Information Storage & Retrieval

Class 7: Evaluation & Learning to Rank

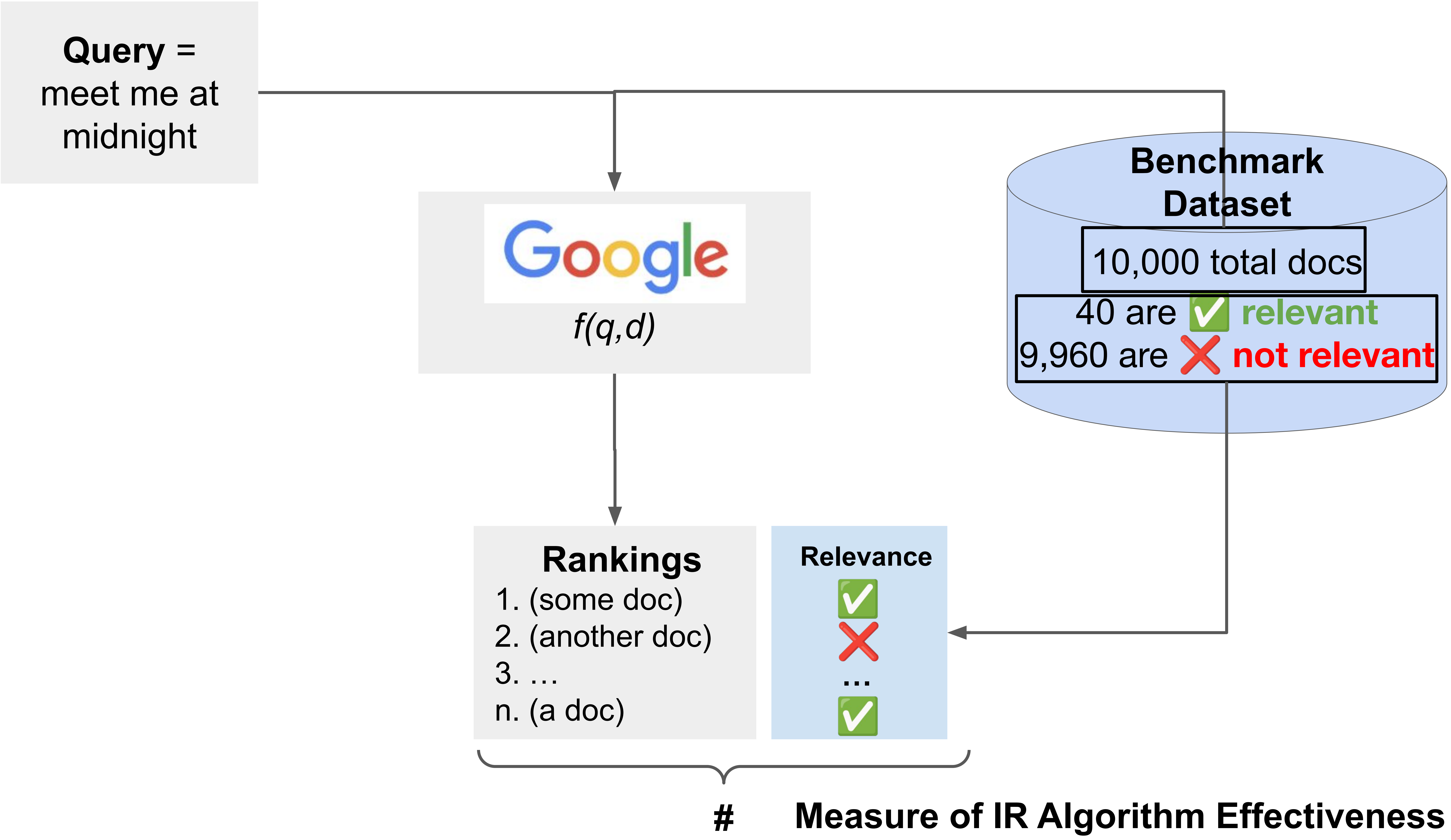
CSCE 670 :: Spring 2024
Texas A&M University
Department of Computer Science & Engineering
Prof. James Caverlee *and Maria Teleki* 🤠

Measuring Relevance

We need 3 things in our BENCHMARK DATASET:

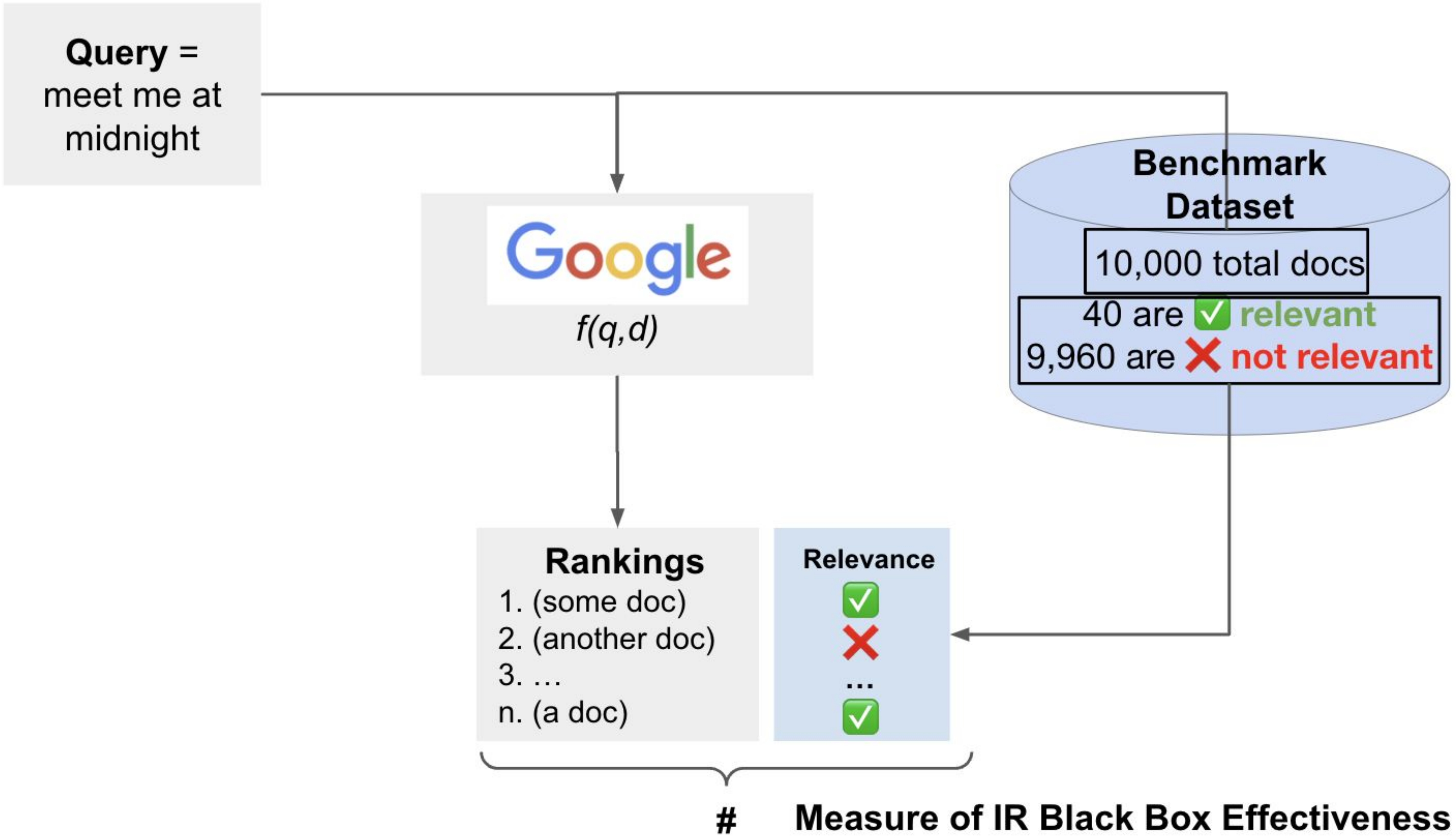
<table>
<tr><th>English</th><th>Math</th><th>Picture</th></tr>
<tr><td>1) A set of documents</td><td rowspan="3">$D = \{(d_i, q_j, r_{ij})\}$
d_i is a vector
q_j is a vector
$r_{ij} \in \{0,1\}$</td><td rowspan="3">D
<table><tr><th>Documents</th><th>Queries</th><th>Relevance</th></tr><tr><td>d_1</td><td>q_1</td><td>r_{11}</td></tr><tr><td>d_1</td><td>q_2</td><td>r_{12}</td></tr><tr><td>d_1</td><td>q_3</td><td>r_{13}</td></tr><tr><td>…</td><td>…</td><td>…</td></tr></table></td></tr>
<tr><td>2) A set of queries</td></tr>
<tr><td>3) A binary assessment of either ✅ Relevant or ❌ Non-Relevant for each query and each document</td></tr>
</table>

The Big Picture



Activity

With your group: What metrics did we learn about last time?



So far, our **evaluation** has been **offline**

We have mainly discussed **offline evaluation**, where we want to test a hypothesis (e.g., compare **new search engine X'** to **old search engine X**)

Assumption: we have a test collection of

- **docs** (representative of our collection),
- **queries** (that we hope are representative of what our users will ask), and
- **relevance judgments** (can be <u>expensive</u> to collect and noisy)

Let's talk about offline experiments…

Useful even in scenarios where you DO have access to **a production system**

– *e.g., internally at Google, Bing, Netflix, … You can just use historic data!*

Good for **comparing results**

– *e.g., I can compare my algorithm to your algorithm*

Challenge: do the results ***generalize*** to the **online** scenario?

Types of **Evaluation**

1. **Offline:** Usually with a BENCHMARK DATASET or using historical interactions from a production system (e.g., at Google)

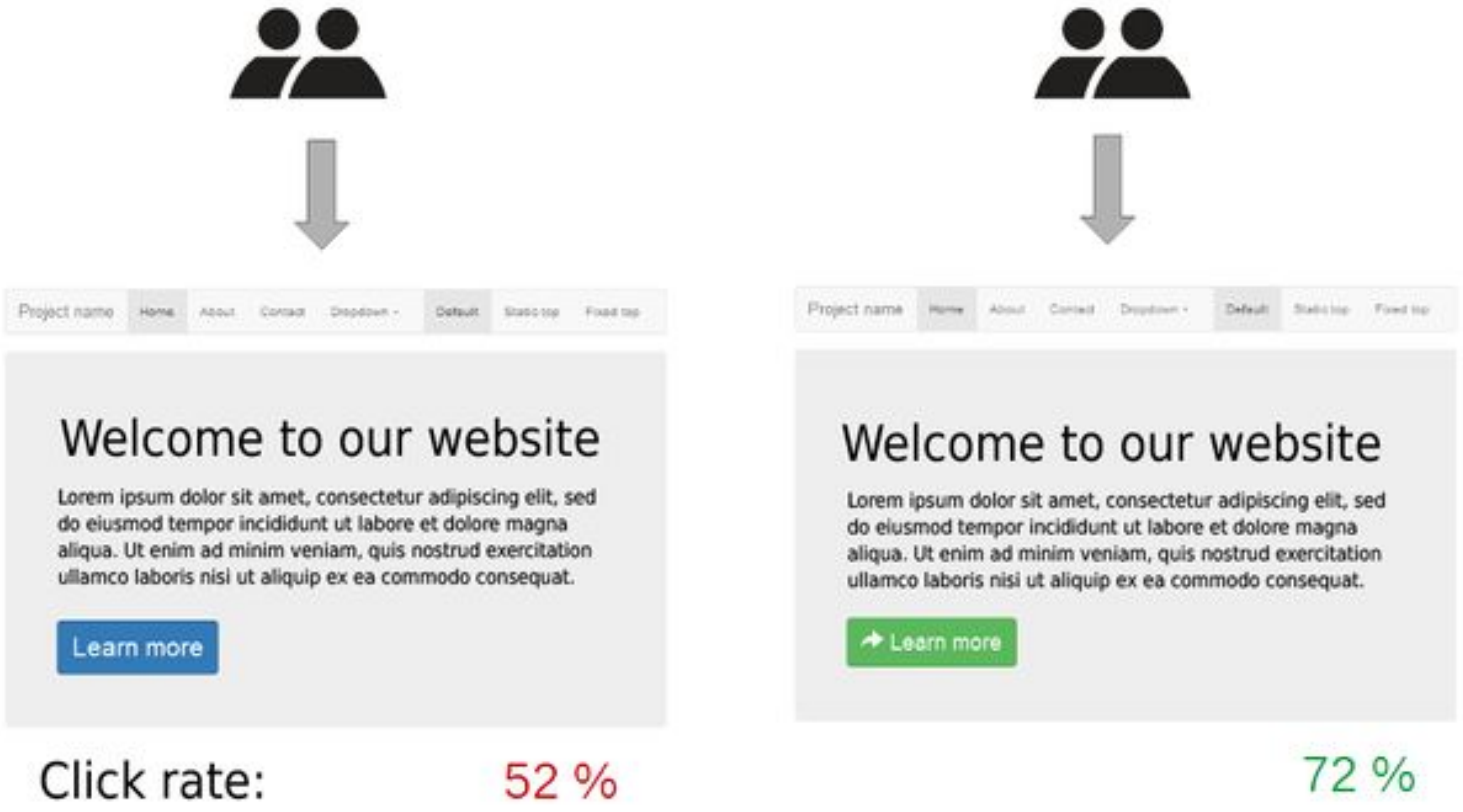
   ex: Recall, Precision, Recall@k, Precision@k, NDCG@k

2. **User Studies:** Present **search interface to a group of users** (say 10-100), often in person or using a system like Amazon Mechanical Turk (can scale to 100s)

3. **Online:** Typically requires **access to a production system** with existing users (challenging for a class project!)

   ex: A/B tests (e.g., to measure click through rate – aka CTR)

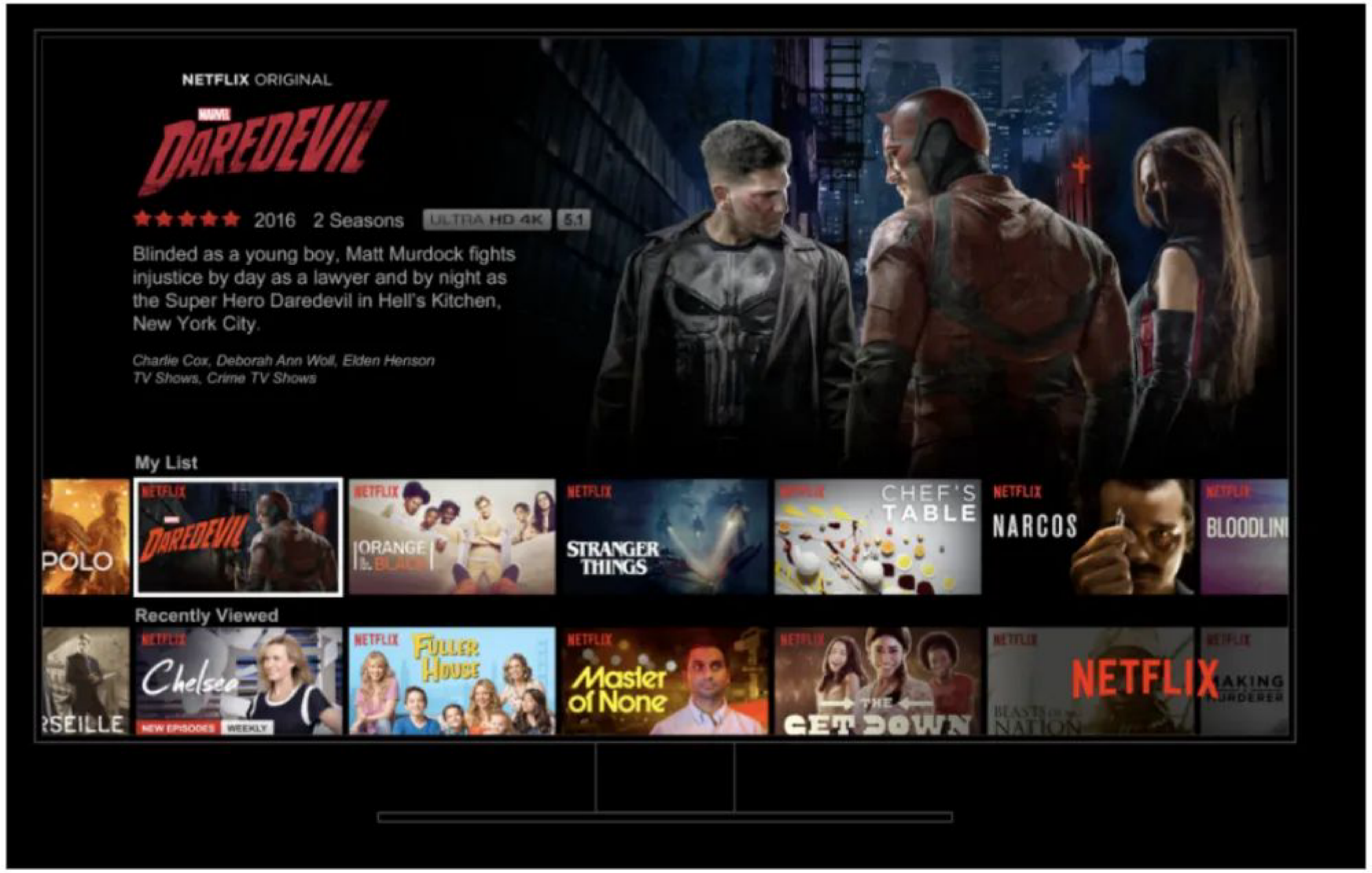
A/B Testing


Project name
Home
About
Contact
Dropdown
Default
Static top
Fixed top
Welcome to our website
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Learn more
Click rate:
52 %
Project name
Home
About
Contact
Dropdown
Default
Static top
Fixed top
Welcome to our website
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Learn more
72 %

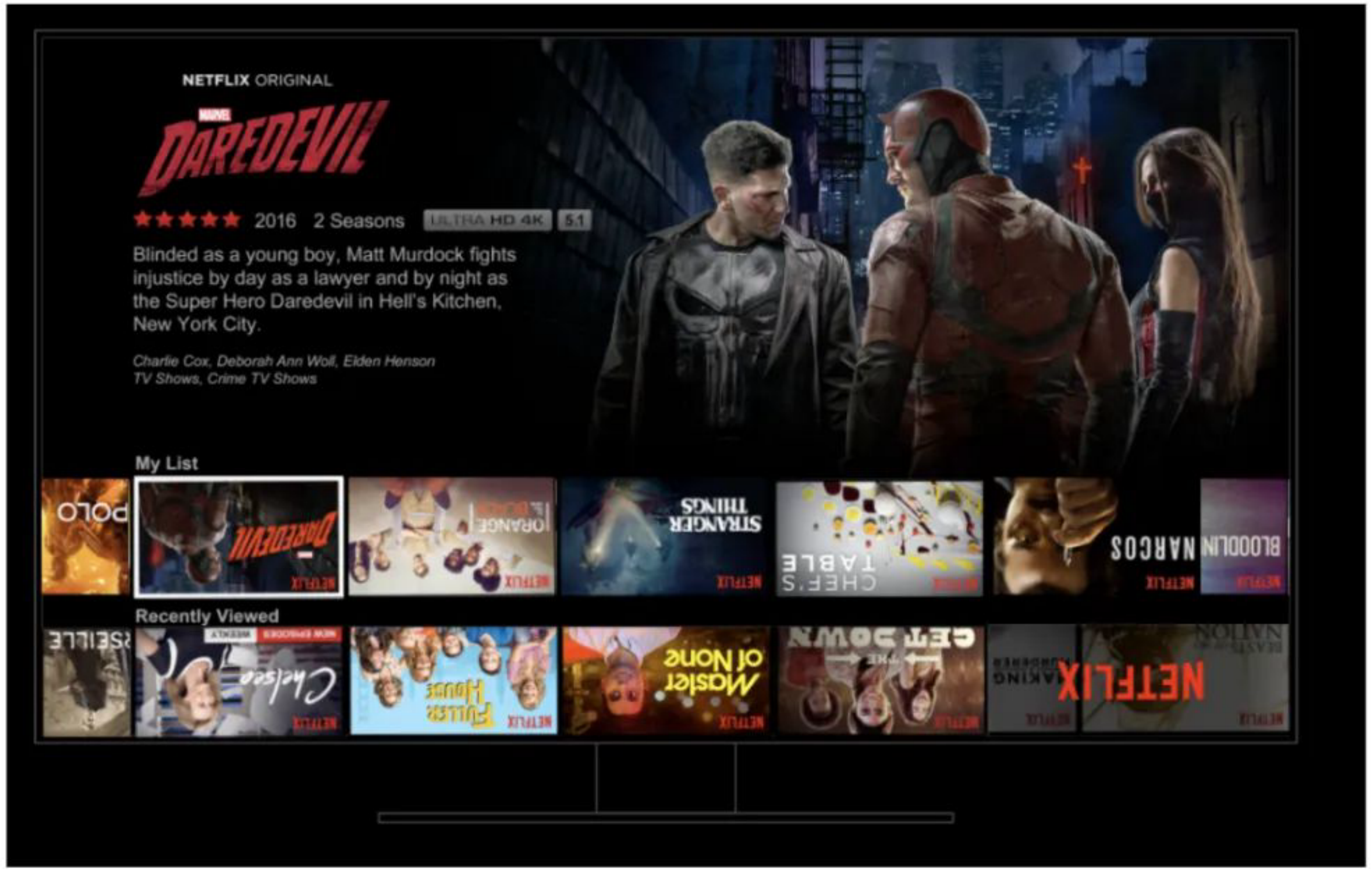
From this blog – it’s awesome go read it:

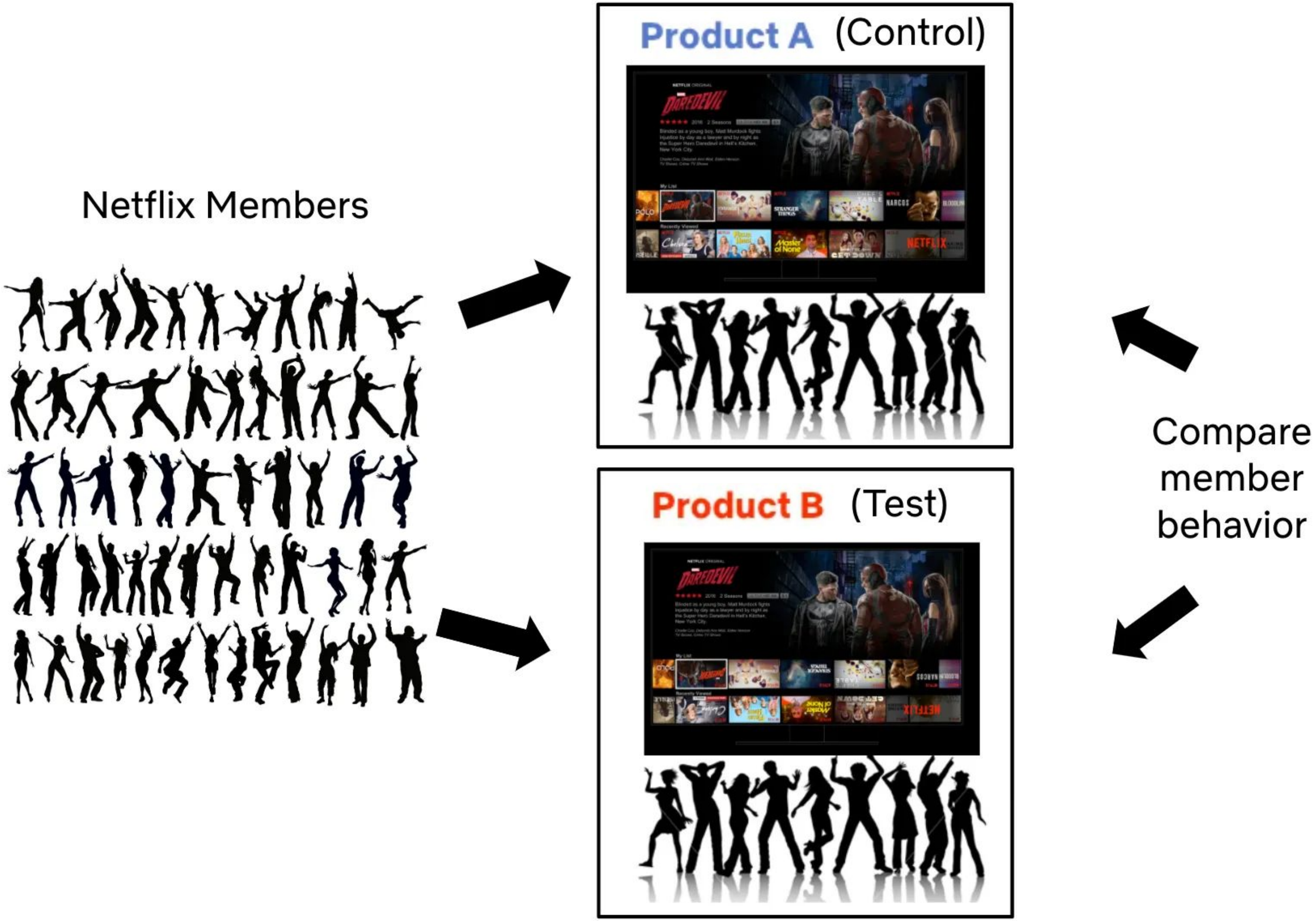
https://netflixtechblog.com/what-is-an-a-b-test-b08cc1b57962

Product A : Standard box art



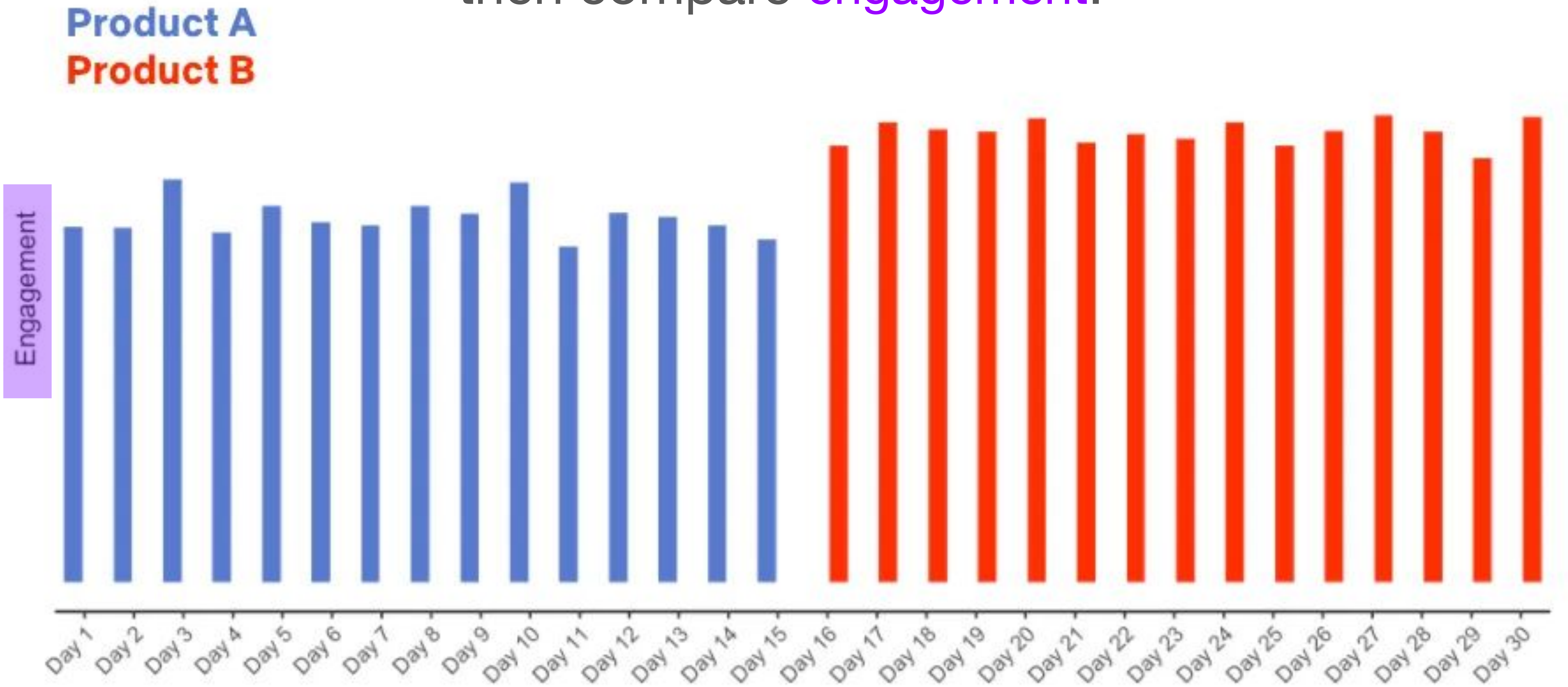
Product B : Upside-down box art




Netflix Members
Product A (Control)
Product B (Test)
Compare
member
behavior

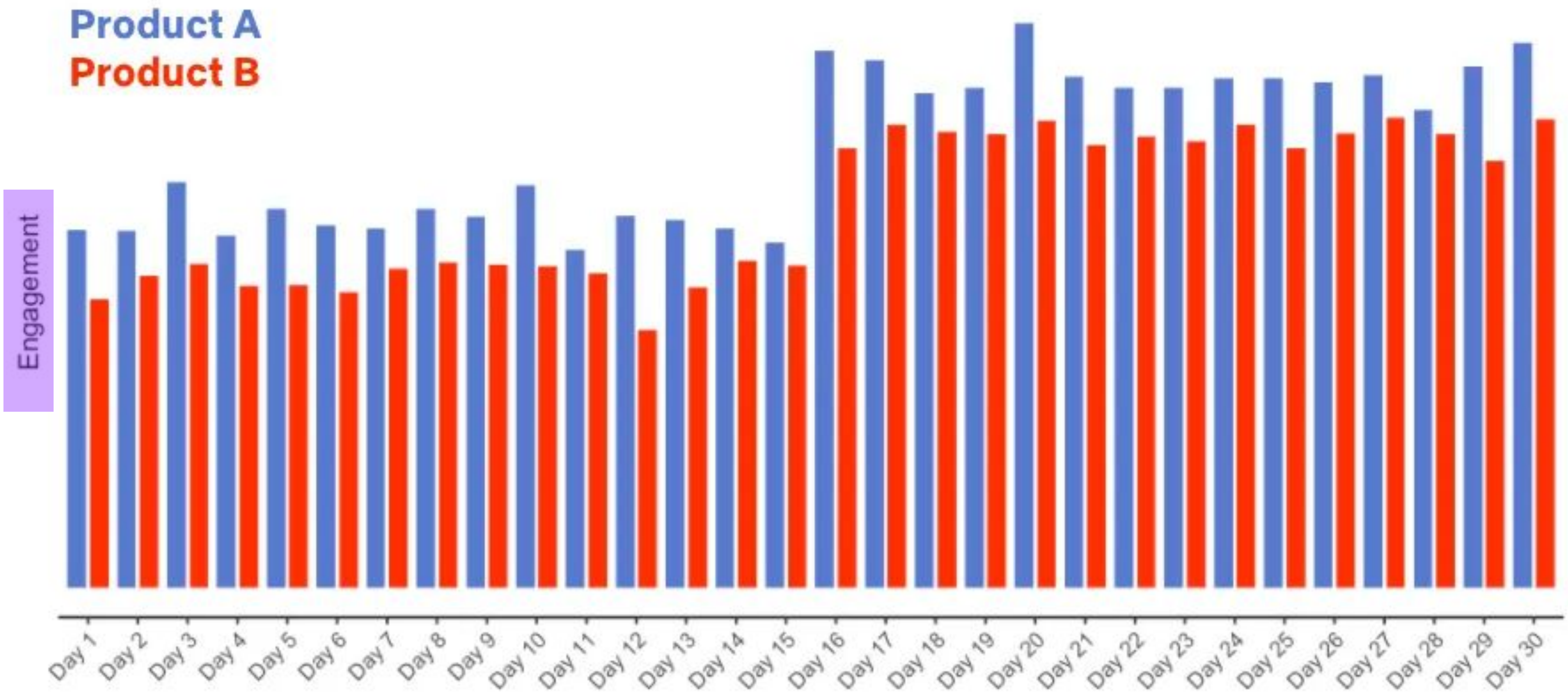
There are different ways to split!

Can run blue algorithm <u>for n days</u>, then red algorithm <u>for n days</u>, then compare engagement.



There are different ways to split!

Can run blue algorithm and red algorithm <u>at the same time</u>, send ½ users to blue and ½ users to red, then compare engagement.



Activity

With your group: In what **situation** would you want to **split your data the 1st way** vs **the 2nd way**?

End of Evaluation

&

Beginning of Learning to Rank!

Activity

With your group, brainstorm some **ranking features** for **Google**!

Let's brainstorm some **ranking features** for **other platforms**!

YouTube: view, subscribers, video length, user profile factors (e.g., age, location), title relevance, video quality, recency, …

LinkedIn: popularity of job posting, # openings, skill match with the user, nearness, recency, salary, …

Spotify: popularity, trustworthiness, location, language, social network, keyword match, …

f(**q**,**d**)

How could we make a ranking function?

STEP 1

f(**q**,**d**) =

a_1 * cosine(**q**,**d**) +

a_2 * BM25(**q**,**d**) +

a_3 *#views in the last day(**d**) +

a_4 *#views in the last week(**d**) +

a_5 * recency(**d**) +

a_6 * PageRank(**d**) +

...

These are the **ranking features!**

STEP 2

If f(**q**,**d**) > *threshold*:

✅ **relevant**

else:

❌ **not relevant**

Instead, let's **learn** a good **ranking function**!

f(q,d)

Very natural idea (especially these days)

But it took a while for ML and IR to be good friends

- Wong, S.K. et al. 1988. Linear structure in information retrieval. *SIGIR.*
- Fuhr, N. 1992. Probabilistic methods in information retrieval. *Computer Journal.*
- Gey, F. C. 1994. Inferring probability of relevance using the method of logistic regression. *SIGIR.*
- Herbrich, R. et al. 2000. Large Margin Rank Boundaries for Ordinal Regression. *Advances in Large Margin Classifiers.*

Brief background: **Learning Tasks**

Different learning tasks are for different types of predictions!

aka outputs

Regression: trying to predict **a real value**

Binary classification: trying to predict **a simple yes/no response (2 classes)**

Multiclass classification: trying to predict **one of a number of classes (n classes)**

Ranking: trying to put a set of objects **in order of relevance (so output a number)**

Text Classification

Given:

- A document space **X**
- A fixed set of classes **C** = $\{c_1, c_2, \ldots\}$
- A training set of labeled documents:

  e.g., $d_1 \rightarrow c_1$, $d_2 \rightarrow c_1$, $d_3 \rightarrow c_2$, …

Learn a **function f** that maps documents to classes **f:** $\mathbf{X} \rightarrow \mathbf{C}$

- e.g., $\mathbf{f(d_1)=c_1}$
- Because **the learning task is classification**, we'll call this function a **classifier**!

Learning f

1 Training Stage 🔥

During the training stage, we know what the the **inputs** and the **outputs** are. So during this stage, we're trying to make **f** really good at mapping **inputs** to **outputs** on $\mathbf{D}_{\mathbf{Train}}$. The **Learning Algorithm** is in charge of this. **So f is changing during this stage.**

2 Testing Stage ❄️

During the testing stage, we pretend we don't know what the **outputs** are for $\mathbf{D}_{\mathbf{Test}}$. Then we plug in different **inputs**, and see if **f** – **So f is frozen (not changing) during this stage.** – gets the outputs right or not (this is **evaluation**)!

D_{Train}

Documents	Classes
d_1	c_1
d_2	c_1
d_3	c_2
…	…

D_{Train}

Learning Algorithm

🔥

$$f: X \rightarrow C$$

$$
\begin{aligned}
f(\mathbf{q},\mathbf{d}) = \; & a_1 * \text{cosine}(\mathbf{q},\mathbf{d}) + \\
& a_2 * \text{BM25}(\mathbf{q},\mathbf{d}) + \\
& a_3 * \text{\#views in the last day}(\mathbf{d}) + \\
& a_4 * \text{\#views in the last week}(\mathbf{d}) + \\
& a_5 * \text{recency}(\mathbf{d}) + \\
& a_6 * \text{PageRank}(\mathbf{d}) + \\
& \ldots
\end{aligned}
$$

What is changing or frozen in f? In this example, all these little coefficients!

❄️ **Test Example** d_{43}

D_{Test}

Documents	Classes
d_{43}	c_1 ✅
…	…

❄️ **Predicted Label** c_1

Evaluation

We can learn f different ways!

Today, we’re going to go over 2 ways:

1. Rocchio
2. kNN

We can learn **f** using

Rocchio

1 **Training Stage** 🔥

Learn **class centroids for each class:** calculate the centroid of all the training examples from each class

2 **Testing Stage** ❄️

Assign **a new example to the class of the nearest class centroid**

D_{Train}

Learning Algorithm

🔥

$f\colon X \rightarrow C$

D_{Train}

Documents	Classes
d_1	c_1
d_2	c_1
d_3	c_2
…	…

❄️ **Test Example**

d_{43}

D_{Test}

Documents	Classes
d_{43}	c_1 ✅
…	…

❄️ **Predicted Label**

c_1

Evaluation

Rocchio Example

There are 3 documents which belong to 2 classes in DTrain. z_i is the formula to calculate the centroid for class i. Use Rocchio with Euclidian distance as the similarity metric to determine which class d_4 belongs to.

$d_1 = [2\ 9\ 3],\ d_1 \in C_1$

$d_2 = [1\ 1\ 1],\ d_2 \in C_2$

$d_3 = [2\ 7\ 0],\ d_3 \in C_2$

$d_4 = [0\ 3\ 1]$

$$z_i = \frac{1}{|C_i|} \sum_{d \in C_i} d$$

Centroid for class 1:

$$z_1 = \frac{1}{1}\big[[2\ 9\ 3]\big] = [2\ 9\ 3]$$

Centroid for class 2:

$$z_2 = \frac{1}{2}\big[[1\ 1\ 1] + [2\ 7\ 0]\big] = \frac{1}{2}\big[[3\ 8\ 1]\big] = \left[\tfrac{3}{2}\ 4\ \tfrac{1}{2}\right]$$

$$\text{dist}(d_4, z_1) = \sqrt{(2-0)^2 + (9-3)^2 + (3-1)^2} = 6.63$$

$$\text{dist}(d_4, z_2) = \sqrt{\left(\tfrac{3}{2}-0\right)^2 + (4-3)^2 + \left(\tfrac{1}{2}-1\right)^2} = 1.87$$

$\text{dist}(d_4, z_1) > \text{dist}(d_4, z_2)$, so d_4 belongs to class 2

We can learn **f** using

kNN

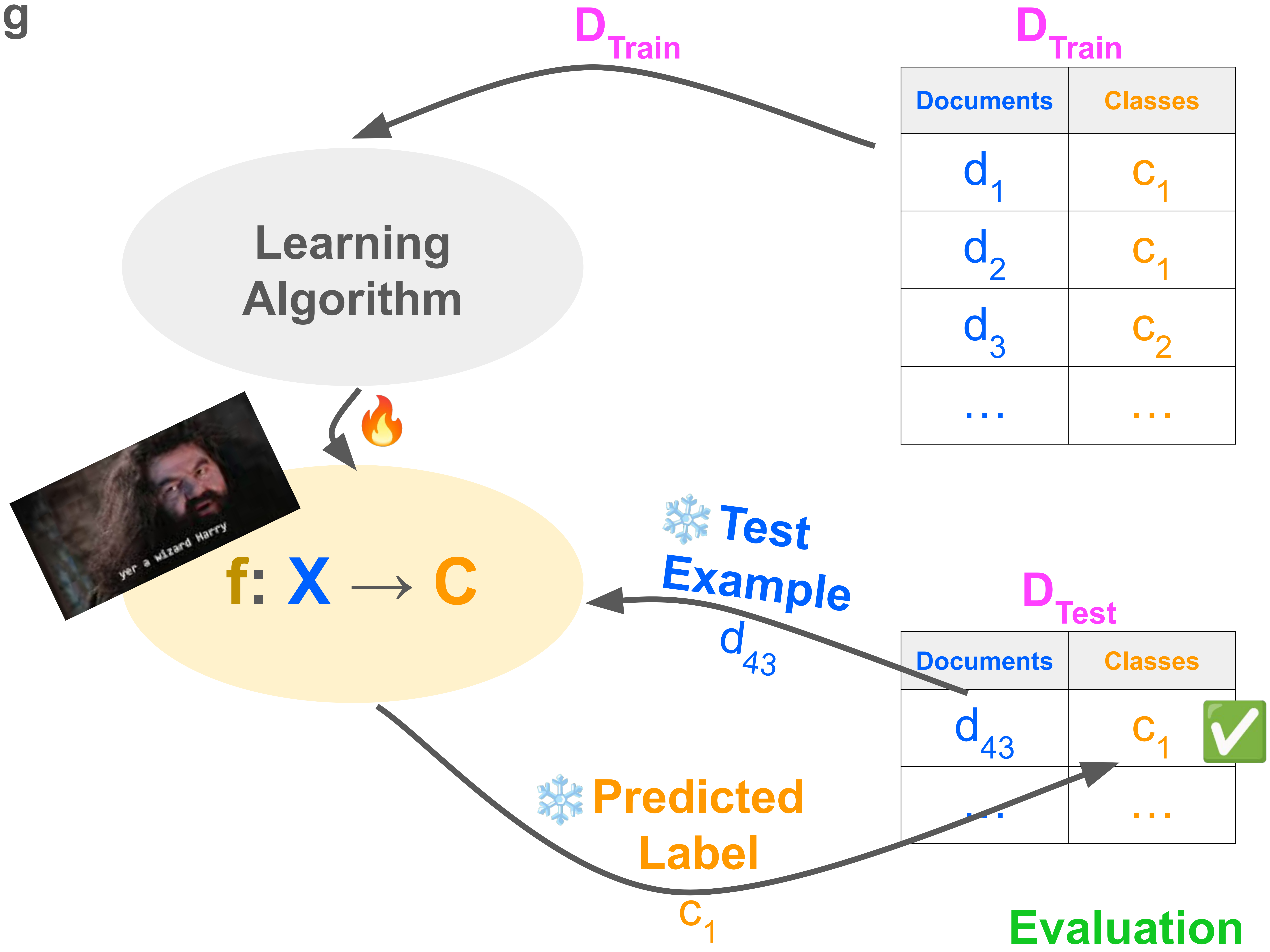
(k Nearest Neighbors)

1 Training Stage 🔥

There's actually no training – **f** doesn't actually learn anything/change at all in this stage! That's ok, it still has a definition, so we can just apply that in the next stage.

2 Testing Stage ❄️

Assign **a new example** to the majority class of the k-nearest training examples.



kNN Example

There are 3 documents which belong to 2 classes in D_{Train}. Use kNN ($K=3$) with Euclidian distance as the similarity metric to determine which class d_4 belongs to.

$d_1 = [2\ 9\ 3]$, $d_1 \in C_1$

$d_2 = [1\ 1\ 1]$, $d_2 \in C_2$

$d_3 = [2\ 7\ 0]$, $d_3 \in C_2$

$d_4 = [0\ 3\ 1]$

K=1

K votes

$\text{dist}(d_4, d_1) = \sqrt{(0-2)^2 + (3-9)^2 + (1-3)^2} = 6.63$

$\text{dist}(d_4, d_2) = \sqrt{(0-1)^2 + (3-1)^2 + (1-1)^2} = 2.24$ $\quad C_2$

$\text{dist}(d_4, d_3) = \sqrt{(0-2)^2 + (3-7)^2 + (1-0)^2} = 4.58$

So we pick the nearest neighbor

C_2 got the most votes

$\therefore d_4$ belongs to C_2

In practice: which **features**?

Very important to select good features to represent our documents

Features we know about:

- TFIDF scores of words (one feature per word)
- Pagerank, Hubs, Authorities
- Popularity, clicks, freshness, …

In practice: which **model (f)**?

Many, many ways to learn a good **classification function** aka **classifier** aka **model** aka **f**:

- Rocchio 😄
- kNN 😄
- Support Vector Machines
- Naive Bayes
- Decision Trees
- Random Forest
- Gradient-Boosted Decision Trees
- Neural Networks
- *… and more! There's, like, a LOT of algorithms for this.*

In practice: which **model (f)**?

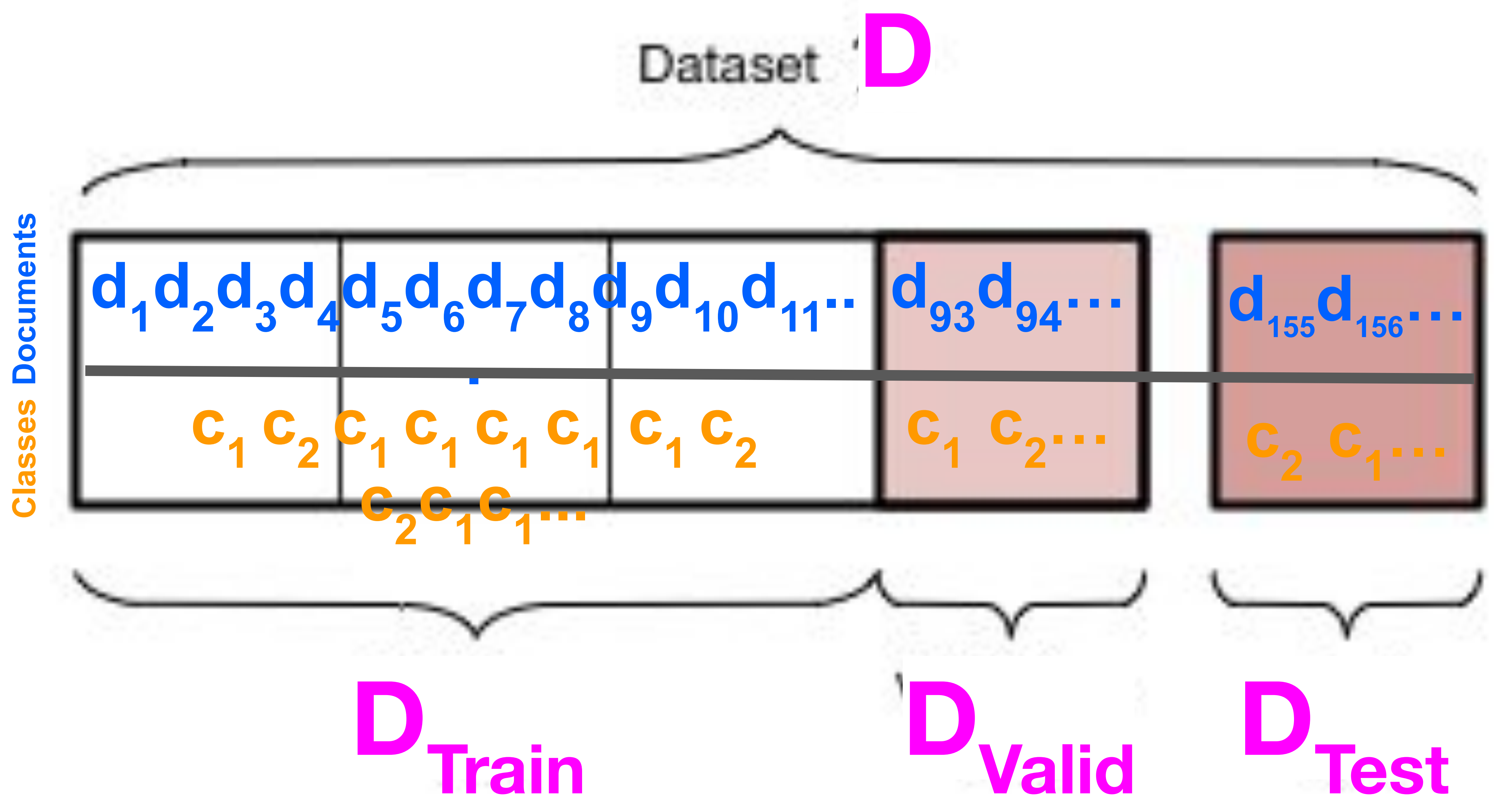
There are a bunch of different models you can choose to use – how do you know which one to use?

Step 1: Keep part of D_{Train} separate as a validation set (D_{Valid})

Step 2: Train **each model (f)** over D_{Train} and "test" over D_{Valid}

Step 3: Choose **the model (f)** that performed the best on D_{Valid} in Step 2

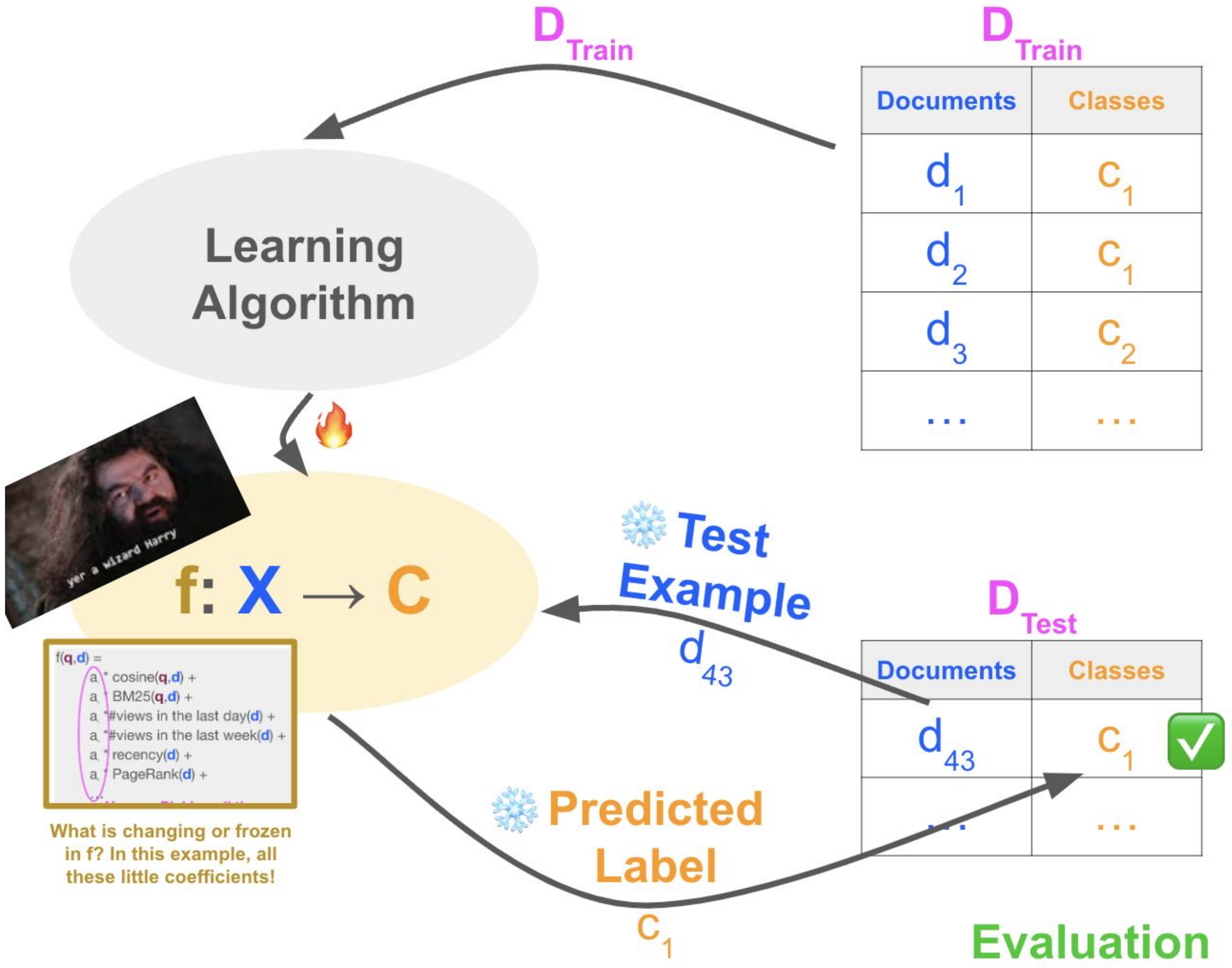
Step 4: Test **that model (f)** on D_{Test} to make sure **it works well** & **didn't overfit**



If we find that our model doesn't work well in the end, we can just start over and make some changes to our process (we can change all these little parts as needed: features, models, model settings/ hyperparameters, evaluation metrics, etc.) to see if that helps.

If the model (f) is overfit on the dataset, that means it won't perform very well on real-world, unseen data! (We're simulating this situation of real-world, unseen data with D_{Test}).

In practice: how to **evaluate**?



We need a way to evaluate how well we do!

*aka how good **f** is*

Accuracy is one way, count up the ✅s and ❌s and report the percent of ✅s!

*Tells us how many our **classifier** guessed correctly!*

There are lots of ways – we may talk about some later 👍

BENCHMARK DATASET:

English	Math	Picture
1) A set of documents 2) A set of queries 3) A binary assessment of either ✅ Relevant or ❌ Non-Relevant for each query and each document	$D = \{(d_i, q_j, r_{ij})\}$ d_i is a vector q_j is a vector $r_{ij} \in \{0,1\}$	D

Documents	Queries	Relevance
d_1	q_1	r_{11}
d_1	q_2	r_{12}
d_1	q_3	r_{13}
…	…	…

Different **learning tasks** are for **different types of predictions!** *aka outputs*

Regression: trying to predict **a real value**

Binary classification: trying to predict **a simple yes/no response (2 classes)**

Multiclass classification: trying to predict **one of a number of classes (n classes)**

Ranking: trying to put a set of objects **in order of relevance (so output a number)**

Activity

With your group, which learning task is the best fit for our benchmark dataset?

Hmm… sounds like a **classification** situation!

1 **Training** 🔥

- Given $\mathbf{D}_{\mathbf{Train}}$ of (**query**, **doc** → **relevance**) triples*
- Learn **f** that outputs ✅ **relevant** or ❌ **non-relevant**

2 **Testing** ❄️

- Given (**query**, **doc**) from $\mathbf{D}_{\mathbf{Test}}$, apply **f(query, doc)**
- Output **relevance**: ✅ **relevant** or ❌ **non-relevant**

** note that our input is not just a doc but both a doc and a query!*

Relevance Classification Example f(query, doc) = relevance

example	docID	query	cosine score	ω	judgment
Φ_1	37	linux operating system	0.032	3	*relevant*
Φ_2	37	penguin logo	0.02	4	*nonrelevant*
Φ_3	238	operating system	0.043	2	*relevant*
Φ_4	238	runtime environment	0.004	2	*nonrelevant*
Φ_5	1741	kernel layer	0.022	3	*relevant*
Φ_6	2094	device driver	0.03	2	*relevant*
Φ_7	3191	device driver	0.027	5	*nonrelevant*