
31      documents_clean.append(d_temp)
32
33  # Step 2: Vectorize the documents
34  # Use the Scikit-learn built-in vectorizer.
35
36  # Instantiate the Tfidfvectorizer
37  tfidf_vectorizer=TfidfVectorizer()
38
39  # Send our docs into the Vectorizer
40  tfidf_vectorizer_vectors=tfidf_vectorizer.fit_transform(docs)
41
42  # Transpose the result into a more traditional TF-IDF matrix, and convert it to an array.
43  X = tfidf_vectorizer_vectors.T.toarray()
44
45  # Convert the matrix into a dataframe using feature names as the dataframe index.
46  df = pd.DataFrame(X, index=tfidf_vectorizer.get_feature_names())
47
48
```

# Component 3: Retrieval System + UI

**Understanding Retrieval System and the UI**

Component that takes a query and assesses its similarity to collect info.

→ The interface provide a mechanism for entering the query and seeing results.

## Retrieval is a 3-Step Process

1. The query is converted into a vector that mirrors the existing information vector.
2. Cosine similarity is calculated between the query vector and all possible documents.
3. The process concludes by returning the Top *N* documents with the highest similarity.

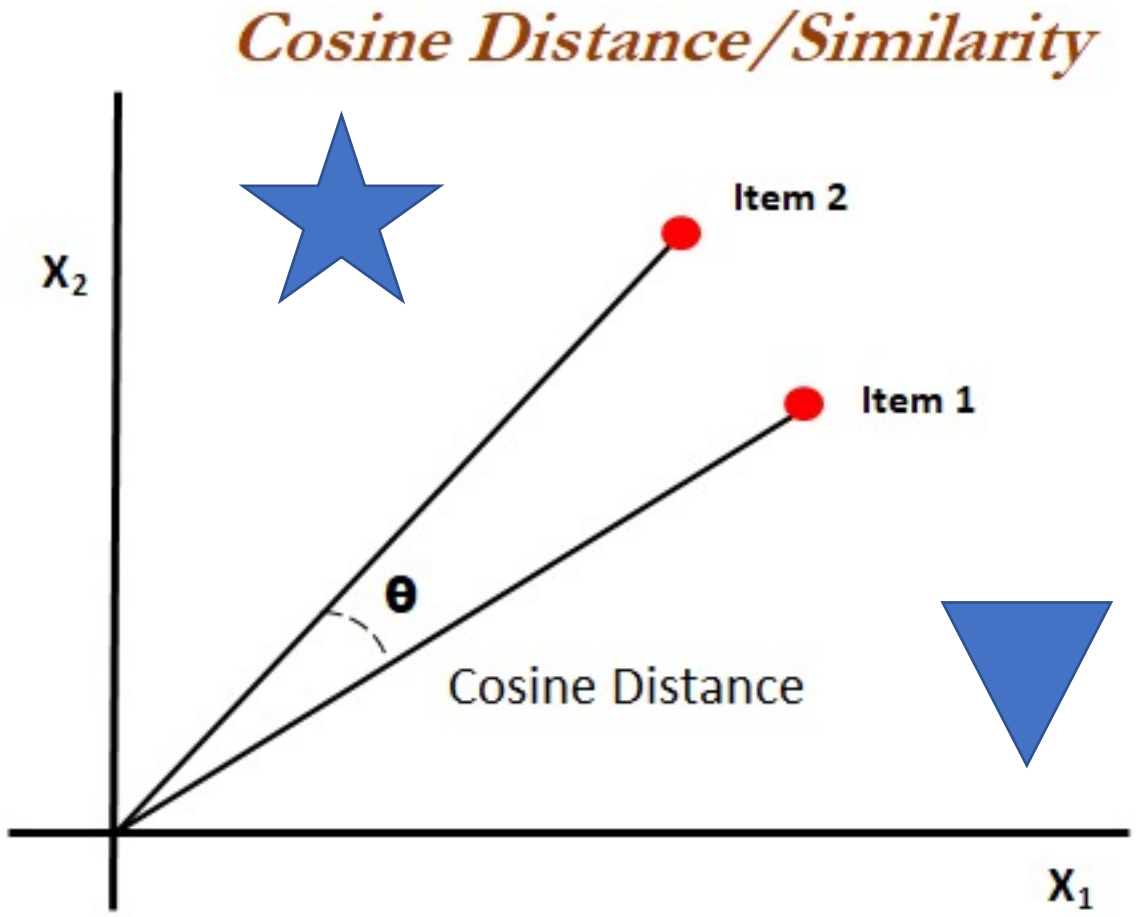
**User Interface**

→ Relevant documents are reported to the user in a readable fashion.

# Cosine Similarity

$x = \{3, 2, 0, 5\}$

$i = \{1, 0, 0, 0\}$




**Intuition:** Further apart = larger angle = less similar.

# Example

**Collected Data**

$D_1$ = The sky is blue
$D_2$ = The sky is not blue

| | TF | | IDF | TF-IDF (ω) | |
|---|---|---|---|---|---|
| **Word** | **$D_1$** | **$D_2$** | | **$D_1$** | **$D_2$** |
| The | 1 | 1 | log(2/2) | 0 | 0 |
| sky | 1 | 1 | log(2/2) | 0 | 0 |
| is | 1 | 1 | log(2/2) | 0 | 0 |
| blue | 1 | 1 | log(2/2) | 0 | 0 |
| not | 0 | 1 | log(2/1) | 0 | log(2) |

# New Query

$Q_1$ = not blue

**Vector Representation**

$Q_1 = \{0, 0, 0, 1, 1\}$

$D_1 = \{0, 0, 0, 0, 0\}$
$D_2 = \{0, 0, 0, 0, 1\}$

**Cosine Similarity Scores**

Nonzero indicates relevance.

$D_1 = 0.0$
$D_2 = 0.3$

**Compare Q1's vector to all D vectors.**

Note: Hypothetical values for demonstration.

**Scikit-Learn (0.15.0)**
**Pandas (0.25.1)**
**NumPy (1.18.5)**

```
48
49  # [ RETRIEVAL STAGE ]
50
51  query = 'program'
52
53  # Vectorize the query.
54  q = [query]
55  q_vec = tfidf_vectorizer.transform(q).toarray().reshape(df.shape[0],)
56
57  # Calculate cosine similarity.
58  sim = {}
59  for i in range(len(df.columns)-1):
60      sim[i] = np.dot(df.loc[:, i].values, q_vec) / np.linalg.norm(df.loc[:, i]) * np.linalg.norm(q_vec)
61
62  # Sort the values
63  sim_sorted = sorted(sim.items(), key=lambda x: x[1], reverse=True)
64
65  # Print the articles and their similarity values
66  for k, v in sim_sorted:
67      if v != 0.0:
68          print("[DOCUMENT "+str(
69
```

```
(base) alex@MacBook-Pro-5:~/UTK-GDrive/UTK-Teaching/C
ts/Project5/examples$ python rank_and_retrieve.py
[DOCUMENT 3] - (0.13)
(base) alex@MacBook-Pro-5:~/UTK-GDrive/UTK-Teaching/C
ts/Project5/examples$
```

# Project 4: A Search Engine in the Terminal


**General Overview (Deadline: Nov 30 @ 11:59pm)**

You're building a search interface for the EECS website.

→ Use the examples I've given you. Read the specification carefully.

# Next Time

AI and Creativity