Lecture 22 · Part V · Extended notes

Recommender systems: neural, and evaluated honestly

Dataset: MovieLens 1M, binarised

Why these notes exist

Lectures 19–22 sit outside Géron, the course's primary text. For those four lectures *these notes are the primary source*, and they are examined on exactly the same terms as the textbook parts. Everything in the slide deck for this lecture is here, with the arguments written out at length rather than compressed onto a slide; the numbers are the ones the accompanying notebook prints, and every one of them is a measurement on the corpus named above rather than a figure quoted from a paper.

Contents

1 Where we are **2**

2 From ratings to rankings **3**

2.1 Precisely why RMSE fails here . . . 3

2.2 The data, binarised . . . 3

2.3 The metrics, from Lecture 19 . . . 4

2.3.1 Why already-seen items must be excluded . . . 4

2.4 The baseline that must be in every table . . . 4

3 Two towers **4**

3.1 It is Lecture 20, with the query replaced . . . 5

3.2 What the towers can hold . . . 5

3.3 Training: the softmax over the catalogue . . . 5

4 The protocol is an experimental variable **6**

4.1 Two decisions nobody reports . . . 6

4.2 Decision 1: which interaction is held out . . . 6
4.3 Decision 2: against what is it ranked . . . 7
4.4 The experiment . . . 7
4.5 The table . . . 7
4.6 Take it apart: the split . . . 8
4.7 Take it apart: the sampling . . . 8
4.7.1 Why sampling flatters unequally . . . 8
4.8 How large are modelling gains, for comparison? . . . 8
4.9 What is stable across protocols . . . 9
4.10 What the numbers actually license . . . 9
4.10.1 The mean hides the users . . . 9
4.11 So what should you actually do? . . . 9

5 Popularity bias, and what an offline number is worth 10
5.1 The feedback loop . . . 10
5.2 The long tail, and who is served . . . 10
5.3 What to report, beside the metric . . . 10
5.4 None of it is the real test . . . 11

6 Design exercise: a news app's personalised feed, with freshness 11

7 Exercises 13

8 Reference 14
8.1 Vocabulary . . . 14
8.2 Summary . . . 14

1 Where we are

Lecture 21 predicted ratings, and did it well: an RMSE of 0.8602 from a rank-8 factorisation fitted on the observed ratings alone, against 1.1185 for the global mean, with the bias model and item–item neighbourhoods in between. It ended on implicit feedback, where nobody ever says no. But nobody deploys a rating predictor. A system does not fill in a matrix; it picks ten items out of a catalogue and puts them in front of a person. That is a ranking problem over items the user has not interacted with, and RMSE measures neither of those things. So we change the question, keep the machinery, and discover that the change is not cosmetic.

The architecture in this lecture is the one from Lecture 20; the metrics are the ones from Lecture 19.

What is new is the *experimental design*. There is no derivation: Part V's mathematics was in Lecture 19 and Lecture 21, and this lecture is measurement, which is the other half of the course.

2 From ratings to rankings

Lecture 21 asked what a user would rate a film. This lecture asks what to show them. They are not the same question, and the second is the one a recommender is actually deployed to answer.

	Lecture 21	**Lecture 22**
Question	what would they rate it?	what should we show?
Output	a number per pair	an ordered list of ten
Evaluated on	ratings they gave	items they had not yet touched
Metric	RMSE	HR@10, NDCG@10 — Lecture 19's
Data	explicit ratings	implicit interactions

Row three changes everything. The evaluation set is no longer a sample of what the user did; it is a claim about what they *would* do, on items they never saw.

2.1 Precisely why RMSE fails here

Lecture 21's RMSE of 0.8602 was computed on held-out ratings — pairs where the user chose to watch the film *and* chose to rate it.

- A recommender operates on the complement of that set: films the user has not chosen.
- RMSE weights every prediction equally, and a recommender shows ten items. Being accurate on the three-star films costs nothing and buys nothing.
- And it is a *pointwise* error, whereas showing ten items is a decision about *order*.

A concrete case

A model that predicts every rating 0.3 stars too low has a poor RMSE and a *perfect* ranking: a constant offset changes no order at all. The two metrics can disagree completely.

2.2 The data, binarised

Call a rating of 4 or 5 an interaction and discard the rest:

Ratings	1,000,209
Positive interactions	575,281
As a fraction of ratings	57.5%
Cells with no interaction	97.4%

Note what the threshold did: a film a user rated 3 is now indistinguishable from one they never saw. We have thrown away the only genuine negatives in the dataset, in order to simulate the situation of a real system — which never had any.

2.3 The metrics, from Lecture 19

HR@10 is the hit rate: is the held-out item in the top ten? For one held-out item this is recall@10, and it is 0 or 1 per user. **NDCG@10** is the same, discounted by position: $1/\log_2(\text{rank} + 1)$ when hit, 0 otherwise. Both come straight from Lecture 19's derivation with $|R| = 1$. Nothing new is needed, which is the point of having done retrieval first. One rule is carried over and about to matter: an item the user has already interacted with is removed from the candidates before ranking.

HR@10, defined precisely

1. Hold out exactly one interaction per user; call its item g.
2. Score the candidate set for that user; *remove every item they interacted with in the training data*.
3. Take the top ten. HR = 1 if g is among them, else 0.
4. Average over users — macro, one vote each.

Step 2 is where the candidate set is defined, and the candidate set is half of what this lecture measures.

Why already-seen items must be excluded

A small implementation detail with a large effect, and a good example of how a protocol leaks. Leave them in and the popularity baseline recommends the blockbusters every heavy user has already watched — scoring nothing, but occupying the ten slots. Leave them in for the factorisation and it recommends what the user already interacted with, because that is what it was trained to score highly. Which of the two is hurt more depends on the model, so the choice is *not neutral between them*. We exclude them everywhere, and say so. It is the only line of the popularity baseline that varies by user, and without it the baseline would not be a baseline.

2.4 The baseline that must be in every table

Recommend the ten most popular items. To everybody. Every time. No personalisation, no model, no parameters — minus the items this user has already interacted with, which is the only thing in it that varies by user.

It is not a straw man

A large reproducibility study of eighteen published neural recommenders found most of them matched or beaten by well-tuned classical baselines, and popularity alone beats a substantial fraction of them; by the end of this lecture you will see exactly how that happens. Every number needs a baseline, and here most of all: a recommender that does not beat popularity has not been shown to personalise anything.

3 Two towers

3.1 It is Lecture 20, with the query replaced

$$\text{score}(u, i) = E_{\text{user}}(u) \cdot E_{\text{item}}(i)$$

	Lecture 20	**Here**
Left tower	the query text	the user — id, history, context
Right tower	the document	the item — id, genre, text
Precomputed	document vectors	item vectors
At request time	encode the query	encode the user

Identical architecture, identical indexability argument, identical approximate-nearest-neighbour problem. *Retrieval and recommendation converge here*, and that convergence is why Part V is one part rather than two.

3.2 What the towers can hold

Matrix factorisation is the special case where each tower is a *lookup table*: $E_{\text{user}}(u) = p_u$ and $E_{\text{item}}(i) = q_i$, and there the resemblance ends. A tower can instead be a network over *features*: the item's genre, its text, its age; the user's recent history, device, time of day. Which is what buys the cold start — a brand-new item has no q_i to look up, but it does have a genre and a description.

The honest summary of "neural recommenders"

The architecture is a dot product of two learned vectors, exactly as before. What is new is *what the vectors are computed from* — and on datasets with no features beyond ids, there is nothing for the network to compute from. That is a large part of why the reproducibility results came out as they did.

	Lookup table (MF)	**Feature tower**
New item	no vector at all	encode its genre and text
New user	no vector at all	encode context and first actions
Tail item	estimated from very few interactions	shares parameters with similar items

35.5% of films here have fewer than twenty interactions, so row three is more than a third of the catalogue. *This* — not accuracy on the head — is the honest case for a neural recommender, and it is invisible to a metric dominated by popular items.

3.3 Training: the softmax over the catalogue

$$\mathcal{L} = -\log \frac{\exp(s_{ui})}{\sum_{j \in \text{catalogue}} \exp(s_{uj})}$$

The natural objective: the observed item should win against every other item. It is the cross-entropy of Lecture 17 with the catalogue as the classes.

And it is unaffordable. The denominator runs over 3,706 films here, and over tens of millions in a real catalogue — per interaction, per step. So in practice the denominator is estimated from a sample of the catalogue rather than summed over all of it.

4 The protocol is an experimental variable

4.1 Two decisions nobody reports

To evaluate a recommender you must decide two things, and papers routinely state neither.

1. **Which interaction is held out?** One chosen at random from the user's history, or their last one in time.
2. **Against what is it ranked?** The whole catalogue, or the held-out item against a sample of, say, 100 items the user did not interact with.

Two binary choices, four protocols. We are going to run the same model on the same data under all four.

4.2 Decision 1: which interaction is held out

	Random (leave one out)	**Temporal (leave last)**
Held out	one interaction, chosen uniformly	the user's most recent
Training set contains	the user's future	only their past
Matches deployment	no	yes
In the literature	very common	less common

Lecture 3 defined a leak: a quantity computed from rows outside the training set enters the training path. With time in the data, the training set a deployed system can have is everything before the moment it predicts, and nothing after it. Check the definition against the random split. The model is asked to predict interaction t; it was trained on interactions $t+1, t+2, \ldots$ by that same user; a deployed system has none of them, because they have not happened.

It is a leak, and it looks like a convention

No column was duplicated, no target was copied, nothing was fitted before the split. The leak is in the *shape of the evaluation*, which is why it survived so long as a default — and why its effect is measured below rather than argued.

Failure condition — a random split over interactions

Split a user's interactions at random and the model is trained on what they did afterwards and tested on what they did before. The score improves, the protocol is the reason, and no error is raised because every row is genuinely that user's.

4.3 Decision 2: against what is it ranked

	Full catalogue	Sampled, 100 negatives
Candidates ranked	3,706	101
Cost	score every item	score 101
Matches deployment	yes — the system ranks the catalogue	no
Why anyone does it	—	it was expensive, once

Sampling was a computational shortcut from a time when scoring a catalogue was expensive. Ranking one held-out item against 100 random others is a far easier task than ranking it against 3,706 — and the 100 are drawn uniformly, so they are mostly obscure films nobody would recommend.

4.4 The experiment

One model: a rank-32 factorisation of the binarised interaction matrix — the simplest two-tower model there is, with both towers lookup tables. One dataset: MovieLens 1M, 575,281 positive interactions. One baseline: the most popular items, minus what the user has already seen. Four protocols: $\{\text{random}, \text{temporal}\} \times \{\text{full}, \text{sampled 100}\}$. Everything is held fixed except the protocol: the model is fitted once per split, then scored under both candidate protocols.

Predict the four numbers

You have the pieces: 3,706 films in the catalogue and one right answer; sampling replaces that catalogue with 101 candidates; a random split lets the model see the user's later choices. Which of the two decisions do you expect to matter more? And — the question that turns out to be the interesting one — do you expect the ordering of the two methods to survive both? Both effects are large, and one is much larger. Guess before reading on: a protocol that produces a wrong answer does not feel wrong from the inside.

4.5 The table

HR@10	Full catalogue	Sampled 100
Temporal · factorisation	0.0926	0.6792
Temporal · popularity	0.0404	0.4864
Random · factorisation	0.2132	0.8102
Random · popularity	0.0879	0.6186

The same model on the same data scores 0.0926 and 0.8102 — a factor of 8.8. Nothing about the model changed. Only the two sentences describing how it was scored.

4.6 Take it apart: the split

HR@10, full catalogue	Temporal	Random	Ratio
Factorisation	0.0926	0.2132	×2.30
Popularity	0.0404	0.0879	×2.18

Switching from a temporal to a random split more than doubles both numbers, and does it to the personalised model and the unpersonalised baseline alike.

That last part is the tell

If the gain were the model learning something, it would not lift a popularity list by the same factor. It is *the task getting easier*: predicting a randomly chosen item from a history you have already seen most of.

4.7 Take it apart: the sampling

HR@10, temporal split	Full	Sampled 100	Ratio
Factorisation	0.0926	0.6792	×7.3
Popularity	0.0404	0.4864	×12.0

Sampling is by far the larger effect. HR@10 against 100 random negatives asks "is the right item in the top ten of *101*?" — and ten of 101 is already a tenth of the candidate set. A random ranker scores 0.0990 under this protocol and about 0.0027 against the full catalogue: *the floor moved by a factor of 37*, and every method rides up with it.

Why sampling flatters unequally

The 100 negatives are drawn uniformly, and the catalogue is mostly tail: the top 1% of films hold 12.2% of interactions.

- A uniform sample of 100 therefore contains, in expectation, about one head film — the only items a user might plausibly have chosen instead.
- A popularity model is competing against almost no popular items, so it wins easily.
- A model whose skill is discriminating *among plausible items* has nothing to discriminate.

Failure condition — so sampled metrics are not merely noisy

The distortion depends on what kind of model you have, which is exactly the condition under which a metric can *reverse an ordering*. A bias that were uniform would at least preserve rankings; this one does not.

4.8 How large are modelling gains, for comparison?

To read those effects you need a sense of what a real improvement looks like in this field. A new architecture reporting a 5–10% relative gain on HR@10 is a publishable result. Our two protocol decisions moved HR@10 by 130% and 634% relative, on an unchanged model.

The reproducibility problem, in one comparison

The undocumented choices are an order of magnitude larger than the documented contributions, which is why the reproducibility study found what it found.

4.9 What is stable across protocols

Protocol	Factorisation ÷ popularity
Temporal, full catalogue	×2.29
Random, full catalogue	×2.42
Temporal, sampled 100	×1.40
Random, sampled 100	×1.31

The absolute numbers spanned a factor of 8.8; the ratio to the baseline spans from ×1.31 to ×2.42.

The practical rule

Report the baseline in the same table, under the same protocol. It does not repair the protocol, and it does make your number readable by someone whose protocol differs — which is everyone.

4.10 What the numbers actually license

Under the honest protocol — temporal, full catalogue — the factorisation scores 0.0926 and popularity 0.0404. The model is worth 2.3 times the baseline, and *that* is the real result. Every other number in the table is larger and means less. NDCG@10 tells the same story: 0.0464 against 0.0196.

The honest headline

"A rank-32 factorisation places the user's next film in the top ten 9.3% of the time, against 4.0% for a popularity list, under a temporal split with the full catalogue ranked."
Every clause is load bearing.

The mean hides the users

HR@10 per user is 0 or 1, so a mean of 0.0926 means precisely this: the held-out film was in the top ten for 9.3% of users and not for the rest. There is no partial credit and no distribution to inspect — the mean *is* the proportion, which makes it honest and makes its confidence interval computable, since it is a proportion over 6,038 users. NDCG@10 does have a spread, and it is rarely reported. "Report the variance" is good advice that means different things for different metrics: for HR the right companion is an interval, for NDCG a distribution.

4.11 So what should you actually do?

1. Rank the full catalogue. It is affordable now; the shortcut is a historical artefact.

2. Split in time, per user, and say so.
3. Include popularity and a tuned classical baseline in the same table.
4. Never quote a number from another paper into your table without re-running it under your protocol.
5. Report coverage and the popularity profile beside the accuracy metric.

All five are cheap. None of them is standard. Every one of them can only make your headline number smaller, which is the whole difficulty.

5 Popularity bias, and what an offline number is worth

5.1 The feedback loop

A recommender is not an observer of its data. It *produces* it. The system recommends popular items because they are the safe bet; users interact with what they are shown, since they cannot interact with what they are not; those interactions become the next training set; and the popular items become more popular still.

Failure condition — nothing in the objective notices

Every offline metric in this lecture is computed on interactions produced by whatever system was running at the time. A model that reproduces the old system's behaviour scores well, and a model that would have shown something better scores badly — because the evidence for that something was never collected.

5.2 The long tail, and who is served

The top 1% of films hold 12.2% of interactions, and 35.5% of films have fewer than twenty. Recommending from the head is correct often enough to score well — popularity alone reaches 0.0404 honestly measured — and it is worth nothing to the user, who had already heard of those films.

The metric cannot see usefulness

HR@10 gives full credit for a correct popular recommendation and the same credit for a correct obscure one. If what you value is discovery, you must measure something else as well — coverage, the popularity distribution of what you recommend, the fraction of the catalogue ever shown.

5.3 What to report, beside the metric

- **Catalogue coverage** — what fraction of items are ever in anyone's top ten. A model with a great HR@10 and 2% coverage is a popularity list with extra steps.
- **Popularity profile** — the distribution of recommended items' interaction counts, against the catalogue's.
- **Per-user spread**, not just the mean — a mean HR of 0.09 can be most users at zero and a few at one.

- **The protocol**, in the sentence that states the number.

None of these is hard. They are absent from papers because nothing forces them to be present, and because each of them can only make the headline look worse.

5.4 None of it is the real test

Every number in this lecture is *offline*: computed on a log, about a system that was never run. The only real measurement is an online experiment with users, and it measures something different — whether people engaged, returned, cancelled. Offline and online results correlate weakly, and the literature is candid about that. An offline metric is a *filter*: cheap enough to reject bad ideas before they reach users, not evidence that a good one works.

The honest position

Use offline evaluation to decide what is worth testing. Use an online test to decide what is worth keeping. Do not use either to make the claim the other one supports.

What to change

Raise the sampled negatives from 100 to 1,000 and re-run. Predict the direction first: the sampled column should move towards the full-catalogue one, because sampling is the same metric at a different candidate-set size.

6 Design exercise: a news app's personalised feed, with freshness

The lecture closes with a design exercise in the manner of an ML system-design interview, on a product with the shape of data this model suits.

The task. A news company's app opens on a "for you" feed of ten stories, chosen per reader from the articles live right now — and an article is live for a day or two, so the catalogue the model ranks is never the one it was trained on. The output is a ranking over items the reader has not touched, scored on the ten slots the app shows; the business measures opens, time in the app, whether the reader returns the next day, and subscriptions. Given: the log of impressions and clicks — which stories each reader was shown, in which slot, and which they opened — plus each article's headline, section and publication time. Millions of readers, thousands of live articles at any moment, a catalogue that turns over daily. The output feeds the feed, the push notification, and the newsroom.

1. **Frame it.** The question is what should we show, not what would they rate it: an ordered list of ten, evaluated on articles the reader had not touched, by the hit rate and NDCG at ten — ten because the feed shows ten, and a hit at rank forty is a story nobody saw. The baseline in every table is the most-clicked live articles, the same for everybody, minus what this reader already opened. *Pitfall:* choosing the cut-off for the number. Recall at a hundred is larger and kinder to every model, and it measures a page nobody is shown, so a release approved on it changes nothing a reader sees. Fix the cut-off to the product before the first run, and report the metric with its protocol in the same sentence: which interaction was held out, and against what it was ranked.

2. **The log.** The signal is implicit: a click is an interaction, and a story not clicked was never shown, or shown and passed over, or shown below the fold — no negatives, only the absence of positives. Keep the impressions with the slot each story occupied, because unlike a ratings dataset this log knows what the reader was offered; a reader's profile is their recent clicks, and last month's are a different reader. *Pitfall:* position bias. The top slot is clicked most because it is the top slot, so a model trained on raw clicks learns what the previous ranker put first and reproduces it — the feed produces its own training data, and the same stories stay pinned. Train on clicks given the slot, comparing a story with the others shown in the same position, and keep a small share of impressions whose order was shuffled, so that evidence exists which the old ranker did not choose.

3. **Split in time, per reader.** Hold out each reader's last click and train on everything before it: the training set is the reader's past, which is all a deployed system ever has. Candidates at test time are the articles live at that moment, nothing published later; protocol choices are made on a validation slice of readers, and the test readers are read once. *Pitfall:* holding out one click at random. The model is then trained on what the reader opened after the click it is asked to predict — a leak in the shape of the evaluation, with no column duplicated and no error raised. On the course's own data a random split more than doubled the hit rate of the model and of the popularity list alike, the tell that the task got easier, not the model; the gain promised to the product team never reaches the app.

4. **Two towers.** $\text{score}(u, i) = E_{\text{user}}(u) \cdot E_{\text{item}}(i)$: the reader encoded at request time, every live article encoded as it is published and stored. The item tower reads the headline, section, source and age; the user tower reads the recent clicks, the time of day and the device. Matrix factorisation is the special case where each tower is a lookup table on an id — useless here, because every article is new. *Pitfall:* an id embedding per article. A story published an hour ago has no row in the table, so the model has no vector for what it is supposed to recommend, and the story the newsroom just filed cannot reach a feed: the cold start is not an edge case in news, it is the whole catalogue. The feature tower is what buys freshness, and it must be measured on articles the model never saw in training, which the time split guarantees.

5. **Negatives.** The natural objective says the clicked story should win against every other live story — the cross-entropy with the catalogue as the classes — and it is unaffordable per click, per step, so the denominator is estimated from a sample of the catalogue. The choice of sample is a modelling decision and should be reported: sample from what was shown and passed over, and from the live catalogue weighted by exposure, not uniformly. *Pitfall:* what a sampled negative assumes. It assumes the reader would not have clicked that story. Sampled uniformly from the catalogue, the negatives are obscure stories nobody would have been shown, so the model learns to beat the tail and nothing about choosing among the plausible; and a story the reader never saw may be the one they would have opened. Being unclicked is not evidence of disinterest: say what the sample assumes, and measure what it costs.

6. **Evaluate honestly.** The honest protocol holds out the reader's last click, ranks it against every live article, puts popularity in the same row under the same protocol, and reads the hit rate as a proportion of readers with its interval; the flattering one holds out a click at random, ranks against a hundred sampled articles, quotes a number from a paper, and reports a mean alone. *Pitfall:* ranking the held-out click against a hundred random articles. The floor rises by an order of magnitude, and unequally: the sample holds almost no popular stories, so popularity wins easily and a model whose skill is choosing among plausible stories

has nothing to choose among. A sampled metric can reverse the order of two systems and ship the worse one. Rank the full live catalogue, and beat popularity, or nothing has been personalised.

7. **Diversity, as quantities.** Coverage: the share of live articles that ever reach a feed — a fine hit rate with tiny coverage is the front page with extra steps. The popularity profile: the click counts of what is recommended, against the catalogue's. Per reader: how many sections the ten slots come from, against the sections that reader has ever opened — the bubble measured as a number the newsroom can track. *Pitfall:* the hit rate gives full credit for a correct popular story and the same credit for a correct discovery, and a feed that narrows to one section scores well on the clicks it caused, until the subscriber cancels because the app "only shows sport". If breadth is what the product values it must be measured, because nothing in the objective will notice. Report coverage and the per-reader spread beside the headline, and let the policy stage enforce a floor on sections.

8. **In production.** Every offline number comes from a log produced by whatever ranker was running: a model that reproduces it scores well, and one that would have shown something better scores badly, because the evidence was never collected. Use offline evaluation to decide what is worth testing; roll out to a share of readers behind an A/B test — opens, time spent, whether they return the next day — to decide what is worth keeping. Retrain daily, since the catalogue has turned over, and on call watch coverage and the share of feeds served by the popularity fallback. *Pitfall:* the offline–online gap. Shipping on an offline gain: offline and online results correlate weakly, and the feed also changes the readers it is measured on — it recommends popular stories, readers click what they are shown, those clicks train the next model, and the popular grow more popular. Read the A/B test on what readers do afterwards, and watch coverage week on week for the loop closing.

9. **Ship it.** Impressions and clicks with their slot → last click held out per reader, candidates the live articles → feature towers for reader and article → sampled softmax, negatives from what was shown → hit rate at ten over the full live catalogue, popularity in the same row → coverage and per-reader spread → A/B test on a share of readers → retrain daily, monitor.

7 Exercises

The five set at the end of this lecture's deck, in the style of the written paper. The solutions are at the end of this lecture's deck and in the exercise book.

1. A recommender is scored by RMSE on held-out ratings. State why that is the wrong metric, using the set the model actually operates on. [5]

2. The same model scores HR@10 of 0.0926 and 0.8102 on the same data. Name the two decisions that differ, and which moves the number more. [5]

3. Explain why a sampled metric is not merely noisier than a full-catalogue one, and give the consequence for comparing two published systems. [5]

4. A new film arrives with a genre and a description but no interactions. State what a matrix factorisation can do with it, what a two-tower model with a feature tower can do, and why. [4]

5. A recommender is trained on interactions produced by an earlier recommender. Name the effect, and say why no offline metric computed on that log can detect it. [4]

8 Reference

8.1 Vocabulary

Two-tower	user encoder and item encoder, scored by a dot product
Leave-one-out	hold out one interaction per user, chosen at random
Temporal split	hold out the most recent interaction instead
Sampled metric	rank the held-out item against a sample of negatives
Popularity bias	systematic over-recommendation of already-popular items
Feedback loop	the system shaping the data it will next be trained on

8.2 Summary

A recommender ranks a catalogue for a user, so it is scored by Lecture 19's metrics on items the user had not touched — not by RMSE. The two-tower model is Lecture 20's bi-encoder with users in place of queries; matrix factorisation is its lookup-table special case, and feature towers are what buy the cold start. The protocol moved HR@10 from 0.0926 to 0.8102 on one unchanged model; the ratio to a popularity baseline under the same protocol is far steadier, so it goes in the same table. Popularity bias is a feedback loop no offline metric can see; offline numbers filter ideas, and only an online test measures the thing you care about. The closing design exercise built a news app's personalised feed with freshness in nine steps — the metric at the ten slots the app shows, clicks read given their slot, the last click held out per reader, an item tower built from features because every article is new, negatives sampled from what was shown, the full live catalogue ranked with popularity in the same row, coverage and sections per reader as the bubble measured, and an A/B test on a share of readers before anything is kept — each step named for the pitfall it avoids.

Failure condition — the protocol, not the model

One unchanged model reports HR@10 of 0.0926 or 0.8102 depending only on how it was evaluated. A recommender's headline is a property of the evaluation protocol at least as much as of the model, and the protocol is the half that goes unreported.