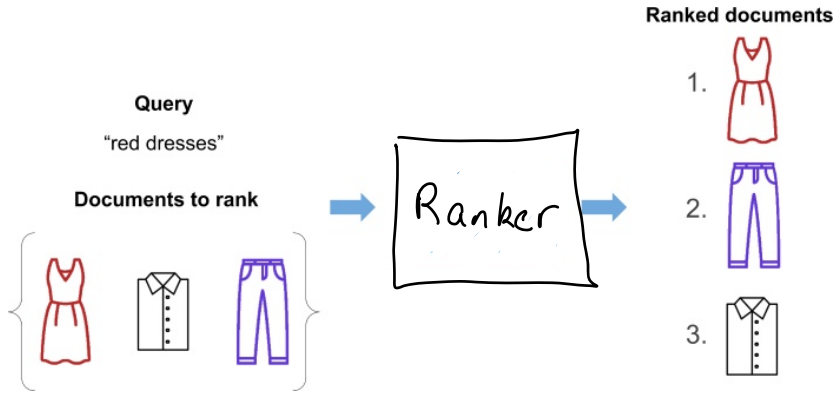
Ranking

Ranking

Task :- Given a "query" + a set of "documents"
produce an ordered list of documents
in decreasing order of relevance



Ranking

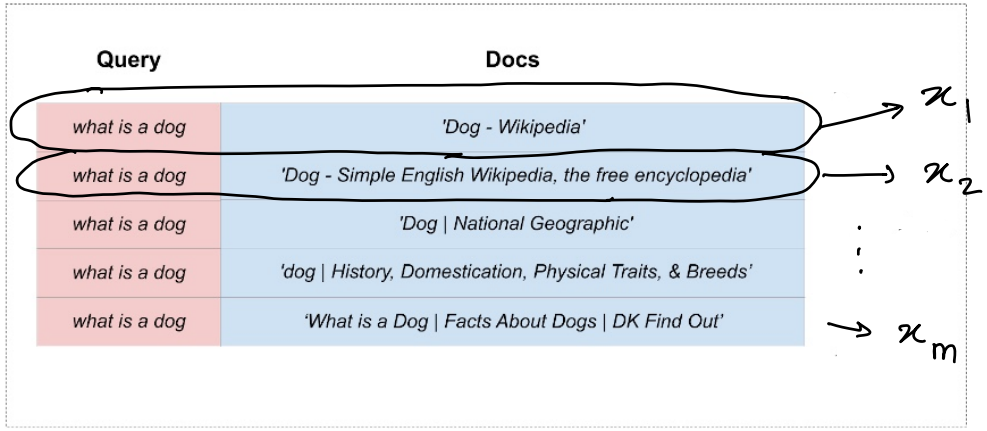
- Arguably the most important ML problem
- Almost everything online is the output of a ranking algorithm
  - Search results (Google, YouTube)
  - Social feeds (Instagram, TikTok ...)
  - Streaming music / video (Netflix, Spotify ...)
  - E-commerce (Amazon, Shein, Walmart ...)
  - News sites / Review sites / ...

A common abstraction

Task :- Given a ~~"query" + a set of "documents"~~ set of feature vectors

produce an ordered list of documents

in decreasing order of relevance



Examples

Use Case	"Query"	"Documents"
Google Search	User-entered text string	Web pages, Maps results, videos, ...
Amazon / Ebay / Walmart	"	Products
Instagram, TikTok	User history, context	Posts from the accounts you follow
YouTube Autoplay, TikTok	"	All video shorts
⋮	⋮	⋮

Relevance Scores

- Need to quantify how relevant any ~~document is for a query~~ feature vector is.

- Typically obtained from how users react to that ~~document for that query~~ feature vector

Relevance Scores - Examples

Use Case	Relevance Score
Web Search	Clicks
Product Search	Purchases, Clicks, Reviews
Streaming Video	Number of minutes/seconds it was watched
Social media content	Likes, reposts, comments ...
Ads	Clicks, Purchases/Sign-ups, ...
⋮	⋮

Setting Up a Ranking Problem

→ Running example of web search

① Identify what is the "query" & "documents"

- "query": only user's text string? or also their history / demographics / location?
- "documents": only web pages? or also videos? maps results? social media results? ...

& how these should be converted to feature vectors

- Number of common words
- deep-learned embeddings
- "web site quality"
- ⋮
- date
- location
- ⋮

(2) Determine a date range for training data & a subsequent date range for validation

all searches Jan 1 - Oct 30, 2023

all searches in Nov 1-20, 2023

Because ranking models are causally deployed in any application

3. Make the samples

- Each sample is a set of pairs of (feature vector, relevance score)

One Sample

Query	Docs	Result when shown
what is a dog	'Dog - Wikipedia'	Clicked
what is a dog	'Dog - Simple English Wikipedia, the free encyclopedia'	Clicked
what is a dog	'Dog | National Geographic'	
what is a dog	'dog | History, Domestication, Physical Traits, & Breeds'	
what is a dog	'What is a Dog | Facts About Dogs | DK Find Out'	Clicked

$$S^{(i)} = \{ (x_1^{(i)}, y_1^{(i)}) \dots (x_m^{(i)}, y_m^{(i)}) \}$$

(Now that we are closer to a stylized ML problem)

4. Choose the ML model(s) to try out

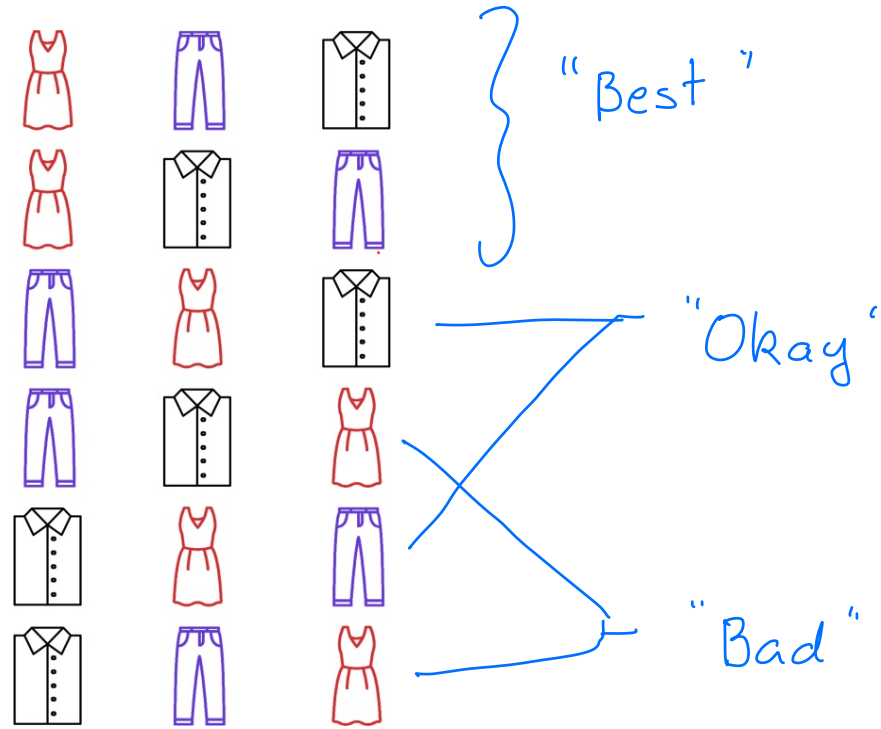
5. Choose the metric to optimize

6. Train the different models / hyperparametrs etc. on the train set & choose the one with the best validation performance.

Ranking Metrics

query:

"red dress"



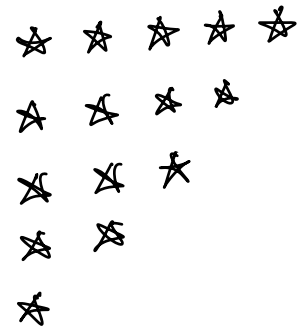
Need to assign a numerical value to any possible permutation of the set of feature vectors

Ranking Metrics

Choice of metric depends on what the relevance scores $\{ y_1^{(i)}, \ldots, y_m^{(i)} \}$ are ...

Eg:- Binary relevance $y_k^{(i)} = 0$ or 1 "Click / NoClick"

One-hot $y_k^{(i)} = 1$ for exactly one $k \in \{1 \ldots m\}$

Multi-level relevance $y_k^{(i)} =$ 

"Ratings / Reviews"

Real-valued $y_k^{(i)} =$ num_mins_played

Ranking Metrics for Binary Relevance

Given query group $q = (x_1, y_1) \dots (x_m, y_n)$

Let π be a permutation of $1 \dots m$. π_j = element in j^{th} position

$$\text{Precision@k}(\pi) = \frac{1}{k} \sum_{j=1}^{k} y_{\pi_j} = \frac{\#\text{ Relevants in top k}}{k}$$

Prec@1 = 1

Prec@2 = $\frac{1}{2}$

Prec@3 = $\frac{1}{3}$

Prec@4 = $\frac{2}{4}$

Prec@5 = $\frac{3}{5}$

$$\text{Average Precision}(\pi) = \frac{\sum_i \left(y_i \times \text{Prec@}\pi_i(\pi) \right)}{\sum_i y_i}$$

eg: $AP = \frac{1}{3}\left(\frac{1}{1} + \frac{2}{4} + \frac{3}{5}\right) = \frac{1}{3}\left(\frac{42}{20}\right) = \frac{7}{10}$

Ranking metrics for One-hot Relevance

Now we only care about position of the relevant item (since there is only 1 such item)

$$RR(\pi) = \frac{1}{(\text{position of first item with } y=1)}$$

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} RR(\pi_q)$$

DCG

commonly accepted assumption: top positions should be given to the highest relevance items

For a ranking π, Discounted Cumulative Gain

$$DCG(\pi) = \frac{y_{\pi_1}}{\log(1+1)} + \frac{y_{\pi_2}}{\log(1+2)} + \frac{y_{\pi_3}}{\log(1+3)} + \cdots + \frac{y_{\pi_n}}{\log(1+n)}$$

$$= \sum_{j=1}^{n} \frac{y_{\pi_j}}{\log(1+j)}$$

OR, alternatively

$$= \sum_{j=1}^{n} \frac{2^{y_{\pi_j}} - 1}{\log(1+j)}$$

NDCG

DCG grows as size n grows, or if the fraction of relevant documents (or, numerical values of the scores) is higher

- Hard to get a consistent numerical barometer of quality from DCG score

$$\text{Normalized DCG}(\pi) = \frac{DCG(\pi)}{DCG(\pi^*)}$$

π^*: ideal permutation

NDCG is now the standard ranking metric

ML models for Ranking

Most common approach: score functions

(a) Find a "scoring" function $\phi(x)$

(b) Evaluate $\phi(x_1) .. \phi(x_n)$

(c) Sort in decreasing order of $\phi(x_i)$

To find $\phi(\cdot)$ we need to choose

(a) MODEL ARCHITECTURE :- functional form of $\phi(\cdot)$

- e.g. GBDT, neural network etc.

(b) LOSS FUNCTION that connects (sorted) $\{\phi(x_i)\}$ to the given relevances $\{y_i\}$

Loss Functions

Pointwise
- Treat ranking like any regular ML problem where y_i's need to be predicted / classified
- Generally seen to not perform well

Pairwise
- Make a loss based on pairs i_1 & i_2
- Usually significantly better than pointwise

Listwise
- Make a loss based on whole set
- Performs better than pairwise but can be much slower