Relevance Feedback in Informational Retrieval

Evgeniya Sukhodolskaya • March 27, 2025

> A problem well stated is a problem half solved.

This quote applies as much to life as it does to information retrieval.

With a well-formulated query, retrieving the relevant document becomes trivial. In reality, however, most users struggle to precisely define what they are searching for.

While users may struggle to formulate a perfect request — especially in unfamiliar topics — they can easily judge whether a retrieved answer is relevant or not.

Relevance is a powerful feedback mechanism for a retrieval system to iteratively refine results in the direction of user interest.

In 2025, with social media flooded with daily AI breakthroughs, it almost seems like information retrieval is solved, agents can iteratively adjust their search queries while assessing the relevance.

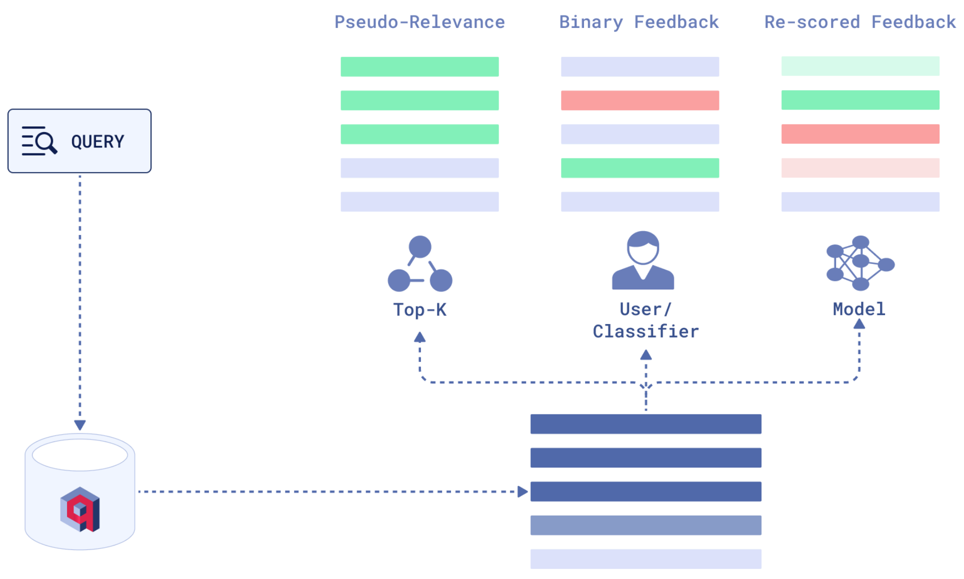
Of course, there's a catch: these models still rely on retrieval systems (*RAG isn't dead yet, despite daily predictions of its demise*). They receive only a handful of top-ranked results provided by a far simpler and cheaper retriever. As a result, the success of guided retrieval still mainly depends on the retrieval system itself.

So, we should find a way of effectively and efficiently incorporating relevance feedback directly into a retrieval system. In this article, we'll explore the approaches proposed in the research literature and try to answer the following question:

If relevance feedback in search is so widely studied and praised as effective, why is it practically not used in dedicated vector search solutions?

Dismantling the Relevance Feedback

Both industry and academia tend to reinvent the wheel here and there. So, we first took some time to study and categorize different methods — just in case there was something we could plug directly into Qdrant. The resulting taxonomy isn't set in stone, but we aim to make it useful.



Types of Relevance Feedback

Pseudo-Relevance Feedback (PRF)

Pseudo-Relevance feedback takes the top-ranked documents from the initial retrieval results and treats them as relevant. This approach might seem naive, but it provides a noticeable performance boost in lexical retrieval while being relatively cheap to compute.

Binary Relevance Feedback

The most straightforward way to gather feedback is to ask users directly if document is relevant. There are two main limitations to this approach:

First, users are notoriously reluctant to provide feedback. Did you know that Google once had an upvote/downvote mechanism on search results but removed it because almost no one used it?

Second, even if users are willing to provide feedback, no relevant documents might be present in the initial retrieval results. In this case, the user can't provide a meaningful signal.

Instead of asking users, we can ask a smart model to provide binary relevance judgements, but this would limit its potential to generate granular judgements.

Re-scored Relevance Feedback

We can also apply more sophisticated methods to extract relevance feedback from the top-ranked documents - machine learning models can provide a relevance score for each document.

The obvious concern here is twofold:

1. How accurately can the automated judge determine relevance (or irrelevance)?
2. How cost-efficient is it? After all, you can't expect GPT-4o to re-rank thousands of documents for every user query — unless you're filthy rich.

Nevertheless, automated re-scored feedback could be a scalable way to improve search when explicit binary feedback is not accessible.

Has the Problem Already Been Solved?

Digging through research materials, we expected anything else but to discover that the first relevance feedback study dates back *sixty years*. In the midst of the neural search bubble, it's easy to forget that lexical (term-based) retrieval has been around for decades. Naturally, research in that field has had enough time to develop.

Neural search — aka vector search — gained traction in the industry around 5 years ago. Hence, vector-specific relevance feedback techniques might still be in their early stages, awaiting production-grade validation and industry adoption.

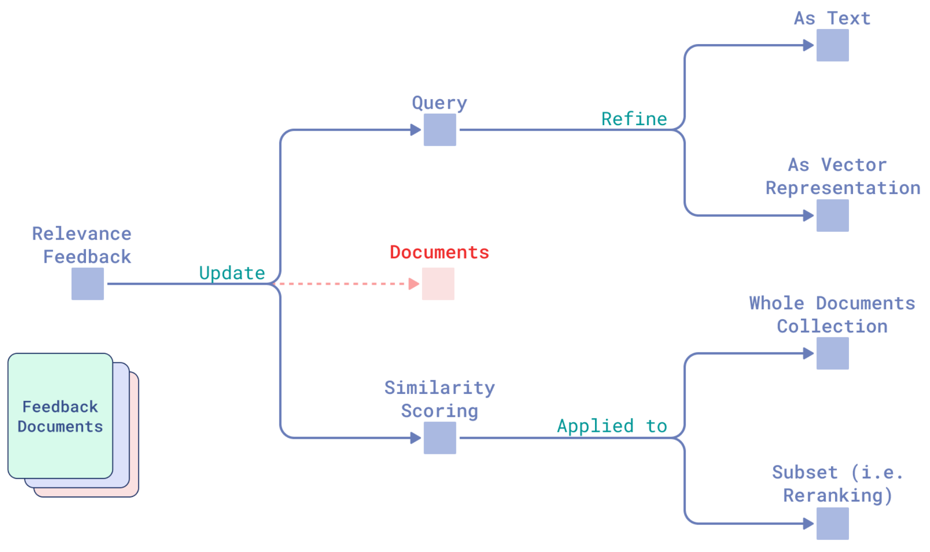
As a dedicated vector search engine, we would like to be these adopters. Our focus is neural search, but approaches in both lexical and neural retrieval seem worth exploring, as cross-field studies are always insightful, with the potential to reuse well-established methods of one field in another.

We found some interesting methods applicable to neural search solutions and additionally revealed a **gap in the neural search-based relevance feedback approaches**. Stick around, and we'll share our findings!

Two Ways to Approach the Problem

Retrieval as a recipe can be broken down into three main ingredients:

1. Query
2. Documents
3. Similarity scoring between them.



Research Field Taxonomy Overview

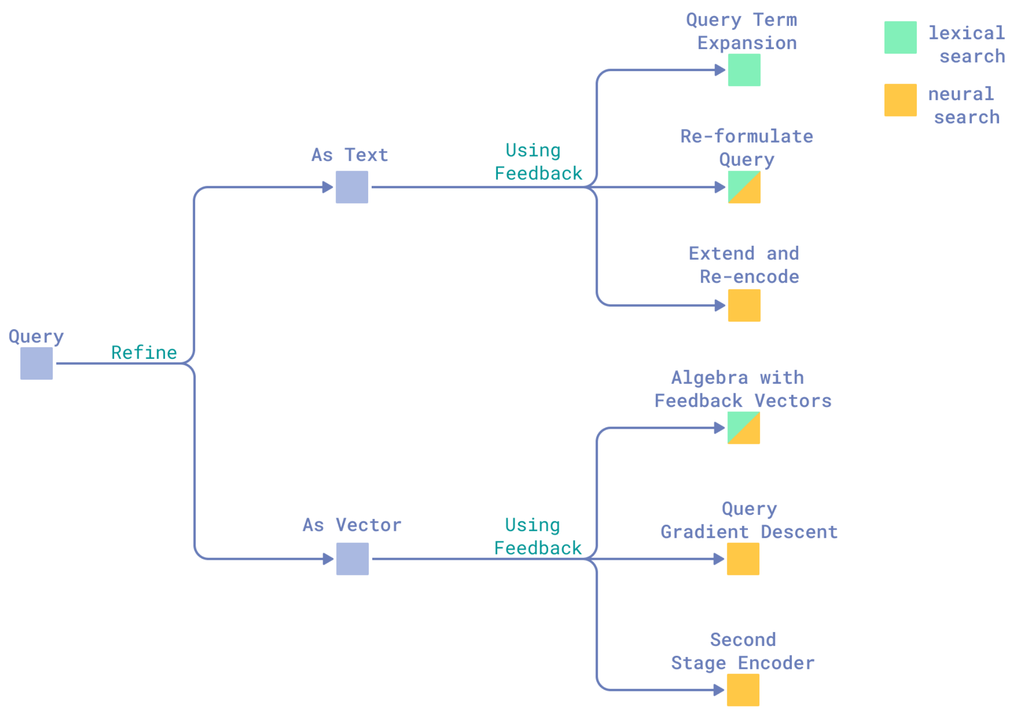
Query formulation is a subjective process — it can be done in infinite configurations, making the relevance of a document unpredictable until the query is formulated and submitted to the system.

So, adapting documents (or the search index) to relevance feedback would require per-request dynamic changes, which is impractical, considering that modern retrieval systems store billions of documents.

Thus, approaches for incorporating relevance feedback in search fall into two categories: **refining a query** and **refining the similarity scoring function** between the query and documents.

Query Refinement

There are several ways to refine a query based on relevance feedback. Globally, we prefer to distinguish between two approaches: modifying the query as text and modifying the vector representation of the query.



Incorporating Relevance Feedback in Query

Query As Text

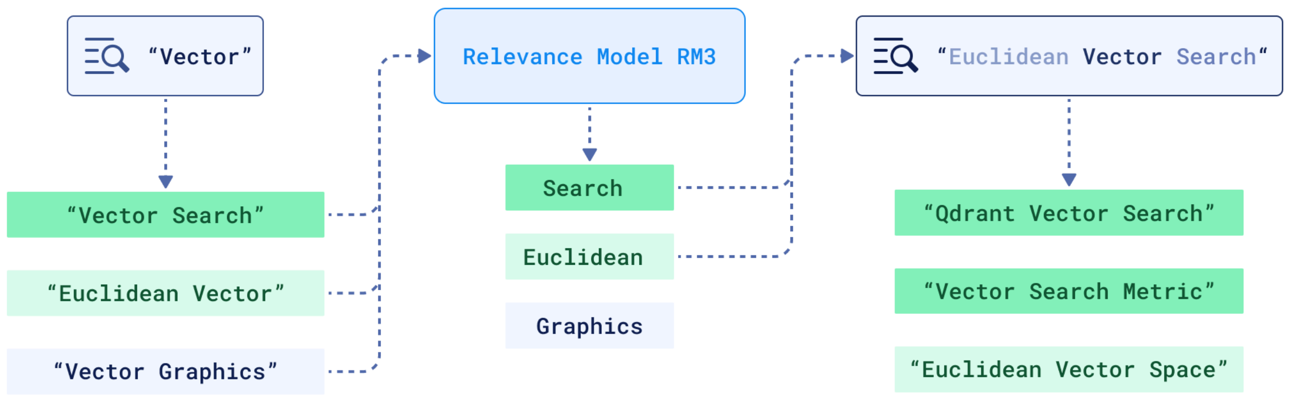
In **term-based retrieval**, an intuitive way to improve a query would be to **expand it with relevant terms**. It resembled the "*aha, so that's what it's called*" stage in the discovery search.

Before the deep learning era of this century, expansion terms were mainly selected using statistical or probabilistic models. The idea was to:

1. Either extract the **most frequent** terms from (pseudo-)relevant documents;
2. Or the **most specific** ones (for example, according to IDF);
3. Or the **most probable** ones (most likely to be in query according to a relevance set).

Well-known methods of those times come from the family of Relevance Models, where terms for expansion are chosen based on their probability in pseudo-relevant documents (how often terms appear) and query terms likelihood given those pseudo-relevant documents - how strongly these pseudo-relevant documents match the query.

The most famous one, `RM3` — interpolation of expansion terms probability with their probability in a query — is still appearing in papers of the last few years as a (noticeably decent) baseline in term-based retrieval, usually as part of anserini.



Simplified Query Expansion

With the time approaching the modern machine learning era, multiple studies began claiming that these traditional ways of query expansion are not as effective as they could be.

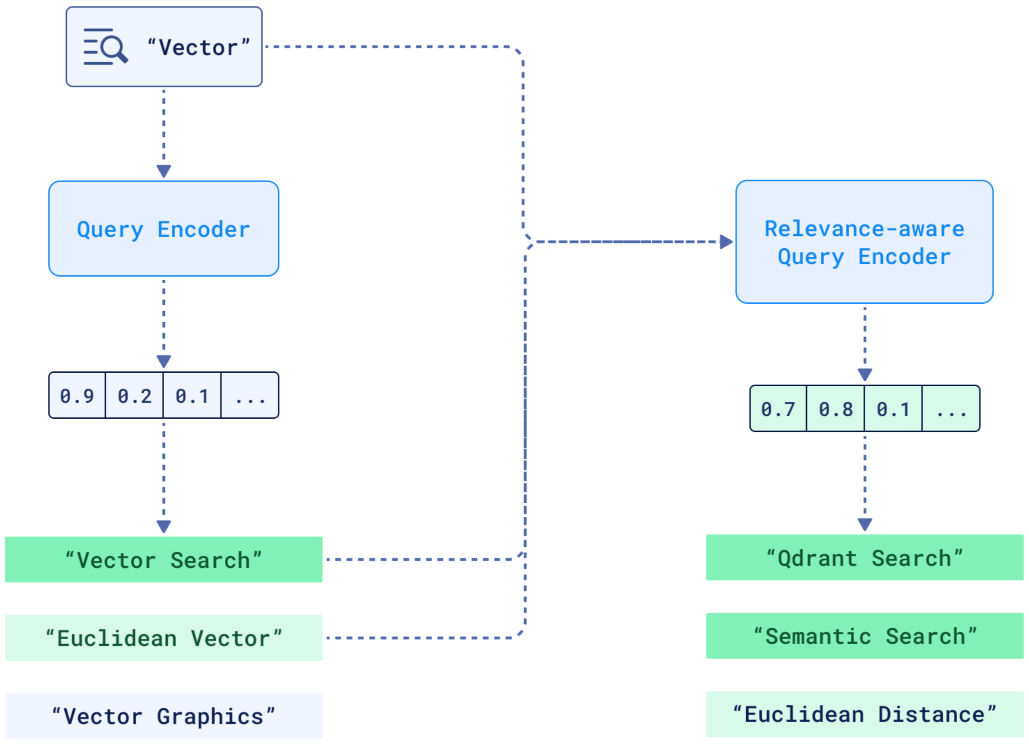
Started with simple classifiers based on hand-crafted features, this trend naturally led to use the famous BERT (Bidirectional encoder representations from transformers). For example, `BERT-QE` (Query Expansion) authors came up with this schema:

1. Get pseudo-relevance feedback from the finetuned BERT reranker (~10 documents);
2. Chunk these pseudo-relevant documents (~100 words) and score query-chunk relevance with the same reranker;
3. Expand the query with the most relevant chunks;
4. Rerank 1000 documents with the reranker using the expanded query.

This approach significantly outperformed BM25 + RM3 baseline in experiments (+11% NDCG@20). However, it required **11.01x** more computation than just using BERT for reranking, and reranking 1000 documents with BERT would take around 9 seconds alone.

Query term expansion can *hypothetically* work for neural retrieval as well. New terms might shift the query vector closer to that of the desired document. However, this approach isn't guaranteed to succeed. Neural search depends entirely on embeddings, and how those embeddings are generated — consequently, how similar query and document vectors are — depends heavily on the model's training.

It definitely works if **query refining is done by a model operating in the same vector space**, which typically requires offline training of a retriever. The goal is to extend the query encoder input to also include feedback documents, producing an adjusted query embedding. Examples include `ANCE-PRF` and `ColBERT-PRF` — ANCE and ColBERT fine-tuned extensions.



Generating a new relevance-aware query vector

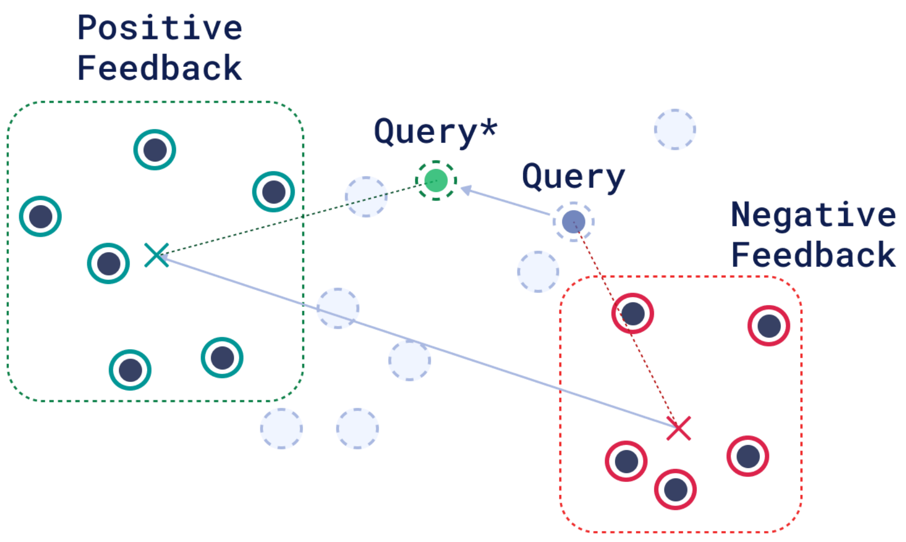
The reason why you're most probably not familiar with these models — their absence in the industry — is that their **training** itself is a **high upfront cost**, and even though it was "paid", these models struggle with generalization, performing poorly on out-of-domain tasks (datasets they haven't seen during training). Additionally, feeding an attention-based model a lengthy input (query + documents) is not a good practice in production settings (attention is quadratic in the input length), where time and money are crucial decision factors.

Alternatively, one could skip a step — and work directly with vectors.

Query As Vector

Instead of modifying the initial query, a more scalable approach is to directly adjust the query vector. It is easily applicable across modalities and suitable for both lexical and neural retrieval.

Although vector search has become a trend in recent years, its core principles have existed in the field for decades. For example, the SMART retrieval system used by Rocchio in 1965 for his relevance feedback experiments operated on bag-of-words vector representations of text.



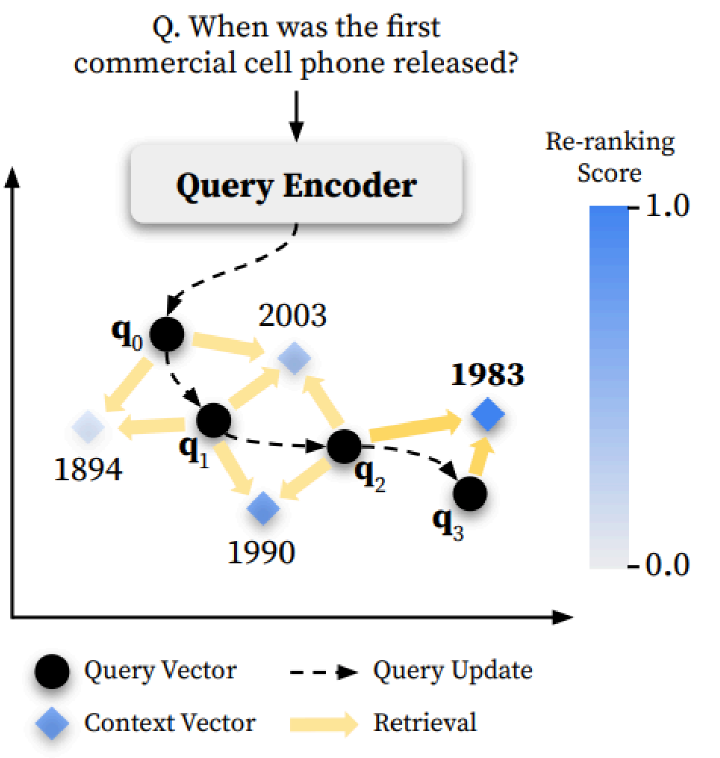
Roccio's Relevance Feedback Method

Rocchio's idea — to update the query vector by adding a difference between the centroids of relevant and non-relevant documents — seems to translate well to modern dual encoders-based dense retrieval systems. Researchers seem to agree: a study from 2022 demonstrated that the parametrized version of Rocchio's method in dense retrieval consistently improves Recall@1000 by 1—5%, while keeping query processing time suitable for production — around 170 ms.

However, parameters (centroids and query weights) in the dense retrieval version of Roccio's method must be tuned for each dataset and, ideally, also for each request.

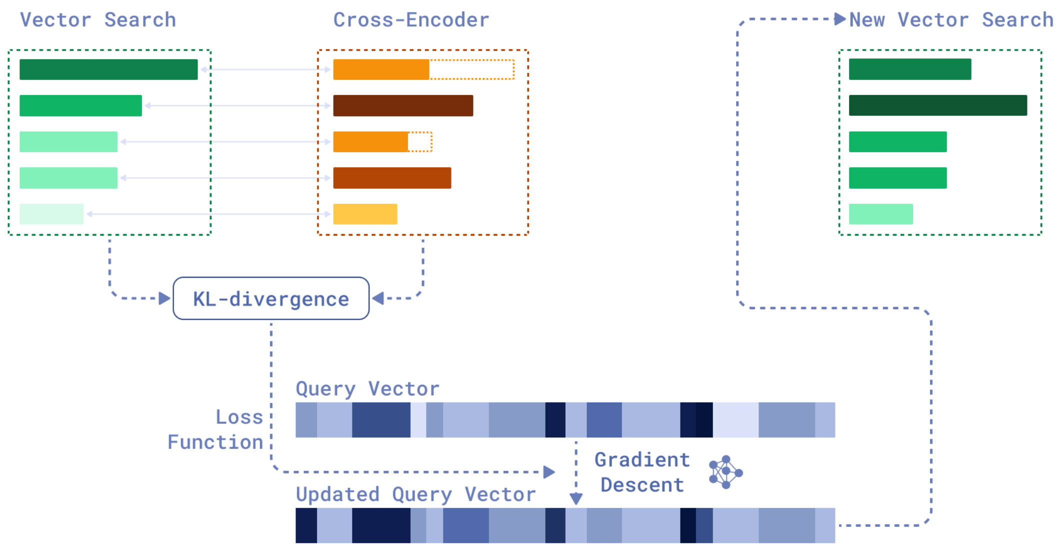
Gradient Descent-Based Methods

The efficient way of doing so on-the-fly remained an open question until the introduction of a **gradient-descent-based Roccio's method generalization**: `Test-Time Optimization of Query Representations (TOUR)`. TOUR adapts a query vector over multiple iterations of retrieval and reranking (*retrieve → rerank → gradient descent step*), guided by a reranker's relevance judgments.



An overview of TOUR iteratively optimizing initial query representation based on pseudo relevance feedback. Figure adapted from Sung et al., 2023, Optimizing Test-Time Query Representations for Dense Retrieval

The next iteration of gradient-based methods of query refinement — `ReFit` — proposed in 2024 a lighter, production-friendly alternative to TOUR, limiting *retrieve → rerank → gradient descent* sequence to only one iteration. The retriever's query vector is updated through matching (via Kullback—Leibler divergence) retriever and cross-encoder's similarity scores distribution over feedback documents. ReFit is model- and language-independent and stably improves Recall@100 metric on 2—3%.



An overview of ReFit, a gradient-based method for query refinement

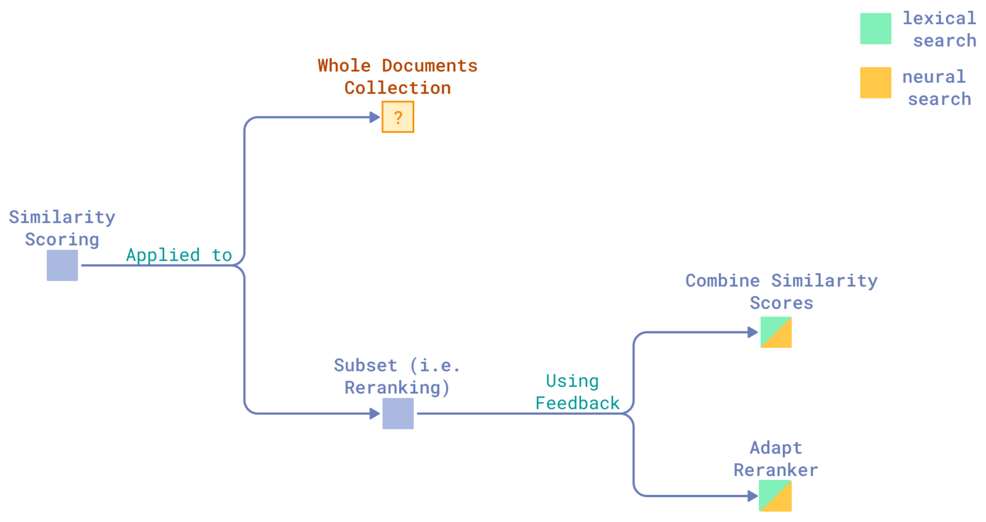
Gradient descent-based methods seem like a production-viable option, an alternative to finetuning the retriever (distilling it from a reranker). Indeed, it doesn't require in-advance training and is compatible with any re-ranking models.

However, a few limitations baked into these methods prevented a broader adoption in the industry.

The gradient descent-based methods modify elements of the query vector as if it were model parameters; therefore, they require a substantial amount of feedback documents to converge to a stable solution.

On top of that, the gradient descent-based methods are sensitive to the choice of hyperparameters, leading to **query drift**, where the query may drift entirely away from the user's intent.

Similarity Scoring



Incorporating Relevance Feedback in Similarity Scoring

Another family of approaches is built around the idea of incorporating relevance feedback directly into the similarity scoring function. It might be desirable in cases where we want to preserve the original query intent, but still adjust the similarity score based on relevance feedback.

In **lexical retrieval**, this can be as simple as boosting documents that share more terms with those judged as relevant.

Its **neural search counterpart** is a `k-nearest neighbors-based method` that adjusts the query-document similarity score by adding the sum of similarities between the candidate document and all known relevant examples. This technique yields a significant improvement, around 5.6 percentage points in NDCG@20, but it requires explicitly labelled (by users) feedback documents to be effective.

In experiments, the knn-based method is treated as a reranker. In all other papers, we also found that adjusting similarity scores based on relevance feedback is centred around reranking — **training or finetuning rerankers to become relevance feedback-aware**. Typically, experiments include cross-encoders, though simple classifiers are also an option. These methods generally involve rescoring a broader set of documents retrieved during an initial search, guided by feedback from a smaller top-ranked subset. It is not a similarity matching function adjustment per se but rather a similarity scoring model adjustment.

Methods typically fall into two categories:

1. **Training rerankers offline** to ingest relevance feedback as an additional input at inference time, as here — again, attention-based models and lengthy inputs: a production-deadly combination.
2. **Finetuning rerankers** on relevance feedback from the first retrieval stage, as Baumgärtner et al. did, finetuning bias parameters of a small cross-encoder per query on 2k, k={2, 4, 8} feedback documents.

The biggest limitation here is that these reranker-based methods cannot retrieve relevant documents beyond those returned in the initial search, and using rerankers on thousands of documents in production is a no-go — it's too expensive. Ideally, to avoid that, a similarity scoring function updated with relevance feedback should be used directly in the second retrieval iteration. However, in every research paper we've come across, retrieval systems are **treated as black boxes** — ingesting queries, returning results, and offering no built-in mechanism to modify scoring.

So, what are the takeaways?

Pseudo Relevance Feedback (PRF) is known to improve the effectiveness of lexical retrievers. Several PRF-based approaches — mainly query terms expansion-based — are successfully integrated into traditional retrieval systems. At the same time, there are **no known industry-adopted analogues in neural (vector) search dedicated solutions**; neural search-compatible methods remain stuck in research papers.

The gap we noticed while studying the field is that researchers have **no direct access to retrieval systems**, forcing them to design wrappers around the black-box-like retrieval oracles. This is sufficient for query-adjusting methods but not for similarity scoring function adjustment.

Perhaps relevance feedback methods haven't made it into the neural search systems for trivial reasons — like no one having the time to find the right balance between cost and efficiency.

Getting it to work in a production setting means experimenting, building interfaces, and adapting architectures. Simply put, it needs to look worth it. And unlike 2D vector math, high-dimensional vector spaces are anything but intuitive. The curse of dimensionality is real. So is query drift. Even methods that make perfect sense on paper might not work in practice.

A real-world solution should be simple. Maybe just a little bit smarter than a rule-based approach, but still practical. It shouldn't require fine-tuning thousands of parameters or feeding paragraphs of text into transformers. **And for it to be effective, it needs to be integrated directly into the retrieval system itself.**
