IR0

Zipf's law**:** The probability of occurrence of words or other items starts high and tapers off. Thus, a few occur very often while many others occur rarely. ***Heap's law****:* Number of unique words is sublinear.

Text analysis pipeline: Tokenize, lower case, remove stop words, stemming, deal with phrases.

1. **Inverted Indexing:** word – documents (inverted lists can store the doc. Id's, term freq., position, etc.)
2. **Constructing index: 1.** In-memory: **Two-pass-index**: reserve memory space and fill, **One-pass index**: Look at number (4?) of docs, create index, pass to disc, start again, finally merge all indexes**. 2.** Single-threaded: **Distributed Indexing:** MapReduce, processing is done in parallel.
3. **Updating index: 1. No merge:** create new index for every new word, never merge, search in both indexes with query. (Low index maintenance cost, high query processing cost) **2. Incremental update:** Allow buffer in original index, add new words. (Can run out of space) **3. Immediate merge:** build extra index, merge at some point. (Merging is time consuming) **4. Lazy merge: (**Often used**),** Delta index is merged into bigger delta indexes.

Evaluation (offline)

Test collection: Test documents, test queries, and ground truth.

Depth K pooling: Consider multiple search systems, consider top K results, remove duplicates, present documents to judges in a random order. (Produces a large number of judgements but is still incomplete).

Multiple-assessors: K =(P(A) – P(E))/ (1 – P(E)), where P(A) = prob. of an agreement (two assessors agree), P(E) = expected chance of agreement (accidental agreement).

Precision = #relevant items retrieved/#retreived items; **Recall** = #relevant items retrieved/#relevant items. If you retrieve all documents, recall will be 1 but precision will be very low (and the other way around). **F1 = (**2PR)/(P+R). Problem with these metrics is that ranking is not taken into account.

Ranked evaluation: Precision@K = #relevant items at K/K, **Recall@K**=#relevant items at K/#relevant items, **Reciprocal Rank (RR)** = 1/rank of first relevant item, **Average precision (AP)** = sum(P@K) (for every doc that is relevant)/#relevant items. **Average over multiple queries:** mean at P@K, mean at R@K, MRR, MAP. *(Problems: User search behaviour is not taken into account)*

Discounted Cumulative Gain (DCG) =

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

Normalized DCG (NDCG) =

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCG_p},$$

Average exam (average number of examined items) =

$$\begin{aligned}\text{Avg. exam} &= \sum_{k=1}^{\infty} k \cdot P(\text{look at } k) \cdot P(\text{stop at } k) \\ &= \sum_{k=1}^{\infty} k \cdot \theta^{k-1} \cdot (1-\theta) \\ &= \frac{1}{1-\theta}\end{aligned}$$

- View next item with probability θ
- Stop with probability $1 - \theta$
- Probability of looking at rank k

Utility@K =

$$U@k = P(\text{look at } k) \cdot R_k = \theta^{k-1} \cdot R_k$$

Expected reciprocal rank (ERR) =

$$\begin{aligned}ERR &= \sum_{k=1}^{N} \frac{1}{k} \cdot P(RR = \frac{1}{k}) \\ &= \sum_{k=1}^{N} \frac{1}{k} \cdot \theta^{k-1} \cdot R_k \cdot \prod_{i=1}^{k-1}(1 - R_i)\end{aligned}$$

Rank-biased precision (RBP) =

$$RBP = \frac{\sum_{k=1}^{N} U@k}{\text{Avg. exam}} = (1-\theta) \cdot \sum_{k=1}^{N} \theta^{k-1} \cdot R_k$$

Document representation and matching

Query processing: First phase ranking = Matching, simple ranking, heap, **second phase ranking** = complex heap, heap, **Inverted list:** per word check if it is in the document, **Document at the time:** Check for all query words if it is in a document, and do this for all documents.

Term based retrieval:

1. **Vector Space Model (VSM):** documents and queries as vectors, match using cosine similarity, weights can be: 1, binary, 2. Term frequency, 3. TF-IDF.

$$idf(t) = \log \frac{N}{df(t)}$$

- $df(t)$ – document frequency of term t
- N – total number of documents in a collection

$$\text{TF-IDF}(t,d) = tf(t,d) \cdot idf(t)$$

2. **Language modelling in IR:** statistical language model is a probability distribution over sequences of words.

If a term of the query does not appear in the document **ULM** or **QLM** will be 0, this is addressed by smoothing.

Unigram language model (bag of words (right)):

$$P(t \mid M_d) = \frac{tf(t,d)}{dl(d)} \qquad P(q \mid M_d) = \prod_{t \in q} P(t \mid M_d) = \prod_{t \in q} \frac{tf(t,d)}{dl(d)}$$

Query likelihood model:

$$P(d \mid q) = \frac{P(q \mid d) P(d)}{P(q)}$$

KL-divergence:

$$KL(M_d \| M_q) = \sum_{t \in V} P(t \mid M_q) \log \frac{P(t \mid M_q)}{P(t \mid M_d)}$$

Jelineck-mercer smoothing:

$$\begin{aligned}P_s(t \mid M_d) &= \lambda P(t \mid M_d) + (1-\lambda) P(t \mid M_c) \\ &= \lambda \frac{tf(t,d)}{dl(d)} + (1-\lambda) \frac{cf(t)}{cl}\end{aligned}$$

Dirichlet smoothing:

$$P_s(t \mid M_d) = \frac{tf(t_i,d) + \mu P(t_i \mid M_c)}{dl(d) + \mu}$$

3. BM25:

$$BM25 = \sum_{t \in q} \log \left[\frac{N}{df(t)}\right] \cdot \frac{(k_1 + 1) \cdot tf(t,d)}{k_1 \cdot \left[(1-b) + b \cdot \frac{dl(d)}{dl_{avg}}\right] + tf(t,d)}$$

- k_1, b – parameters
- $dl(d)$ – length of document d
- dl_{avg} – average document length

Semantic based retrieval: Even if a query term does not appear in the document but it is in general about the query, we still want the document to be ranked high.

1. **Topic modelling:**

Probabilistic Latent Semantic Analysis (pLSA):

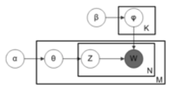
$$P(w \mid d) = \sum P(w \mid \phi_z) P(z \mid \theta_d)$$

Latent Dirichlet Allocation (LDA): is a model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's presence is attributable to one of the document's topics.

2. **Latent Semantic Indexing/analysis:**

Single Value Decomposition (SVD): Given a document and a query, represent them as a vector in the obtained "semantic" vector space, then match the obtained "semantic" vector representations d(vec) and q(vec) Using cosine similarity.

$$\begin{aligned}C = U \Sigma V^T &= \sum_{i=1}^{\min(m,n)} \sigma_i \vec{u}_i \vec{v}_i^T \\ &\approx \sum_{i=1}^{k} \sigma_i \vec{u}_i \vec{v}_i^T = U_k \Sigma_k V_k^T\end{aligned}$$



W	Words in documents (observed)
Z	Topics for words (not observed)
$\boldsymbol{\theta}$	Distributions of topics in documents (parameters)
ϕ	Distributions of words in topics (parameters)
$L(\boldsymbol{\theta}, \phi; W, Z) =$	Log-likelihood of observed data W and unobserved
$p(X, Z \mid \boldsymbol{\theta}, \phi)$	random variables Z, given parameters $\boldsymbol{\theta}, \phi$

3. Neural models:

Word embeddings: models used to create semantic representations of words (e.g. Word2vec, skipgram, CBOW). Given a document and a query, compute their vector representations as average word embeddings (AwEs). **Document embeddings:** build documents vector representation directly – given a query, fix word matrix, add column to doc. matrix that represents the query, update doc matrix using gradient descent, get the vector representation of the query from the updated matrix, and finally match using cosine similarity.

Neural models for ranking

Representation-based: Take a single representation of the query and of the document (los van elkaar) and compare these. **Interaction-based:** Looking at the interactions between individual terms in the query and the document.
Interaction-based (FFN/RNN/CNN): Many interaction-based approaches consume a similarity matrix to predict a documents relevance.
Approach KNRM: Generate similarity matrix, apply gaussian kernels to each row, sum kernel scores across query terms (with the use of the similarity score), compute document score. Note: embeddings are trained along with the model.
Interaction-based transformer (cross-encoder): BERT: produces contextualized embeddings of query and document (text). Limitations of BERT: computationally expensive, another limitation is that we feed both query and document in the model at once so we can't pre-compute anything. Another limitation: restricted to an index of 0-511, this can be a problem with a full document.
Representation-based (bi-encoder): solution to the previous problem because the document and query scores can be computed offline separately, and then use nearest neighbour search. Challenges: choosing negative training examples, formatting multiple representations for long documents, and "zero shot" setting.

```
score = BERT(query) · BERT(document)
```

Learning to rank

The task to automatically construct a ranking model using training data, such that the model can sort new objectives according to their degrees of relevance, preference or importance.
LTR: preliminaries & goal: Represent the document and query in a format that a ML model can use, ranking model optimized to score each document – query combination so that relevant documents are scored higher.
Features: Document only (e.g. doc length), document – query combination (e.g. BM25), query only (e.g. query length).
Offline or supervised LTR: learn from annotated data, expensive and time consuming, provides ground truth. **Online/counterfactual LTR:** Learn from user interactions, virtually free and easy to obtain, hard to interpret.
Pointwise approach: regression or classification-based approach, treat the labels as classes, Main issues: class imbalance (many irrelevant and few relevant documents), distribution of features differs greatly per query (normalization is needed), but the fundamental problem is that ranking is not a regression or classification problem! A document loss does not work for ranking because documents scores should not be considered independently. **Regression loss:**

$$\mathcal{L}_{Squared} = \sum_{q,d} \|y_{q,d} - f(\vec{x}_{q,d})\|^2$$

Classification loss:

$$\mathcal{L}_{\text{CE}} = \sum_{q,d} -log\Big(p(y_{q,d}|q,d)\Big) = \sum_{q,d} -log\Big(\frac{e^{\gamma\cdot s_{y_{q,d}}}}{\sum_{y\in Y} e^{\gamma\cdot s_y}}\Big)$$

Pairwise approach: instead of looking at document level, consider pairs of documents, do not change the model to take document pairs as input, time complexity = O(N^2), score documents and order according to scores, minimizes the average number of interventions in ranking.

$$\mathcal{L}_{pairwise} = \sum_{d_i \succ d_j} \phi(s_i - s_j)$$

RankNet: cross entropy loss:

$$\mathcal{L}_{ij} = \frac{1}{2}(1 - S_{ij})\gamma(s_i - s_j) + \log(1 + e^{-\gamma(s_i - s_j)}).$$

The factorized cross entropy loss:

$$\frac{\delta \mathcal{L}_{ij}}{\delta w} = \gamma\left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{\gamma(s_i - s_j)}}\right)\left(\frac{\delta s_i}{\delta w} - \frac{\delta s_j}{\delta w}\right).$$

$$\lambda_{ij} = \gamma\left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{\gamma(s_i - s_j)}}\right)$$

These lambdas act like forces pushing pairs of documents apart or together.
Problem with Pairwise approach: not every document pair is equally important
It is much more important to get the correct ordering at the top documents.
Listwise approach: get the difference in score of ranking methods together with the pairwise scores. Normally DCG and precision are non-continuous and non-differentiable, but the difference is not. **LambdaRank:** To train a model we do not need the costs themselves, only the gradients (of the costs w.r.t. model scores), the gradient should be bigger for pairs of documents that produces a bigger impact in NDCG by swapping positions.

$$\lambda_{LambdaRank} = \lambda_{RankNet} \cdot |\Delta \text{NDCG}|$$

ListNet and ListMLE: create probabilistic model for ranking, which is differentiable. **ListNet:** Compute the probability distribution over all possible permutations based on model score and ground-truth labels. The loss is then given by the KL-divergence between these two distributions. This is computationally very costly, computing permutations of only the top-K items makes it slightly less prohibitive. **ListMLE**: Compute the probability of the ideal permutation based on the ground truth. However, with categorical labels more than one permutation is possible which makes this difficult.

User interactions

Position bias: click faster on high ranked documents. **Attention bias:** do you click faster on visually attractive things. **Source bias:** click on wiki instead of something else because you know the sourcs. **Trust bias:** Trust google to give you the best result.
Click models: **Position-based model:** examination depends on rank, a user wants to click on a document after examining its snippet (if it is attractive). P(E = 1) = probability of examination P(A = 1) = probability of attractiveness.
Problem: prob of examining snippets in different positions only depends on the position, while in the real world, this is not the case. (examination does not depend on examination and clicks above), Pro: examination and attractiveness.

$$P(E_{r_d} = 1) = \gamma_{r_d}$$
$$P(A_{qd} = 1) = \alpha_{qd}$$
$$P(C_d = 1) = P(E_{r_d} = 1) \cdot P(A_{qd} = 1)$$

Cascade model: start from first document, examine one by one, if we click, we stop, otherwise continue. Pro: depends on the examination at r and on examinations of the docs above. Con: only one click is allowed.
Estimation: **Expectation maximization:** 1, set parameters to some initial values, 2. Repeat until convergence, - E-step: derive the expectation of the likelihood function, - m-step: maximize this expectation. This can be used for PBM for example.
Applications: **click probabilities:** Full click probability: prob that a user clicks on a doc at rank r, P(Cr = 1), Conditional probability: prob that a user clicks on a documents at rank r given previous clicks. **Dynamic Bayesian network model (DBM):** type of cascade model but it allows multiple clicks.

$$= R_{qd_r} \cdot \prod_{i=1}^{r-1} (\gamma \cdot (1 - R_{qd_i}))$$

Counterfactual LTR

Counterfactual evaluation: evaluating unbiasedly from historical interactions. **Propensity-weighted LTR:** learning unbiasedly from historical interactions. **Counterfactual evaluation:** evaluate a new ranking function for using historical interaction data (e.g. clicks) collected from previously deployed ranking function. **Reason for no click:** Noise: averaging over many clicks will remove their effect. **Bias:** averaging will not remove their effect. Core problem with the naïve estimator is the position bias, in the formula, this is the examination probability P(oi = 1 | R,D).

$$\mathbb{E}_o[\Delta_{NAIVE}(f_\theta, D, c)] = \mathbb{E}_o\left[\sum_{d_i : o_i = 1 \wedge y(d_i)=1} \lambda(rank(d_i \mid f_\theta, D))\right] = \sum_{d_i : y(d_i)=1} P(o_i = 1 \mid R, d_i) \cdot \lambda(rank(d_i \mid f_\theta, D)).$$

Inverse Propensity Scoring: counterfactual evaluation accounts for bias using IPS:

$$\Delta_{IPS}(f_\theta, D, c) = \sum_{d_i \in D} \frac{\lambda(rank(d_i \mid f_\theta, D))}{P(o_i = 1 \mid R, d_i)} \cdot c_i,$$

- $\lambda(rank(d_i \mid f_\theta, D))$: (weighted) rank of document d_i by ranker f_θ,
- c_i: observed click on the document in the log,
- $P(o_i = 1 \mid R, d_i)$: examination probability of d_i in ranking R displayed during logging.

This is an unbiased estimate of any additive linearly decomposable IR metric. Hidden assumption is that the prob of observing a doc is bigger than 0.
Another assumption is no noise, however, It always works as long as the chance of observing one document is bigger than the other (relative difference). Since we can prove relative differences are unbiased, even with noise you won't get the exact score but you will know which documents is better. will know which documents is better.
You want to know how well a ranker works without using annotated data, but by using Historical clicks. **Propensity-weighted LTR:** the IPS estimator can unpriestly estimate performance (rank functions are non-differentiable because they are not soft, which means you are ether at rank 1 or 2, but never in between) **Rank-SVM:** optimizes the differentiable upper bound:

$$rank(d \mid f_\theta, D) = \sum_{d' \in R} \mathbb{1}[f_\theta(d) \le f_\theta(d')] \le \sum_{d' \in R} \max(1 - (f_\theta(d) - f_\theta(d')), 0) = \overline{rank}(d \mid f_\theta, D).$$

This can be applied to various pairwise learning approaches:

$$\Delta_{DCG\text{-}IPS}(f_\theta, D, c) = \sum_{d_i \in D} \frac{\log_2(1 + rank(d_i \mid f_\theta, D)^{-1}}{P(o_i = 1 \mid R, d_i)} \cdot c_i \ge \sum_{d_i \in D} \frac{\log_2(1 + \overline{rank}(d_i \mid f_\theta, D)^{-1}}{P(o_i = 1 \mid R, d_i)} \cdot c_i.$$

Walkthrough: obtain a model of position bias, acquire a large click log, then for every click In the log: - compute the propensity of the click, - calculate the gradient of the bound (upper Or sigmoid)of the Unbiased estimator, update the model by adding/subtracting the gradient.
Estimating position bias: assumption: the observation probability only depends on the Rank of the document. **Rand-top-n:** randomly present top n documents to user multiple Times and check the relative differences in clicks. Drawback: bad user experience.
RandPair: same as Rand-top-n but with pairs, better user experience. **Interventional sets/ harvesting:** Inhered "randomness" in data coming from multiple rankers (A/B testing). **Jointly learning and estimating:** learn an optimal ranker given a correct propensity model, learn an optimal propensity model given a correct ranker. **Address trust bias:** the trust bias explanation for fewer clicks on low ranked documents is because of the fewer wrongly clicked times. While position bias only says it is much less likely that the documents are observed. Trust models are overall better.

$$\Delta_{Bayes\text{-}IPS}(f_\theta, D, c) = \sum_{d_i \in D} P(y(d_i) = 1 \mid c_i = 1, k) \cdot \frac{\lambda(rank(d_i \mid f_\theta, D))}{P(o_i = 1 \mid R, d_i)} \cdot c_i = \sum_{d_i \in D} \frac{\epsilon_k^+}{\epsilon_k^+ + \epsilon_k^-} \cdot \frac{\lambda(rank(d_i \mid f_\theta, D))}{P(o_i = 1 \mid R, d_i)} \cdot c_i.$$

High variance: counterfactual LTR systems will run into the problem of high variance. This can be due to the factors: not enough training data, extreme position bias and very small propensity, large. Amounts of noisy clicks on documents with small propensity. If the propensity score is extremely small and the document gets more than expected clicks, it will blow up the score by dividing with a very small number (see formula). **Propensity clipping:** bound the propensity to prevent any single sample from overpowering the rest of the data. This solution is a trade of between bias and variance: it will introduce some amount of bias but can substantially reduce variance.
Supervised LTR: user manually annotated labels: - expensive to create – impossible to many settings, - often misaligned with actual user preferences, optimization is widely studied and very effective. **Counterfactual LTR:** user clicks logs: - available in abundant quantities, - effectively no cost, - contains noise and biases, Noise amortized over large numbers of clicks (average it out), Biases: position bias mitigated with inverse propensity score, other biases are an active area of research.

Online LTR

Bandits for ranking: find optimal ranking for a single query, upper confidence bounds on relevance per document, divide and conquer: split documents in groups so that there are high confidence relevance differences between groups, click through rate estimator. Goal: find the optimal ranking for a single query, results do not generalize to other queries. Advantages: not limited by features, disadvantage: learning from scratch for every new query. **Difference with LTR** which tries to find a model that generalizes to every query. **Online evaluation: A/B testing:** randomizes system exposure to users to measure differences. **Interleaving:** take rating for two rankers, create interleaved ranking based on both methods, clicks show preference of user for ranker. Idea: deal with position bias by randomize display positions of documents, limit randomization to max user experience. It requires fewer interactions for a reliable preference than A/B testing., it is not effective on historical data. Efficiency comes form: display most important documents, looking for relative differences. **Duelling Bandit Gradient Descent:** intuition: if online evaluation can tell us if a ranker is better than another, then we can use it to find an improvement of our system. By sampling model variants and comparing them with interleaving, the gradient of a model w.r.t. user satisfaction can be estimated. DBGD: wait for user query, sample random direction from unit sphere, compute candidate ranking model, get ratings of both rankers, compare using interleaving, if the new one wins, update the model slightly in that direction. Assumptions: - there is a single optimal set of parameters, - the utility space is smooth, (small changes lead to small changes in the user experience). Proven to have **sublinear regret:** difference in regret must become smaller for every step. **Candidate pre-selection:** sample large number of rankers to create candidate set and compare rankers one by one until you have one left. Works well for small dataset but because it is online learning, you can't stop. **Multi leave gradient Descent:** multiple rankers being compared at the same time, move to the average of the best couple of rankers. Speeds up the learning rate of DBGD, much better user experience, the exploration is done more efficiently, huge computational costs. **Problems with DBGD**: performance is much worse than offline approaches, assumptions don't hold. There is always the same model multiplied by a constant (no single optimal model). **Pairwise Differentiable Gradient Descent (PDGD):** intuition: a pairwise approach can be made unbiased, while being differentiable, without relying on online evaluation method or the sampling of models. Start with initial model, the indefinitely: wait for a user query, sample a ranking from the document distribution, display ranking to the user, infer document preferences from user clicks, simulate reverse pair ranking, update model according to the estimated (unbased) gradient. Distribution:

$$P(d \mid D, \theta) = \frac{\exp^{f_\theta(\mathbf{d})}}{\sum_{d' \in D} \exp^{f_\theta(\mathbf{d'})}}.$$

Update according to estimated (unbiased) gradient:

$$\nabla f_{\theta_t}(\cdot) \approx \sum_{d_i >_{\mathbf{c}} d_j} \rho(d_i, d_j, R) \nabla P(d_i \succ d_j \mid D, \theta_t).$$

Empirical comparison: DBGD: unable to reach optimal performance in ideal settings, strongly affected by noise and position bias. PDGD: capable of reaching optimal performance in ideal settings, robust to noise and position bias, considerably outperforms DBGD in all tested experimental settings.
Comparison: supervised LTR: - Uses manually annotated labels, - optimalization is widely studied and very effective, often unavailable for practitioners, **Online LTR:** - learns from directed interactions, - debiases by randomization, ineffective when applied to historical data, - unbiased w.r.t pairwise preferences, - not guaranteed to be unbiased w.r.t metrics.

Conversational IR

Search: laptop, smartphone, voice/conversational search. **User Interactions: single initiative:** user or system has control and asks question, the other only responds to request. **Mixed initiative:** User and system have control, both are active, responds to once request, feedback. **Possibilities: Feedback first (FF):** ask for feedback before showing result, **Feedback after (FA):** ask for feedback after showing results. **Mixed initiative approach: Query suggestion (QS):** suggest what the query should have been, **Query clarification:** ask clarifying questions to get a better result. **Goal based dialog agents:** Siri. Cost of actions can be measured as time.

Evaluation

Automatic evaluation: single initiative, for goal-oriented dialog system: is the goal achieved, how many turns did it take? **Human evaluation:** mixed initiative, **Limited bandwidth:** smaller screens, top of the list is even more important. **Metrics:** relevance of the retrieved documents to the overall topic, relevance of the retrieved documents to the current question. Top of the metric list: **Mean reciprocal rank (MRR):** what is the ranking of the first relevant item, **Mean average precision (MAP):** what precision do we get for different value of recall? **Normalized Discounted cumulative gain (NDCG@3) at top 3:** what is our cumulative gain at the top 3 results? **For question retrieval:** recall is the most important metric because you want to retrieve as many possible relevant questions. **Predicting the utterance:** BERT, best models retrieve the utterance of the conversation based on that, return documents. **Asking clarifying questions:** document retrieval, question retrieval, and question selection. **Document retrieval:** metric: high precision, given a query question and answer, retrieve documents that are relevant to the conversation. **Question retrieval:** metric: high recall, given a topic and context, retrieve clarifying questions, **Question selection:** Metric: high precision, selecting a clarifying question that leads to retrieval improvement.
Query types: Ambiguous: things that are about the same thing but can be interpreted in different ways (Apple, Jaguar), **Faceted:** different aspects of the same concepts (query: wedding, faceted: wedding cake, organising wedding, etc.). **Ambiguous**: queries benefit most from clarifying questions. **Guided transformers:** get an idea of different interpretations of the same query from different sources. **Learning to rank and conversations:** used for ranking clarifying questions.

Recommender Systems

Evaluation of recommender systems: if ratings are available, an option is to use **Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE).** However, more common to treat it as a ranking problem: precision, recall, NDCG, etc.
Recommendation approaches: challenges: **cold start,** what items to recommend for new users, **Modelling preferences:** Dynamic vs static preferences/ short or long term, users do not retain the same taste, **Exploration vs Exploitation:** recommended based on view/ rating history.

Content-based approach

Recommend items to customer similar to previous items that are rated highly by that same user. Create **Item profiles (**vector of features) for each item (e.g. for a movie: author, title, actor, etc), pick important features with the use of TF-IDF. Create **User profiles:** simple: average of positively rated items profiles. Variant: normalize weights using average rating of user. More sophisticated: use classifies/regressors to predict if a user likes an item. **Recommendation:** suggest items whose feature vector is most similar to profile vector. Cosine similarity is another option. **Advantages of Content-based approach**: user independence, does not need information of other users, can handle unique tastes of user, unpopular items are also recommended, transparency: explanations and straightforward, Cold start (for items) is a non-issue because no recommendations on new items are needed, it is based on content only. **Drawbacks of content-based approach: -** feature engineering is often needed, - often content is not the only factor for users, - overspecialisation, - no unexpected recommendations, cold start (for users) because there is not profile yet.

Collaborative filtering

key idea: if two users have rated items similarly, and user one has rated an item, the other user will have a similar rating even if he did not rate it yet. Content no longer needed. **User-based:** use other users to infer ratings of an item for a user. "neighbour"- users who have similar ratings, **Item-based:** use ratings of similar items (that user has rated) to predict the rating for a given item.
Explicit ratings: like/dislike, IMDB ratings, might not be available. **Implicit ratings:** time spend on a website, clicks. **"Neighbour based model":** used stored ratings directly, nearest neighbour. Drawbacks: limited coverage – users can be neighbours only if they rated common items, Sparsity: makes it worse as number of items increase. **Matrix factorization:** attempts to solve sparsity/coverage problems by projecting user/item vectors to a dense latent space. **Decompose rating matrix:** use Singular value decomposition to find similarities even if not the same movie is rated. **Advantages of Collaborative Filtering:** - no feature engineering needed, - does not rely on content (which may be inaccurate), can handle the overspecialization problem, items recommended may not have similar content. **Drawbacks of Collaborative Filtering:** Neighbourhood based: limited coverage sparsity, - cold start still an issue, popular bias: popular items are often recommended, - cannot recommend to users with unique tastes.

Deep recommenders

Scalable and learn complex interactions, typical to use implicit feedback (lots of data to lean from). **MultVAE:** variational autoencoders for collaborative filtering. Input: users are represented using a binary vector, Dimension: item is 1 if an interaction exists, otherwise 0. **Inference:** encode user history and feed to encoder – decode the representation. (after removing previously interacted items), pick items with the highest score in the output. **Advantages of Deep learning:** - all the advantages that come with DL – powerful, flexible, scalable, - cross-pollination adaptation of advances from ML/DL. **Drawbacks of Deep Learning:** -cold start still a problem, - no consensus if performance gains are significant, can't be reproduced well.

Exploration vs Exploitation

Exploitation: recommend only items that are likely to interest a user. **Exploitation –** recommend other items, learn user's preferences over un-encountered items.