Efficient and Effective Tree-based and Neural Learning to Rank*

Sebastian Bruch
Pinecone, U.S.A.
sbruch@acm.org

Claudio Lucchese
Ca' Foscari University of Venice, Italy
claudio.lucchese@unive.it

Franco Maria Nardini
ISTI-CNR, Pisa, Italy
francomaria.nardini@isti.cnr.it

May 16, 2023

Abstract

As information retrieval researchers, we not only develop algorithmic solutions to hard problems, but we also insist on a proper, multifaceted evaluation of ideas. The literature on the fundamental topic of retrieval and ranking, for instance, has a rich history of studying the effectiveness of indexes, retrieval algorithms, and complex machine learning rankers, while at the same time quantifying their computational costs, from creation and training to application and inference. This is evidenced, for example, by more than a decade of research on efficient training and inference of large decision forest models in Learning to Rank (LtR). As we move towards even more complex, deep learning models in a wide range of applications, questions on efficiency have once again resurfaced with renewed urgency. Indeed, efficiency is no longer limited to time and space; instead it has found new, challenging dimensions that stretch to resource-, sample- and energy-efficiency with ramifications for researchers, users, and the environment.

This monograph takes a step towards promoting the study of efficiency in the era of neural information retrieval by offering a comprehensive survey of the literature on efficiency and effectiveness in ranking, and to a limited extent, retrieval. This monograph was inspired by the parallels that exist between the challenges in neural network-based ranking solutions and their predecessors, decision forest-based LtR models, as well as the connections between the solutions the literature to date has to offer. We believe that by understanding the fundamentals underpinning these algorithmic and data structure solutions for containing the contentious relationship between efficiency and effectiveness, one can better identify future directions and more efficiently determine the merits of ideas. We also present what we believe to be important research directions in the forefront of efficiency and effectiveness in retrieval and ranking.

*Preprint of article accepted for publication in Foundations and Trends® in Information Retrieval

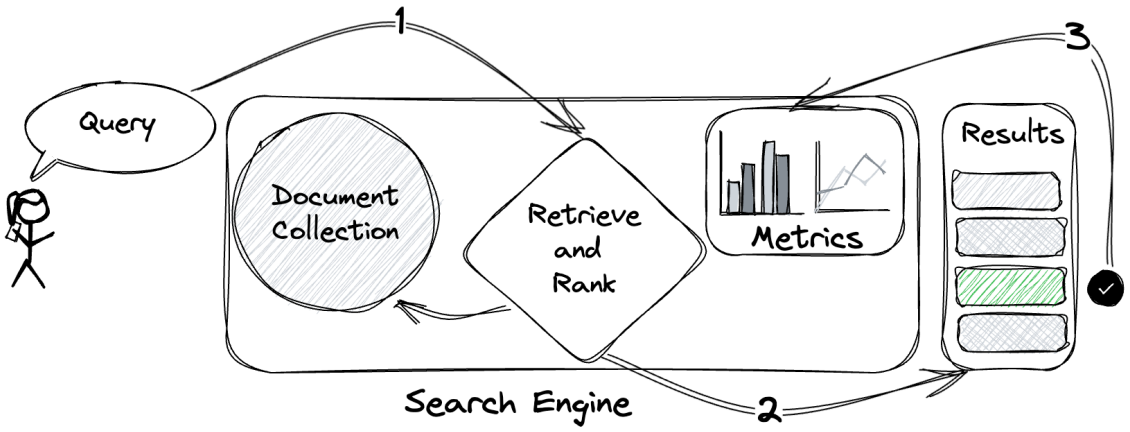


Figure 1.1: The Document Ranking Problem in the context of web search—our running example. The user sends a text query to the search engine (1), which, in turn, *retrieves* the most relevant documents from a large collection, and presents them as a *ranked* list (2). The user then decides if and to what extent the ranked list satisfies their information need, which affects metrics of interest (3).

1 Introduction

Search engines are a familiar tool to the reader of this manuscript. In fact, you have likely arrived at this copy by typing a few keywords into one and perusing the relevant links and page descriptions in its results page. Indeed, the abundance of data on the web makes search engines an integral tool, without which it would be nearly impossible to discover the right information and satisfy an information need.

We similarly rely on a suite of other algorithmic tools to get what is pertinent to us, such as discovering news articles, movies, or songs (recommendation systems), getting answers to natural language questions (question answering and conversational agents), finding images depicting a given description (image search), and many more. What all of these tools have in common is that they are different manifestations of the *retrieval and ranking* problem, which seeks to discover a *set of relevant items* from a large collection and order them according to some *criteria* and with respect to some *context*.

Definition 1.0.1 (The Document Ranking Problem)**.** Given a query q (*context*) and a set of documents D (*items*), the goal is to order elements of D such that the resulting ranked list maximizes a user satisfaction metric Q (*criteria*).

We take web search as the theme of this monograph and delve into the ranking problem in that context. In document ranking, the query q is an intent expressed (often briefly) as a set of textual keywords or in natural language, the documents D are (possibly long) texts

written in natural language, and Q is any utility metric that captures the relevance of an ordered list to q. We have illustrated this setup in Figure 1.1.

Document ranking presents a number of unique questions that are the subject of much research in the field of information retrieval: How do we define Q to quantify the perceived quality of a ranked list and its utility to a user? How do we capture and interpret implicit, noisy, and sometimes circular user preferences, which are represented by clicks? And, more pertinent to this monograph, how do we arrive at a ranked list given a query, a set of documents, a metric, possibly subject to a set of other constraints?

Over a decade ago, machine learning transformed how we approach the document ranking problem and answer the questions above. That wave resulted in a paradigm shift from early statistical methods, heuristics, and hand-crafted rules to determine the relevance of documents to a query, to what would later be called Learning to Rank (LtR) (Liu, 2009), where the relevance of a document to a query is estimated by a learnt function, hence "learning" to rank. This leap was perhaps best exemplified by LambdaMART (Burges, 2010) in the Yahoo! Learning-to-Rank Challenge (Chapelle and Chang, 2011).

This transformation of the document ranking problem culminated in a framework that comprises of two distinct algorithms, depicted in Figure 1.2: *top-k retrieval*, which finds a *subset* of k documents that are more relevant to a query, followed by *ranking* which orders the documents in the top-k set. In LtR, the ranking stage uses an often expensive function that was trained using supervised or online learning methods, while the retrieval algorithm solves a form of the maximum inner product search (MIPS) problem. As we will describe later, in "dense retrieval," retrieval is often (but not always) an approximate nearest neighbor search while ranking is the identity function.

1.1 The importance of efficiency

Any solution that addresses the ranking problem, including LtR, by definition seeks to maximize a user satisfaction metric, Q. But in many real-world applications achieving the highest **effectiveness** is only one of many requirements. We may indeed desire to impose additional constraints on the ranked list, such as a requirement that ranked lists fairly represent underrepresented categories; that they guarantee privacy when the set D consists of documents private to a user; or that they counter biases and ensure trust. Each of these additional constraints is an important objective to optimize in its own right.

An objective that is equally as important as effectiveness in many applications is the **efficiency** of the retrieval and ranking systems. For example, it is often imperative to find the right documents and finalize a ranked list within a small time budget to meet demand and ensure a timely delivery of information. In fact, a perfectly-ordered ranked list may be

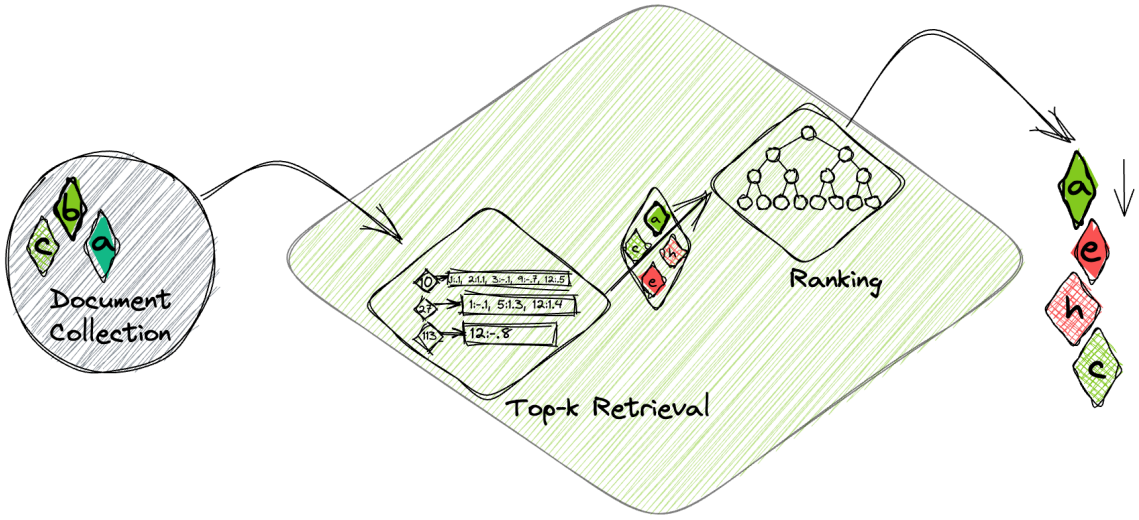


Figure 1.2: Retrieval and ranking algorithms in a modern search system. The *retrieval* algorithm often solves one form of the maximum inner product search (MIPS) problem using, for example, an approximate nearest neighbor (ANN) search or an inverted index-based top-k retrieval algorithm where closeness is determined by lexical matching scores. The *ranking* algorithm may be as simple as an identity function (e.g., in deep learning-based "dense retrieval") or a complex learnt function such as decision forests or deep learning models.

of little value or have a low perceived quality if delivered too late or with substantial delay.[1]

The question of efficiency gained increasing significance with the rise of LtR whose training and serving require large amounts of computational power. Indeed, the success of LambdaMART and subsequent decision forest-based descendents (Ganjisaffar *et al.*, 2011; Dato *et al.*, 2016; Bruch, 2021; Lucchese *et al.*, 2018b) in improving the quality of rankings came at the expense of the efficiency of training and inference. The training of such models is expensive because we must often (and repeatedly) learn ensembles of hundreds to thousands of deep decision trees sequentially with gradient boosting (Friedman, 2001), with each node in every tree requiring a search in the feature space (Breiman *et al.*, 1984). To become accurate, these large models need to be trained on vast amounts of data, often represented as complex features that are in turn costly to compute. Inference, too, is computationally intensive because estimating the relevance of a single document to a query requires the traversal of paths, from roots to leaves, of every decision tree in the model.

[1] Kohavi *et al.* (2013), reporting on an experiment conducted at Bing, a web search engine, estimated that *"every 100msec improves revenue by 0.6%."*

1.2 Efficiency considerations beyond latency

A decade later, deep neural networks, and in particular, Transformer-based (Vaswani *et al.*, 2017) pre-trained language models advanced the state-of-the-art in ranking dramatically (Lin *et al.*, 2021; Nogueira and Cho, 2020; Nogueira *et al.*, 2019a; Nogueira *et al.*, 2020). Learnt representations of queries and documents by deep networks, too, offer a range of opportunities including the development of a new generation of "dense" retrieval methods (Karpukhin *et al.*, 2020; Xiong *et al.*, 2021), document expansion techniques (Nogueira *et al.*, 2019b), and others. These recent developments mark the beginning of a new era known as Neural Information Retrieval (NIR).

NIR is a leap forward, reaching new highs in quality. Whatever the reason behind its success may be, NIR achieves a greater effectiveness than the previous wave of machine learning models like decision forests on many information retrieval tasks, but with orders of magnitude more learnable parameters and much greater amounts of data. The new scale drastically increases the computational and economic costs of model training and inference. GPT-3 (Brown *et al.*, 2020), for example, required 285,000 CPU cores and 10,000 GPUs to train, with an estimated economic cost of $4.6M.[2] Although it may be argued that the high cost of training deep models is amortized because large language models can, through a process known as "fine-tuning," be recycled and reused for a variety of applications with a substantially smaller effort, it is still a significant price to pay upfront. Furthermore, not all large neural models can be easily recycled—in fact, that is one of the properties Scells *et al.* (2022) call out in their article. What is more, once trained, the use of such large models in production similarly requires a nontrivial amount of tensor multiplications and other complex operations.

Due to their alarming computational requirements, NIR models underline several dimensions of efficiency that have thus far been less obvious. Crucially, "efficiency" is no longer characterized by low latency, but is instead a concept that amalgamates space-, sample-, and energy-efficiency, among other emerging factors, as summarized in Table 1.1.

In other words, the inefficiency of an algorithm cannot and should not be understood solely in terms of negative user experience due to greater latencies, but instead, we must acknowledge that ineffciency has adverse implications for resource-constrained researchers and practitioners, and more importantly, for the environment (in the form of emissions and carbon footprint) (Scells *et al.*, 2022; Strubell *et al.*, 2019; Xu *et al.*, 2021). We must therefore acknowledge that, due to environmental factors, attempting to address the efficiency problem by relying on advances in hardware systems or by utilizing more resources is not a sustainable long-term solution. Instead, combating this multi-faceted issue of efficiency necessitates a careful study and design of efficient algorithms and data

[2] https://lambdalabs.com/blog/demystifying-gpt-3/

Table 1.1: Taxonomy of a multi-faceted view of ranking efficiency and the stages in which they manifest.

Dimension	Definition	Scope
Query	Time elapsed between the arrival of a query and the presentation of ranked list of documents	Inference
Sample	Number of training examples required to learn a ranking function	Training
Space	Total storage used to serve a ranking model	Training; Inference
Training	Time required to train a ranking model	Training
Energy	Amount of energy required to train a model or evaluate a learnt model on a query-document pair	Training; Inference

structures, as highlighted by deliberations at recent academic workshops (e.g., the Workshop on Reaching Efficiency in Neural Information Retrieval (Bruch *et al.*, 2022b; Bruch *et al.*, 2023)).

1.3 Efficient and effective ranking

Accuracy by way of ever-increasing complexity presents a challenge: how do we then optimize for both effectiveness and efficiency? Must we lose accuracy to find a more efficient solution, inevitably trading off effectiveness for efficiency and vice versa? These and other similar questions give rise to a research topic that extends the document ranking problem as follows:

Definition 1.3.1 (The Efficient Document Ranking Problem)**.** Given a query q and a set of documents D, the goal is to order elements of D *efficiently* such that the resulting ranked list maximizes a user satisfaction metric Q.

The problem above spawned a line of research in the information retrieval community to systematically investigate questions of efficiency and explore the trade-offs between efficiency and effectiveness in ranking models, leading to several innovations. The community widely adopted multi-stage, *cascade* rankers, separating light-weight ranking on large sets of documents from costly re-ranking of top candidates to speed up inference at the expense of quality (Wang *et al.*, 2011; Asadi and Lin, 2013a; Dang *et al.*, 2013; Culpepper *et al.*, 2016; Mackenzie *et al.*, 2018; Liu *et al.*, 2017; Asadi, 2013). From probabilistic data structures (Asadi and Lin, 2012; Asadi and Lin, 2013b), to cost-aware training and *post hoc* pruning of decision forests (Asadi and Lin, 2013c; Lucchese *et al.*, 2017b; Lucchese *et al.*,

2016a; Dato *et al.*, 2016), to early-exit strategies and fast inference algorithms (Cambazoglu *et al.*, 2010; Asadi *et al.*, 2014; Lucchese *et al.*, 2016b; Lucchese *et al.*, 2015b), the information retrieval community thoroughly considered the practicality and scalability of complex ranking algorithms.

In addition to volumes of publications, the output of this research effort included standardized algorithms and reusable software packages (Ke *et al.*, 2017; Lucchese *et al.*, 2015b). Perhaps more crucially, the community developed an understanding that quality is not the be-all and end-all of information retrieval research and that model complexity must be managed (through more efficient training and inference) and justified (e.g., by contextualizing quality gains in terms of the amount of computational resources required).

As complex neural network-based models come to dominate the research on document ranking, it is unsurprising that there is renewed interest in the question above, not just in the information retrieval community but also in related branches such as natural language processing. Interestingly, many of the proposals put forward to date to contain efficiency are reincarnations of past ideas, such as stage-wise ranking with BERT-based models (Nogueira *et al.*, 2019a; Matsubara *et al.*, 2020), early-exit strategies in Transformers (Soldaini and Moschitti, 2020; Xin *et al.*, 2020; Xin *et al.*, 2021), neural connection pruning (Gordon *et al.*, 2020; McCarley *et al.*, 2021; Lin *et al.*, 2020b; Liu *et al.*, 2021), precomputation of representations (MacAvaney *et al.*, 2020b), and enhancing indexes (Zhuang and Zuccon, 2022; Nogueira *et al.*, 2019b; Mallia *et al.*, 2022; Lassance and Clinchant, 2022). Other novel but general ideas such as knowledge distillation (Jiao *et al.*, 2020; Sanh *et al.*, 2020; Gao *et al.*, 2020) have also proved effective in reducing the size of deep models. Yet other innovative ideas developed specifically for ranking include efforts to reinvent Transformers from the ground-up (Mitra *et al.*, 2021; Hofstätter *et al.*, 2020).

1.4 About this monograph

Given the resurgence of the question of efficiency and the trade-offs between efficiency and effectiveness in ranking, and the apparent overlap between the neural and pre-neural ideas to address this question, we believe it is necessary to present a comprehensive review of this literature with a particular focus on the document ranking problem. We have thus prepared this monograph in four parts in the hope that it serves as one such resource.

The first part introduces the document ranking problem and reviews a machine learning formulation of it in the context of web search in depth. We also describe the architecture of a modern search engine to illustrate an application of ranking that is of primary interest to this work. As we explain the ingredients of a search engine and all that is involved in the training and serving of a ranking model within this framework, we highlight the costs to efficiency and call out the levers that trade off effectiveness for efficiency.

While the first part of this monograph concerns an abstract, general setup, the two subsequent parts get more specific and examine two popular families of ranking algorithms through the lens of efficiency. One presents a treatment of a branch of LtR that is based on forests of decision trees, while another turns to neural networks and deep learning methods for retrieval and ranking. Each family presents its own unique challenges and requires its own set of solutions to explore the Pareto front on the efficiency-effectiveness optimization landscape.

As the reader will notice, the approaches developed for the two families of ranking algorithms appear to be—and in many ways, are—independent. But the ideas behind them overlap too. We attempt, in the last part of the monograph, to identify the common threads that can help translate ideas from one space to another. We also discuss emerging research directions, made urgent by the rise of deep neural networks in information retrieval, and explore open challenges within this space.

2 Learning to Rank: A Machine Learning Formulation of Ranking

A ranking algorithm, at its core, is a function of a set of documents and a query. It is no surprise then that its simplicity or complexity is determined by how we represent documents and queries.

Let us, for example, strip away grammar and sentence structure from a text query or document. That leaves us with a bag-of-words perspective of the text: a query or document is simply a multiset of terms from a fixed vocabulary. Modeled this way, we can represent documents and queries naïvely as vectors in a space that has as many dimensions as there are terms in our vocabulary, and where each dimension records the frequency of the corresponding term in that document or query. Perhaps we would further weight each dimension to reflect its "importance" (Sparck Jones, 1972)—an article like "the" that occurs frequently in a large subset of documents but that carries little information would have a lower weight.

In the vector space construction above we can measure the relevance of a document to a query using a similarity measure between query and document vectors (Salton and Buckley, 1988), such as cosine similarity or inner product, as illustrated in Figure 2.1.[1] Alternatively, because each vector is a distribution over a vocabulary, we may use a probabilistic approach based on language models (Ponte and Croft, 1998) to measure query-document similarity

[1] If this reminds the reader who is familiar with neural information retrieval of neural rankers, it is because in both models queries and documents are represented as vectors. While similar in principle, in the bag-of-words model, these vectors are sparse vectors of basic statistics such as (weighted) term frequencies, while a neural ranking function *learns* a dense or sparse vector representation of its input from the raw data.

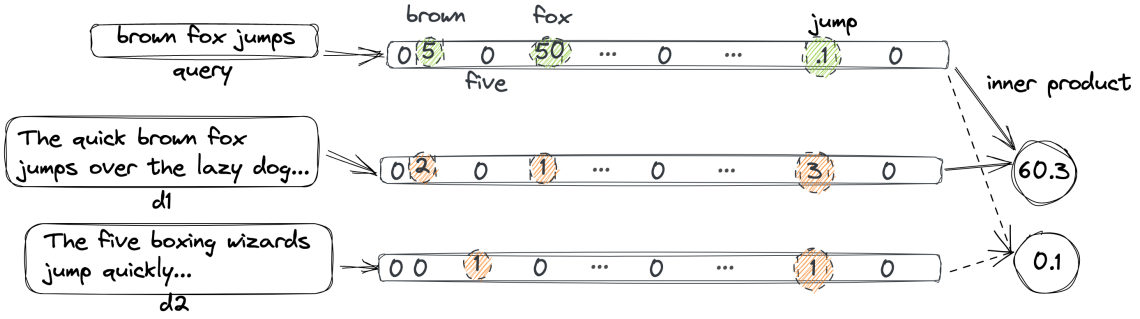


Figure 2.1: Vector representation of query and documents in the bag-of-words model. The relevance of a document to a query may be estimated using the inner product of their vectors.

(e.g., the likelihood of observing a query given a document). In either case, the ranking function is simple: it computes the similarity scores for pairs of query and document vectors and sorts the documents in decreasing order of similarity.

Term frequencies in a bag-of-words model only carry so much information, and there is indeed far richer signals available in queries and documents to aid ranking. Term co-occurrences and increasingly more advanced semantic features provide greater insight than individual words, for example. Another source of useful features, especially in the context of web search, is the structure within documents, where "fields" such as titles and sections carry different weights (Jones *et al.*, 2000; Robertson *et al.*, 2004). Continuing with the web search example, there are numerous other indicators of relevance in the web graph (such as the anchor text of incoming links, in-degree and out-degree of a web page, number of references in social network streams) and in the user interaction with the search system (e.g., clicks, user sessions, effect of query reformulation).

The list of statistics used in modern search engines is indeed long, encompassing many facets of documents and queries beyond term frequencies. Benchmark ranking datasets, for example, represent queries and documents with hundreds or thousands of statistics. As examples, there are 136 features for a query-document pair in the MSLR datasets,[2] and 700 in the Yahoo! Learning to Rank Challenge datasets.[3]

As the size and complexity of the query and document representations grow, it becomes impractical to hand-craft a ranking function and ineffective to use basic similarity measures. It is rather more practical and effective to view the ranking problem as a supervised learning task—a task known as LtR, visualized in Figure 2.2. We should highlight that, other texts often use LtR to refer specifically to the first wave of machine learning-based methods that came before deep learning. In this monograph, we view neural networks and deep learning

[2] Available at https://www.microsoft.com/en-us/research/project/mslr/.
[3] Available at https://webscope.sandbox.yahoo.com/catalog.php.

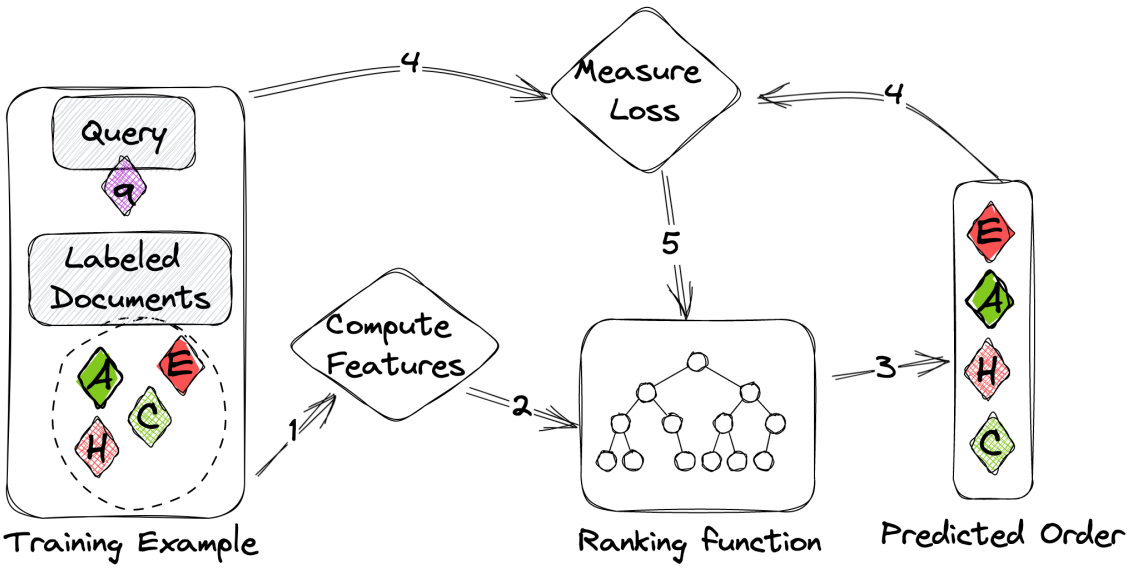


Figure 2.2: Visualization of a supervised framework for learning a ranking function. A training example in ranking consists, at least, of a query and a set of (partially) labeled documents. Each example is represented by a vector of features, either learnt using a deep learning model or engineered based on statistics. A ranking function is then trained to produce ranked lists so as to minimize the difference between the produced ranked list and the ideal one.

as just another hypothesis class in the LtR framework and use LtR to discuss both pre- and post-neural research in ranking in a unified framework.

Equipped with a rich representation of queries and documents as vectors of feature values, and a set of labels indicating the relevance of a document to a query, we have one piece of the puzzle to learn a ranking function. To complete the picture, we also need a method of evaluation to determine the success of the learnt ranking function, and an optimization objective to enable learning. In this chapter, we unpack each of these components in the context of ranking.

2.1 Ranking datasets

To understand the structure of a ranking dataset, we find it helpful to first revisit other tasks in the family of supervised learning algorithms which have a simpler construction. Take regression and classification as its well-known members. A typical regression or classification dataset is made up of a number of examples that, respectively, come with a value and a class label. An example could be an image to be classified as a cat or a dog, or a piece of text to be decided as relaying a positive or negative sentiment. Each example is assumed to have been sampled independently of others, but from the same underlying data distribution. Examples are either given as vectors of feature values—hence, tabular or structured—or represented in their raw, unstructured form such as text, or a mix of both.

Ranking adds a new dimension to the construction above, with an example now comprising of two parts: A query and a *set* of documents.[4] The objective of a ranking algorithm is to sort the set of documents with respect to the query in some order. In web search, that order is often determined by relevance, but other factors such as diversity, fairness, freshness, personalization, or a combination of those may play a role.

The additional dependence of a ranking example on query and document affects how we represent and label a ranking dataset. The next two sections explain its implications.

2.1.1 Representation

How a query and documents are represented is largely similar to the regression or classification setting, with tabular features summarizing examples or unstructured data from which latent features may be learnt. The one difference in ranking, however, is that in addition to extracting features from the query, and another set from individual documents, we must also obtain signals that jointly describe a query-document pair.

[4]Note that, while "query" and "document" may suggest textual content, these notions, in fact, extend to a variety of multi-modal domains such as a recommendation engine where a query may be a user "profile" and documents are movies.

Consider the query alone. We may classify the query and record its type (e.g., news vs. health) or categorize a user session (Bennett *et al.*, 2010; Jiang *et al.*, 2016; Lucchese *et al.*, 2013). These signals allow the ranking model to better determine the user intent and match the query to a document from the appropriate class, and to adaptively use the available features (e.g., recency should be considered as a more discriminative signal if we are searching for news articles rather than historical facts).

Now consider the document in isolation. There are important signals in the structure of a web document, such as the number of incoming links, or the PageRank value, its quality or spamminess, or other more advanced link analysis features (Henzinger *et al.*, 2000). The MSLR dataset, for example, includes some rather surprising document features such as the number of slashes in a document's URL, click count, and dwell time, capturing aspects of user interaction and experience!

Now consider the numerous signals that may describe or capture the relevance of a document to a query. The MSLR dataset, for example, uses a vector of 136 real-valued features for each query-document pair. These features include signals from a vector space model (such as the weighted sum of term frequencies of query terms in that document (Salton and Buckley, 1988)), as well as the expected relevance according to different language models (Ponte and Croft, 1998). Another class of features that can be computed jointly for a query-document pair is term proximity (Rasolofo and Savoy, 2003): the terms of a query with multiple terms are likely to appear within a short span in relevant documents. It is common to compute such content-based features not just from the body of a document, but also from its other parts such as its URL, title, and anchors, resulting in a fine-grained feature set.

Table 2.1 summarizes some of the most common features used to represent a query-document pair, and categorize by whether they model a document, a query, a user or their combination. We refer the interested reader to an analysis of these features by Macdonald *et al.* (2012).

2.1.2 Relevance labels

Viewed as a simple yes or no question—is this and only this ranked list correct?—the ranking problem reduces to a classification one, where a single binary label suffices for each example. But the ranking question is seldom this coarse and it is unlikely there is just a single correct ranking. Instead, for a single query, many different ranked lists may equally make sense. For instance, it hardly matters how documents that are irrelevant to a query are ordered relative to each other, so long as they appear below the more relevant ones.

In fact, in most applications of ranking, labels are defined at the granularity of query-document pairs: Is this document relevant to that query? If so, what is its degree of relevance? In most practical settings where document collections are vast and query possibilities endless,

Table 2.1: Typical features to represent a query-document pair.

Category	Description	Feature
Document	Properties of body and URL (MSLR datasets)	Number of slashes in URL; length of URL; page length
Document	Properties of the page as a node in the Web graph (MSLR datasets)	Inlink count; outlink count; PageRank
Document	Document quality	Spam score (Cormack *et al.*, 2011); fraction of stopwords; fraction of visible terms (Bendersky *et al.*, 2011)
Query	Query classification (Bennett *et al.*, 2010; Jiang *et al.*, 2016)	Query topic; query intent
Query-Document	Properties of query-document match (MSLR datasets)	Boolean model; TF-IDF; BM25; vector space model; language model
Query-Document	Proximity-sensitive matching (Rasolofo and Savoy, 2003)	Term proximity
User-Document	Properties of user interaction with the page (MSLR datasets)	URL click count; dwell time
User-Query-Document	Properties of the user interaction with the page in response to a query (MSLR datasets)	Query-URL click count

labels may only be defined for a fraction of documents. Documents that are examined and subsequently labeled too are typically not drawn uniformly randomly from the collection. In these ways, labels in a ranking dataset are different from those in classification datasets.

We distinguish between two different methods for labeling document with relevance labels. The first approach is to manually assign labels. It is well known that major web search engines have been using thousands of quality raters for this purpose (The Guardian, 2017). And in this case very accurate guidelines for raters exists (Gomes, 2017) so that the labeling process is accurate, consistent, and reliable. For instances, it might be preferable to collect preference judgments among document pairs (Carterette *et al.*, 2008) (i.e., is document A more relevant to the query q than document B?).

Manually labeling collections of query-document pairs in a naïve manner clearly does not scale in terms of space (number of documents) or time (new queries and new documents come in every day) and can therefore be cost-prohibitive. To scale this effort, researchers have developed other methods of collecting feedback such as by using active learning to minimize the number of annotations but still maximize the quality of the training data (Long *et al.*, 2010). Interestingly, Yilmaz and Robertson (2009) also show that, for the purposes of training a ranking model, it is more advantageous to collect a larger dataset of *shallow* judgements (i.e., fewer annotated documents per query) as opposed to a smaller dataset of *deep* judgments (i.e., with a large number of annotations per query).

An alternative to (targeted) manual labeling, one can look to a different source of information to deduce relevance labels. One important source is the *implicit* feedback generated by the users of a ranking system. Consider, for instance, the number of clicks received by a document in response to a query, or even the absence of any click, or the editing and reformulation of a query by a user after an unsatisfactory search results list, among other signals (Joachims *et al.*, 2005; Joachims, 2002; Radlinski and Joachims, 2005). All of those actions can be mined for and translated into relevance labels! Of course, this data is subject to different biases: users are more likely to click on the top document in the result list; users are unlikely to traverse multiple result pages; and the system may not be able to return the most relevant result for a query anyway. But this noisiness and these biases can be modeled using "click" models (Chuklin *et al.*, 2015) and treated counterfactually (Oosterhuis *et al.*, 2020; Joachims *et al.*, 2017) to enable unbiased learning and evaluation of ranking models. We refer the reader to the vast literature on these topics for details.

2.1.3 Notation

Let us introduce some notation to summarize the discussion in this section and formally define a typical LtR dataset. Such a dataset $\mathcal{D}$ comprises of a set of triplets $(q, \boldsymbol{x}, \boldsymbol{y})$. The

vector[5] $\boldsymbol{x}$ includes the documents that we want to rank in response to the query q. For a given query-document pair (q, x_i), where x_i is a member of $\boldsymbol{x}$, the true relevance of x_i with respect to q is encoded by y_i in the vector $\boldsymbol{y}$ of relevance labels.

Each (q, x_i) is represented in some feature space $\mathcal{X}$ that captures properties of the query (e.g., its likelihood, category), of the document (e.g., its incoming links), and of their relationship (e.g., the number of occurrences of the query in the document). We denote by $\mathcal{Y}$ the set of possible labels. This is typically *binary* indicating relevant vs. non-relevant, or *graded* where $\mathcal{Y}$ is restricted to a small set of integers such as $\mathcal{Y} = [4] \triangleq \{0, 1, 2, 3, 4\}$, encoding different degrees of relevance with larger grades corresponding with stronger relevance.

A *ranker* R is a function that, given a pair $(q, \boldsymbol{x}) \in \mathcal{X}^n$, produces a permutation vector $\boldsymbol{\pi} \in \mathbb{Z}^n$, $\pi_r \in [n]$, of the n items in $\boldsymbol{x}$. Such a permutation defines the ranking produced by R as follows: the item ranked at position r is the π_r-th item in $\boldsymbol{x}$: x_{π_r}. An *ideal* ranking $\boldsymbol{\pi}^*$ is one that sorts documents in decreasing order of their relevance labels, where $y_{\pi^*_r} \geq y_{\pi^*_{r+1}}$. Note that there might be multiple ideal rankings.

2.2 Ranking metrics

Earlier in our discussion, we touched on ways in which labels in ranking are different from those in classification. It is not surprising then that evaluation metrics that help us assess the quality of a ranked list too are different from their classification or regression counterparts.

Ranking metrics attempt to measure the utility of a ranked list to a user. It is therefore helpful to consider the important factors in the way users interact with a ranked list. First, users expect the relative ordering of documents to be correct. That is, documents that are placed higher in the ranked list (i.e., towards the top of the list) should satisfy the information needs of a user better than documents lower on the list, and that as the user goes down the list, documents become less relevant to the query. Second, user attention has a skewed distribution with much of it focused on the top of the ranked list. In other words, users typically do not examine all documents at every position with equal probability or care. As such, higher positions carry more weight.

The information retrieval literature offers a great number of metrics that are designed specifically on the basis of the factors above. Most of these have the following additive form:

$$Q@k(\boldsymbol{\pi}, (q, \boldsymbol{x}, \boldsymbol{y})) = \frac{1}{Z} \sum_{1 \leq r \leq k} \textsc{Gain}(r) \cdot \textsc{Discount}(r). \tag{2.1}$$

At a high level, the additive nature of the formulation above reflects the view that each document contributes to the overall quality Q of a ranking $\boldsymbol{\pi}$ independently of others.

[5] Throughout this monograph, we denote vectors as lowercase letters in bold.

Typically, we only consider the top k high-ranking documents when computing Q and denote it by $Q@k$, reflecting the assumption that user attention dissipates past position k. Within this framework, a document at rank position r provides a contribution of $\textsc{Gain}(r)$ to the metric, which is typically a function of its label y_{π_r}. However, this contribution wanes as r grows, by a factor of $\textsc{Discount}(r)$, a decreasing function of r. Note that, both $\textsc{Gain}(\cdot)$ and $\textsc{Discount}(\cdot)$ are typically formulated based on a model of user behavior or "click" models (Chuklin *et al.*, 2015). The constant Z is often used as a normalization factor to ensure that the metric Q lies within the unit interval. Finally, given a test ranking dataset of examples $(q, \boldsymbol{x}, \boldsymbol{y})$ and their corresponding ranked lists $\boldsymbol{\pi}_q$, we compute $Q@k$ for each example and report its mean as the average quality.

Specific instances of Equation (2.1) differ in how they define $\textsc{Gain}$ and $\textsc{Discount}$. We review a few metrics that are commonly used in the ranking literature in this section, but encourage the reader to refer to (Liu, 2009) for a more thorough treatment.

As an example, consider Rank-Biased Precision (RBP) by Moffat and Zobel (2008). The authors define $\textsc{Gain}(r) = y_{\pi_r}$, but to formulate $\textsc{Discount}$, they make the assumption that, at any given point, a user inspects the next document on the list with probability p and abandons the ranked list altogether with probability $1 - p$. On that basis, they take the probability that a user reaches rank r as the discount factor: $\textsc{Discount}(r) = p^{r-1}$. The normalization constant Z is $1 - p$, the inverse of the average number of documents that a user inspects.

As another example for graded relevance, consider one of the most popular metrics known as Normalized Discounted Cumulative Gain (NDCG) (Järvelin and Kekäläinen, 2000). The gain is computed as $\textsc{Gain}(r) = 2^{y_{\pi_r}} - 1$, leading to a dynamic where a document with label 4 is about twice as important as a document with label 3. The discount is computed as $\textsc{Discount}(r) = \frac{1}{\log_2(r+1)}$. When $Z = 1$, the resulting metric is called Discounted Cumulative Gain (DCG). To compute NDCG, however, we normalize by the *ideal* DCG by setting $Z = \text{DCG@}K(\boldsymbol{\pi}^*, (q, \boldsymbol{x}, \boldsymbol{y}))$.

Both RBP and NDCG assume that the user examines the next document on the ranked list independently of the relevance of the documents observed along the way. In a more practical click model, however, users are likely to stop their inspection of the remainder of a ranked list soon after they find the document that satisfies their information need. With the goal of capturing such a behavior, Chapelle *et al.* (2009) propose the Expected Reciprocal Rank (ERR). The gain there is defined as $\textsc{Gain}(r) = p_r \prod_{i=1}^{r-1}(1 - p_i)$, where p_i is the probability that the user is satisfied with the document at rank i and stops inspecting the rest of the list. This probability is generally a function of relevance, for example, $p_i = \frac{2^{y_{\pi_i}} - 1}{2^{\max \mathcal{Y}}}$. The gain of the document at rank r is thus the probability that the user finds the first satisfactory document at that position, having judged as non-relevant all the preceding documents. The discounting mechanism is simply $\textsc{Discount}(r) = \frac{1}{r}$, and $Z = 1$ as no normalization is necessary. Experiments show that ERR correlates better with

Table 2.2: Common evaluation metrics for ranked lists

Relevance	Name	Metric as a function of $\boldsymbol{\pi}$ and $(q, \boldsymbol{x}, \boldsymbol{y})$
Binary	MAP	$\frac{1}{\sum_r y_{\pi_r}} \sum_{\substack{1 \le r \le n, \\ y_{\pi_r}=1}} \text{P@}r$, $\text{P@}r = \frac{1}{r} \sum_{1 \le i \le r} y_{\pi_i}$
Graded	RBP	$(1-p) \cdot \sum y_{\pi_r} \cdot p^{r-1}$
	DCG@k	$\sum_{1 \le r \le k} \frac{2^{y_{\pi_r}} - 1}{\log_2(r+1)}$
	NDCG@k	$\text{DCG@}k(\boldsymbol{\pi}, (q, \boldsymbol{x}, \boldsymbol{y}))/\text{DCG@}k(\boldsymbol{\pi}^*, (q, \boldsymbol{x}, \boldsymbol{y}))$
	ERR@k	$\sum_{1 \le r \le k} p_r \prod_{i=1}^{r-1} (1 - p_i) \cdot \frac{1}{r}$, $p_i = \frac{2^{y_{\pi_i}} - 1}{2^{\max \mathcal{Y}}}$

user satisfaction (Chapelle *et al.*, 2009).

The metrics we have reviewed thus far are based on Equation (2.1), but not all ranking metrics belong to this family. One such example is Mean Average Precision (MAP) (Buckley and Voorhees, 2005) for binary relevance. Let us parse this metric one term at a time. Let Precision at k, denoted by P@K, be the fraction of relevant documents among the top k documents. Then define Average Precision (AP) as follows:

$$\text{AP}(\boldsymbol{\pi}, (q, \boldsymbol{x}, \boldsymbol{y})) = \frac{1}{\sum_r y_{\pi_r}} \sum_{\substack{1 \le r \le n, \\ y_{\pi_r}=1}} \text{P@}r.$$

MAP is the mean of this value computed over all queries in a dataset.

Table 2.2 summarizes the metrics we have reviewed. We conclude by highlighting that measures such as NDCG and ERR are very difficult to optimize and that changes that may appear small have a significant impact in practice. According to (Chapelle *et al.*, 2012), the differences between major revisions of Bing, "involve changes of over half a percentage point, in absolute terms, of MAP and NDCG."

2.3 Learning objectives

We have just seen what factors are good indicators of the quality of a ranked list and how ranking metrics evolved to take those factors into consideration. In this section, we review how we learn a ranker that produces high-quality ranked lists.

While we defined a ranker R to be a function that permutes documents $\boldsymbol{x}$ in response to a query q, in practice, R instead computes a *relevance score* for every query-document pair (q, x_i) and subsequently sorts x_i's in decreasing order of relevance to produce a permutation.

This two-step trick greatly simplifies the learning of a ranker R, which is also known as a *scoring* function.

How do we learn such a scoring function given a labeled training dataset? At a high level, it is natural to take a ranking metric Q and learn an R that maximizes it, with the intuition that a ranking function trained to maximize Q should produce high-quality ranked lists as measured by Q.

While the instinct to use a ranking metric as the learning objective may be natural, whether that is sensible depends on the optimization method itself. Consider, for example, gradient-based optimizers that are commonplace in machine learning. For an objective to be optimized by such an optimizer, it must have meaningful gradients. A ranking metric, being a function of discrete ranks, does not offer gradients that are all that interesting: small perturbations of relevance scores computed by a ranking function often do not lead to a change in ranks, and as such, the gradients of a ranking metric with respect to relevance scores are typically either zero or nonexistent due to discontinuities.

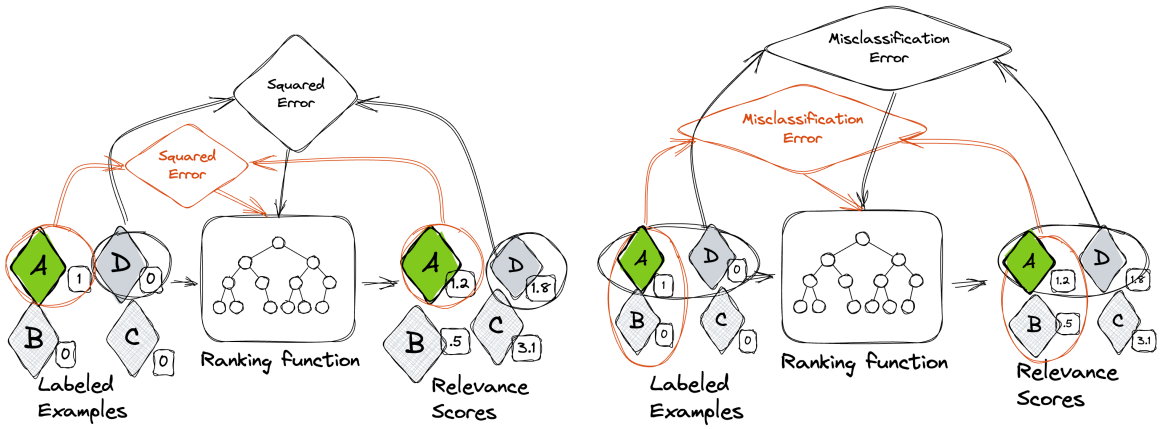
The popularity and effectiveness of gradient-based optimizers and their unfortunate incompatibility with ranking metrics bring us to an important research topic that offers a way to reconcile the two: *surrogate* objectives. The idea is to devise or derive from ranking metrics an objective that is differentiable and consistent. It must be differentiable so that its gradients can inform an optimizer of the correct direction to follow. It must be consistent with a ranking metric so that, in expectation, optimizing it leads to an optimal metric as well.

The LtR literature has long sought and studied surrogate objectives that are differentiable and, while not necessarily consistent, exhibit a behavior that is intuitively in keeping with ranking metrics. To help explain the differences between existing surrogate ranking objectives, let us place them into one of three buckets based on their behavior: pointwise, pairwise, and listwise.

The intuition behind pointwise methods is to reduce the ranking problem to one of regression, multi-class classification, or ordinal regression (Liu, 2009). In regression, for example, we may optimize the squared difference between the true relevance label and predicted relevance score of a query-document pair in expectation, known as the mean squared error, as illustrated in Figure 2.3(a). When cast this way, as noted earlier, the question becomes one of predicting the degree of relevance of each document with respect to a given query independently of others. Furthermore, this framing of the ranking problem implicitly requires an absolutist view of relevance: a document is either relevant or it is not.

We hinted in our earlier discussion that such a view is hardly appropriate in general. For the vast majority of applications, a stronger view is to consider relevance as a relative concept: a document is *more* (or *less*) relevant than another. The next wave of surrogate ranking objectives reflect this paradigm shift.

Pairwise methods are closer to the relative definition of relevance in that they model



(a) Pointwise **(b)** Pairwise

Figure 2.3: Illustration of the machinery of pointwise and pairwise ranking objectives for four documents ($\{A, B, C, D\}$) in the context of a single query, with numbers attached to each document indicating a relevance label or predicted score. In pointwise methods, (a), the predicted relevance score of every document is compared to its label. In pairwise methods, (b), the function is evaluated in terms of its accuracy in predicting the correct order among pairs of documents.

error as a function of not a single, isolated document, but of pairs of documents: When sorted by their relevance scores, does the resulting order between any pair of documents correctly reflect our preference between them? This is illustrated in Figure 2.3(b). Presented this way, the question becomes one of preference learning via binary classification and, as such, any classification objective serves as a suitable surrogate. RankNet (Burges *et al.*, 2005), Ranking-SVM (Joachims, 2002), and RankBoost (Freund *et al.*, 2003) offer examples of this approach.

To make the idea more concrete, consider RankNet, whose surrogate objective was argued to correlate with NDCG (Cao *et al.*, 2007). Given two documents x_i and x_j, it maps the difference between their relevance scores (o_{ij}) to a probability using the logistic function: $P_{ij} = 1/(1 + e^{-o_{ij}})$. This probability can be understood as the strength of the predicted order between the pair. When we have computed these probabilities for every pair, it is simply a matter of optimizing its cross entropy (C_{ij}) with the ground truth $\overline{P}_{ij}$, which is 1 if x_i is more relevant than x_j and 0 otherwise: $C_{ij} = -\overline{P}_{ij} \log(P_{ij}) - (1 - \overline{P}_{ij}) \log(1 - P_{ij})$.

In repeated experiments, pairwise methods have proven successful, particularly when compared with their pointwise counterparts. The empirical success that ensued the shift above motivated the research community to extend the idea of preference learning from a pair of documents to an entire list of documents. In other words, similar to how ranking metrics quantify the quality of an entire ranked list, we seek to quantify the ranking error in

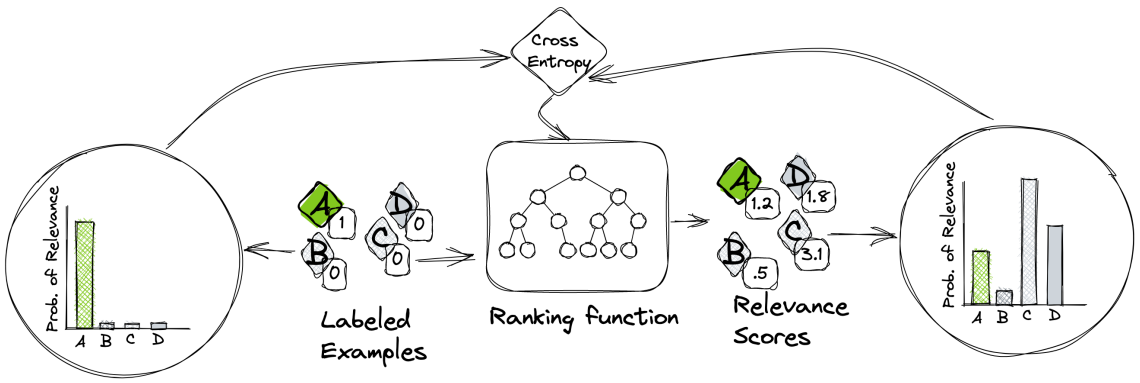


Figure 2.4: Illustration of the ListNet (Cao *et al.*, 2007) objective for a single query with four documents, with numbers attached to each document indicating a relevance label or predicted score. The labels and scores are separately projected onto the probability simplex to form a distribution over documents. Subsequently, the cross entropy between the two distributions is taken as a measure of how far the ranked lists are from each other.

terms of the induced order among a list of documents, not just between pairs. The surrogate objectives that have emerged from this research effort are known collectively as the class of listwise methods.

The LtR literature contains a great number of listwise methods. ListNet (Cao *et al.*, 2007) and ListMLE (Xia *et al.*, 2008) take a probabilistic approach by applying the Plackett-Luce model to estimate the probability of permutations. This is illustrated in Figure 2.4. More interestingly, others like ApproxNDCG (Qin *et al.*, 2010; Bruch *et al.*, 2019b) or SoftRank (Taylor *et al.*, 2008) derive smooth approximations to ranking metrics. LambdaRank and LambdaMART (Burges, 2010) extend the RankNet objective using a heuristic where the contribution to the error from a pair of documents is shrunk or amplified by a multiplicative factor that correlates with the amount of change in NDCG (or any other metric) if the two documents traded ranks. A more recent work (Bruch *et al.*, 2019a; Bruch, 2021) modifies the ListNet objective to improve its consistency. Oosterhuis (2021) proposed an approach to directly optimize the Plackett-Luce model for ranking.

The listwise methods cited above are but a few representatives of a large class of algorithms in the machine learning and information retrieval literature (Swezey *et al.*, 2021; Xie *et al.*, 2020; Cuturi *et al.*, 2019; Blondel *et al.*, 2020; Jagerman *et al.*, 2022). We return to these methods in later chapters and study some of them in greater detail. But first, to complete the supervised learning formulation, we must discuss hypothesis classes.

2.4 Hypothesis classes

Equipped with a training dataset, a ranking objective, and an optimizer, we are ready to learn a ranker R. There is, however, one final piece of the supervised learning puzzle: What relationship do we hypothesize exists between a relevance score and the features that represent a query-document pair? In other words, what family of functions do we think R belongs to?

In its simplest form, R may be a linear function, parameterized by a set of coefficients and a bias term that can be learnt from data so as to optimize our objective. It is clear that such a function, on its own, does not model any nonlinear relationship that may exist in the data, and, as such, is generally less effective than other, more complex families of functions. However, its simplicity facilitates formal and rigorous analysis and allows us to provide certain guarantees on performance. That is why linear rankers are favored in the vast literature on online LtR with bandit algorithms (Radlinski *et al.*, 2008; Yue and Joachims, 2009; Yue *et al.*, 2012; Hofmann *et al.*, 2013a; Hofmann *et al.*, 2013b; Kveton *et al.*, 2015).

Another class of algorithms take R to be a *decision forest*: an ensemble of decision trees, typically with real-valued leaves. At a high level, a decision tree is a piecewise constant function that is learnt by recursively partitioning the feature space into disjoint spaces and assigning a value to each partition. Learning many of these decision trees and putting them together into a forest in an additive manner yields highly complex functions, capable especially of modeling *tabular* features. Indeed, past studies have shown decision forest-based rankers to be highly effective (Ganjisaffar *et al.*, 2011; Szummer and Yilmaz, 2011; Bruch, 2021), among them, LambdaMART (Burges, 2010) remains the state of the art.

In recent years, the success of neural networks and deep learning in related areas of research has led to a rise in deep learning methods for ranking, where R is taken to be an often complex neural network with specialized modules for processing textual data. The effectiveness of pioneering methods such as ConvDNN (Severyn and Moschitti, 2015), DSSM (Huang *et al.*, 2013), and others (Mitra *et al.*, 2016; Mitra *et al.*, 2017; Dehghani *et al.*, 2017; Borisov *et al.*, 2016) attests to the potential of neural networks in ranking. In particular, the ability of deep neural networks in learning an effective representation for query-document pairs from raw, unstructured data opens a new frontier in the ranking research.

Decision forests and deep neural networks represent the most common classes of functions in LtR. While these classes are highly effective, their inherent complexity leads to a number of challenges. To illustrate one such challenge, consider inference. Computing a relevance score for a query-document pair from a decision forest involves traversing many decision branches in a large number of decision trees. Similarly, doing a forward pass through a deep neural network to compute a relevance score requires a large number of matrix

multiplications, each of a considerable size. Finally, producing a single ranked list for a query involves the computation of relevance scores for a large number of query-document pairs. Doing so within a small time budget, therefore, necessitates efficient data structures and inference algorithms. We explore these specialized tools for decision forests and neural networks in the remainder of this work.

3 Efficiency Challenges in Learning to Rank

The modern web search engine is a complex software with one main objective: to identify and return the subset of documents that are more relevant to a user query from a much larger set of all known documents. In the preceding chapter, we reviewed the ingredients of an LtR model and the machinery of its supervised training without explaining how a trained model is used within a search engine and what challenges we may face in adopting a complex ranker for the task above. We examine these unexplored questions in this chapter by describing the anatomy of a ranking pipeline and identifying the costs and efficiency challenges associated with each component at a high level.

Before we even get to the ranking part of a search engine, we should address a more immediate problem. It is clear that, due to the sheer size of document collections, it is simply infeasible to rank all documents known to a search engine in response to a query with a complex LtR model. Instead, we usually first apply a lightweight *retrieval* mechanism to find a smaller subset of documents that potentially match a query. This may be a dense retrieval method over representations learnt by a deep neural network where a match is determined by how similar the representation of a document is to the representation of the query, or it may be a statistical score defined for terms and phrases from the vocabulary where a document is deemed a potential match if it scores high—the latter is also known as sparse or lexical retrieval.

We do not delve into the algorithmic details of dense retrieval methods which often (but not always) use approximate nearest neighbor search algorithms, or lexical retrieval methods which often (but not always) operate over inverted indices. However, we highlight the importance of efficient index structures and top-k retrieval algorithms over index structures, and present the following efficiency challenge:

Efficiency Challenge 1. Given a query q and a large collection of documents $\mathcal{D}$, we seek a *space-efficient* data structure known as an index $\mathcal{I}$ to represent $\mathcal{D}$ and a time-efficient algorithm $\mathcal{A}_{\text{RETRIEVE}}$ that operates on $\mathcal{I}$ and returns the top-k documents that are most similar to q.

A great body of Information Retrieval literature and beyond investigate this particular challenge. We refer the interested reader to these works and citations therein for more

details (Asadi and Lin, 2013a; Asadi and Lin, 2012; Asadi and Lin, 2013b; Wang *et al.*, 2021a; Malkov and Yashunin, 2016; Petri *et al.*, 2019; Mackenzie *et al.*, 2021; Ding and Suel, 2011; Broder *et al.*, 2003; Mallia *et al.*, 2022). Throughout the rest of this monograph, we take for granted the existence of an efficient index and retrieval algorithm as a first step in processing a user query, and focus instead on the LtR stage.

While the retrieval step above greatly reduces the problem size, it does not change the overarching goal; we must still train a ranker and apply it to every retrieved document to compute relevance scores and return a ranked list.

Consider the training of an LtR model. As discussed in Chapter 2, we need labeled training data in the form of queries and sets of documents, which we then use to learn the parameters of a ranking function with the objective of optimizing a ranking loss. It is clear that the efficiency of the training procedure depends on the size of the data collection (as larger datasets require larger storage capacity and lead to longer training duration) as well as the complexity of the parameterized function (as a larger set of parameters requires exponentially more tuning). In addition to memory and time requirements, a training procedure that utilizes more data and requires more parameter updates is likely to result in higher energy consumption. This last point is particularly acute when the parameterized function is the class of deep neural networks (Scells *et al.*, 2022; Strubell *et al.*, 2019; Xu *et al.*, 2021). Together, these factors present the following efficiency challenge:

Efficiency Challenge 2. We seek a *sample-efficient* learning algorithm $\mathcal{A}_{\text{TRAIN}}$—requiring as few training data points as possible—to learn a parameterized function $f(\cdot, \cdot; \Theta)$ with *minimal complexity* required, in an *energy-* and *time-efficient* manner, such that f yields a desired quality measure on unseen data.

Once we have trained a model efficiently, we must apply the learnt function f to user queries in production. A naïve design to accomplish this goal would be to use an LtR model in a single stage, as depicted in Figure 3.1. The resulting ranking architecture is aptly called the *single-stage* pipeline.

It turns out that even with an effective retrieval method, the set of matching documents may yet be too large for an LtR ranker to process *efficiently*. That is because computing a single relevance score requires the execution of two potentially expensive operations. First, the set of query-document pairs must be translated into feature vectors—a phase that is known as "feature computation" or "feature extraction." Second, the model must be applied to each feature vector, which as discussed in the previous chapter, may involve computationally-expensive operations such as tree traversal or matrix multiplication. As the complexity of features and models increase, the cost incurred by these operations may become prohibitive, to the point where computing relevance scores for an entire set of retrieved documents may be impractical. As such, the inference procedure above involves addressing a number of efficiency challenges, which we summarize as follows:

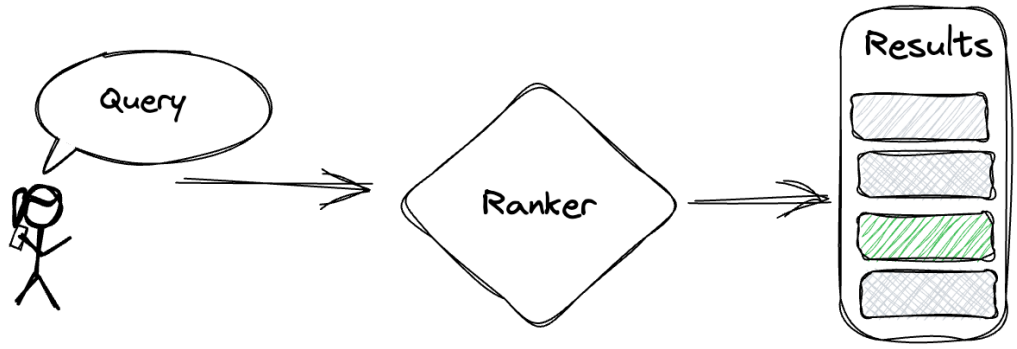


Figure 3.1: Single-stage ranking pipeline

Efficiency Challenge 3. Given a query q and a retrieved set of documents S_q by algorithm $\mathcal{A}_{\text{RETRIEVE}}$, we seek a *time-efficient* algorithm $\mathcal{A}_{\text{INFER}}$ that first represents the set $\{(q, x_i) \mid x_i \in S_q\}$ in a $|S_q| \times d$-dimensional feature space $\mathcal{X} \subset \mathbb{R}^{|S_q| \times d}$ and subsequently applies the function $f(\cdot, \cdot; \Theta)$ learnt by $\mathcal{A}_{\text{TRAIN}}$ to each query-document pair and orders them in decreasing order of relevance scores.

We have so far described the challenges inherent in retrieval, and training and inference of an LtR model. In the remainder of this chapter, we will describe high-level ideas that help address some of these challenges. We start, however, with inference and visit training efficiency last.

3.1 Efficient inference

How may we achieve effective but efficient ranking given a trained model or a collection of trained models to address Challenge 3? At a high level, the answer is quite intuitive and follows how we split the ranking problem to one of retrieval-then-rank: the set of retrieved documents can go through multiple stages, where each stage weeds out less-relevant documents and passes to the next stage a more promising but much smaller subset, and where each stage uses a more complex ranking model with increasingly sophisticated features than the stages before it. This paradigm is known as the *multi-stage* ranking pipeline. But to understand how we arrived at this solution, we must dissect the two operations involved (i.e., feature computation and model inference) and identify the factors that contribute to the overall cost.

3.1.1 Feature computation

Feature computation deals with the computation of query-document features that are given as input to the model to compute relevance scores. This task can be computationally expensive for a number of reasons. First, the number of features used in modern rankers is typically large with hundreds of features describing a single query-document pair. Second, each feature has its own intrinsic complexity: it can be a composition of more basic signals, or itself be the output of another machine-learnt model. This added complexity is justifiable because more sophisticated features often offer a higher discriminative power than basic, cheap-to-compute features such as term frequency.

On that basis, determining the appropriate set of features involves an implicit trade-off:

Trade-off 1. Using a large number of sophisticated features likely leads to improved ranking quality but also increased overall query processing time.

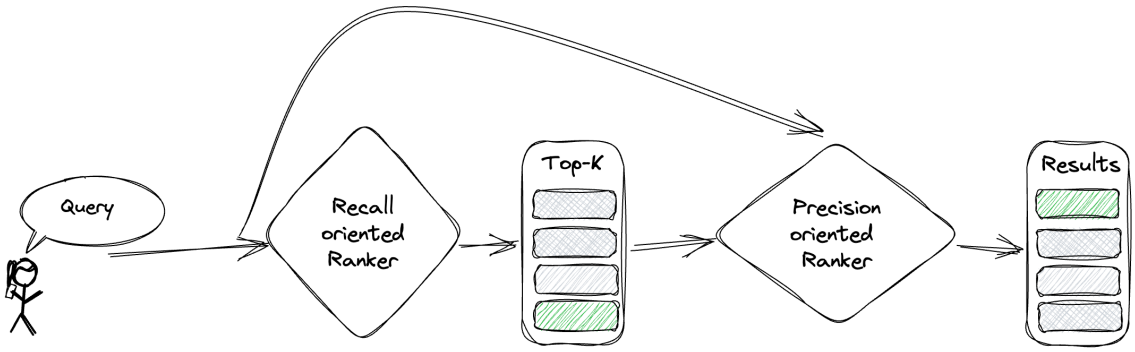


Figure 3.2: Two-stage ranking pipeline

The two-stage design

One idea to rein in the cost of feature computation is to break up the ranking pipeline into two stages, as we illustrate in Figure 3.2. The first stage in this design is in charge of executing a recall-oriented ranking of documents. It is important for this stage to employ simple rankers using cheap-to-compute features, to keep at bay the total cost of ranking the potentially large set of retrieved documents. The second stage, which only ever observes the top-K documents (also known as "candidates") as ranked by the first stage, is free to apply a precision-oriented, complex ranker to re-rank the candidates and produce a final ranked list.

The two-stage design greatly shrinks the set of documents for which we must compute expensive features, and as a result, reduces the overall cost of feature computation. It

therefore makes it feasible to use complex LtR models to produce effective ranked lists but do so efficiently. However, materializing this design requires choosing one key parameter: the cut-off value K that caps the number of documents that the second stage must re-rank. This choice presents our second trade-off:

Trade-off 2. A large K leads to a larger set of candidates to re-rank, in turn, increasing the cost of the second-stage ranker while potentially facilitating a higher-quality final ranked list. A small K, on the other hand, enables faster ranking in the second stage by passing a smaller set of candidates to re-rank, while potentially hurting quality.

The trade-off above has been the subject of much research in the past. Macdonald *et al.* (2013) demonstrated empirically that the cut-off value does indeed affect ranking performance. The authors evaluated the impact of K on two public document collections and gave a detailed analysis of the performance of two-stage pipelines where the first stage used statistical retrieval models (e.g., BM25 and DPH) and the second stage applied pointwise, pairwise and listwise LtR models.

Approaching this trade-off from a slightly different angle, Dang *et al.* (2013) investigated the recall bias of the first stage ranker. They found that the performance of the first stage affects the second-stage ranker in two unsurprising ways: (1) by influencing the quality of the training data available to learn the ranking model; and, (2) by controlling the number of relevant documents observed by the learnt model. The authors then showed that by using a learnt, yet fast model in the first stage, not only did recall improve in the first stage, but so did the overall performance of the second stage ranker.

This ability to trade off effectiveness for efficiency makes the two-stage design suitable for real-world applications where quality and speed are both critical to users. There is indeed evidence in the literature to support this speculation. Yin *et al.* (2016), for example, describe the query processor at Yahoo search engine as a distributed system deployed on hundreds of machines where each search node retrieves "hundreds of thousands" of candidates for a subsequent stage to re-rank. Another known deployment of this architecture is Alibaba's e-commerce search engine (Liu *et al.*, 2017).

The multi-stage design

With all the benefits the two-stage design has to offer, as observed by study after study, and all the knobs it provides to choose the right balance between efficiency and effectiveness, it is natural to wonder if one could simply extend the design to more than two stages. In the work by Yin *et al.* (2016), for example, the ranking pipeline is actually comprised of three stages, as shown in Figure 3.3. As in the two-stage design, the first two stages, which they call "Core Ranking," find top candidates for a query and re-rank those to produce a high-quality ranked list. The third stage, dubbed "Contextual Re-ranking," extracts features

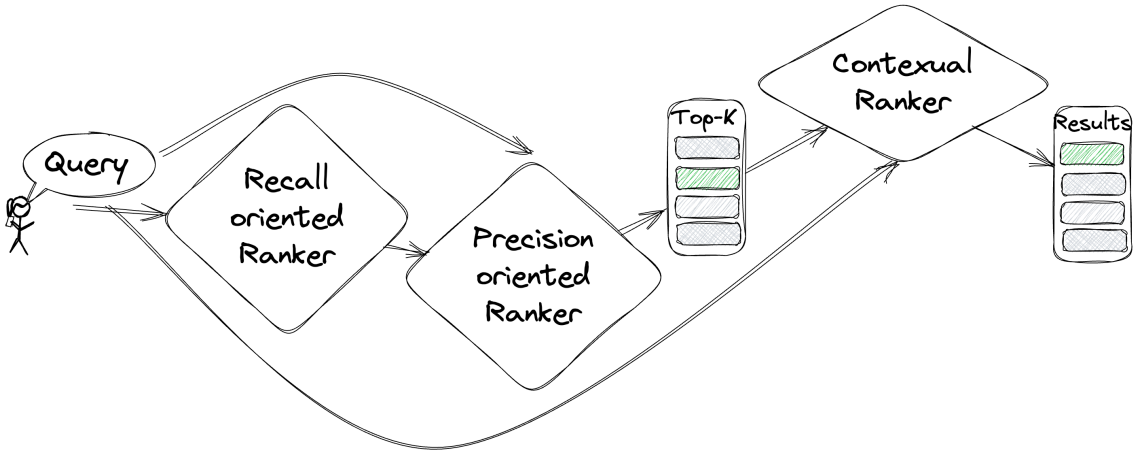


Figure 3.3: Three-stage ranking pipeline

that capture contextual information about the entire list (e.g., rank, feature mean and variance, normalized features, topicality), and uses the resulting richer feature set to re-rank the candidates again. This idea of leveraging contextual, rank-based features showcases the flexibility of a stage-wise view of ranking, and has been shown in other independent studies to greatly improve both ranking quality, and, when applied wisely, speed (Lucchese *et al.*, 2015a).

Given the success of a progression from two stages to three, it is tempting to generalize the design to N stages, as shown in Figure 3.4. Although potentially more effective, the *multi-stage* ranking pipeline is characterized by an increased complexity due to the sequential nature of query processing: each stage has to wait for the output of its predecessor to begin processing the input candidates. There are other questions too: Which features and which model should be used in each stage? How many documents should each stage re-rank?

Chen *et al.* (2017) study a subset of these questions: Suppose we have, in some way, arrived at a particular number of stages in a multi-stage ranking system. Given this particular scaffolding, can we select features and the number of candidate documents passed between consecutive stages so as to maximize effectiveness and efficiency of the overall cascade? Chen *et al.* (2017) formalize this problem using the concept of *regularization* from machine learning and present an optimization framework to minimize the "cost" of a cascade—defined as the cost of computing a particular feature and the number of documents for which this feature must be computed—while maximizing its ranking precision. For example, through $\ell 1$-regularization, one can enforce a certain degree of sparsity in the set of features used within a single stage of the cascade; a more aggressive sparsity rate yields a more compact, but potentially less effective stage.

In a follow-up study, Gallagher *et al.* (2019) identify another gap in the construction

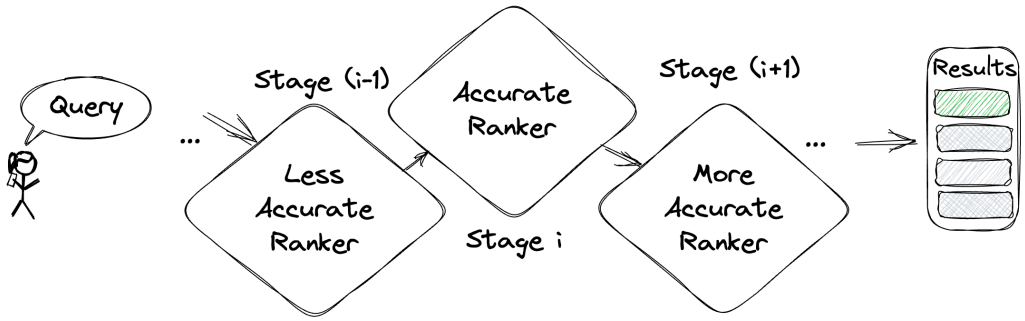


Figure 3.4: Multi-stage ranking pipeline

of multi-stage ranking systems: The models employed within individual stages are often learnt independently of one another, while in reality the decisions and rankings of one stage affects that of subsequent stages. The authors posit that the "stage-wise" ranking loss and the global effectiveness and efficiency objectives of a cascade can be optimized *jointly* using backpropagation. The key insight that enables gradient-based optimization of a cascade is that whether a document enters a stage but is dropped within that stage (i.e., document is *covered* by stage) can be expressed as an indicator function, which can be smoothed and differentiated.

3.1.2 Model inference

We have just seen how the multi-stage design offered a way to manage the cost of feature computation by introducing levers that allow us to trade off speed for quality. That included using simpler features in early stages and computing more complex features in later stages where we have fewer documents to re-rank. In our discussion, we also hinted that rankers in early stages are typically "simpler" and that we are free to use more "complex" models as we get to later stages. But what makes a ranking model more complex than others?

To answer the question above, it helps to consider the wide array of algorithms that the LtR literature has to offer. Many of these learn a ranking function with very few parameters, therefore requiring few operations to compute a relevance score for a given feature vector. Examples include Coordinate Ascent (Tseng *et al.*, 1988), Ridge regression (Hoerl and Kennard, 1970), SVM-Rank (Joachims, 2002), and RankBoost (Freund *et al.*, 2003). But there are also models that comprise of deep learning modules or hundreds or thousands of deep decision trees, resulting in large matrices to be multiplied sequentially or an exponentially large number of comparisons to evaluate recursively. These include GBRT (Friedman, 2001), Initialized GBRT (Mohan *et al.*, 2011), LambdaMART (Burges, 2010), and large language model-based Rankers (Lin *et al.*, 2021).

Even once we choose an LtR algorithm, we are often in control of the complexity of the model it learns. For example, in tree-based algorithms, we can cap the number of leaves each tree is allowed to have, or limit the maximum number of trees in the ensemble, all by adjusting the corresponding hyperparameters in the training algorithm.

Given the diverse set of algorithmic choices before us, it is not surprising that numerous studies have in the past conducted a comparative analysis of models based on their complexity (Tax *et al.*, 2015; Capannini *et al.*, 2016; Liu *et al.*, 2017; Macdonald *et al.*, 2013; Scells *et al.*, 2022). Capannini *et al.* (2016), for example, show that complex models—in particular, those based on decision trees—achieve significantly higher quality. They conclude that choosing the best model depends on the time budget available for query processing, and propose an objective—the Area under the Quality Cost Space (AuQC)—to compare different algorithms in terms of their accuracy-latency requirements. Scells *et al.* (2022), as another example, compare a range of models from decision tree-based to large language model-based rankers and observe significantly higher energy consumption in more complex models, adding a new but important dimension to the efficiency of ranking algorithms.

These empirical observations lead to a third trade-off between efficiency and effectiveness stemming from inherent complexities of LtR models:

Trade-off 3. Models that have fewer parameters and thus require fewer operations for evaluation are fast to execute and consume less energy typically at the expense of ranking quality. The flip side is that more complex models achieve higher effectiveness but incur a significantly higher computational cost and energy consumption.

3.2 Efficient training

Unlike inference, there have been relatively few studies in the LtR literature that investigate Challenge 2: efficient training. This is because up until the advent of deep learning, training even the most complex decision forests for LtR required a relatively modest number of training data points and the algorithms used to learn individual decision trees themselves would complete reasonably fast on general-purpose CPUs. As such, most training procedures were considered sample-, space-, and time-efficient, thereby rendering efficiency in training a non-issue.

That changed with the arrival of deep learning models, whose training needs vast datasets—thereby resulting in larger sample and space requirements—and involves computationally-intensive operations—in turn, requiring longer training duration on specialized, energy-hungry hardware.

The march towards ever larger datasets and ever more complex deep learning-based ranking models led Scells *et al.* (2022) to study the training efficiency challenges with a particular focus on environmental impact. The authors conducted a comparative study of

widely-used LtR models in terms of time-efficiency and effectiveness, as well as their power usage, which can be translated into the amount of CO_2 emissions. Their comprehensive study reveals that the training of deep learning models yields orders of magnitude larger emissions as compared to decision forests. They conclude their study by offering a framework for Information Retrieval researchers to alleviate some of the environmental costs of developing deep learning retrieval and ranking models.

As shown by a few recent works (Scells *et al.*, 2022) and events (Bruch *et al.*, 2022b; Bruch *et al.*, 2023), there is increasing interest in the efficiency challenges during training, driven by the urgency created by the environmental costs of recent ranking models. But more research is needed to help understand the trade-offs and offer solutions.

The challenges and trade-offs reviewed in this chapter capture the existing research in the efficient LtR literature. In the subsequent chapters, we present a detailed analysis of state-of-the-art solutions that explore these trade-offs to improve the efficiency of LtR models. As we pointed out earlier, because decision forests and neural networks require different types of intervention, we study them separately.

4 Tree-based Learning to Rank

Consider a basic supervised learning task where we have a labeled set of data points $\mathcal{D} = \{(x_i, y_i) \,|\, 1 \leq i \leq |\mathcal{D}|\}$ with $x_i \in \mathbb{R}^d$ being a d-dimensional real-valued vector of features and $y_i \in \mathbb{R}$ a target label. As usual, we wish to learn a function $F : \mathbb{R}^d \to \mathbb{R}$ that predicts a label y for an example x by optimizing the empirical loss $\mathcal{L}_\mathcal{D}$:

$$\mathcal{L}_\mathcal{D}(F) = \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \ell(y, F(x)),$$

where $\ell(\cdot)$ is a loss function such as the Squared Error (MSE).

A familiar approach to learning F is to parametrize it with a set of parameters Θ, $F(\cdot; \Theta)$, and find a Θ^* that leads to an optimal $\mathcal{L}_\mathcal{D}$. In other words, we first choose the family of functions that we believe represents the relationship between examples and labels—which may be a line, a neural network, or a decision tree—and then find the right shape by adjusting its parameters. We typically find Θ^* iteratively by applying gradient descent where at each iteration we take a step proportional to the negative gradient of F with respect to the current Θ:

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \eta \nabla_\Theta F,$$

where η is the learning rate.

Contrast the above with a different formulation where we assume that F can be broken up into additive components as follows: $F(x) = \sum_t f_t(x)$. f_t's, which are known as *weak learners*, may be any arbitrary function including parameterized families of functions,

$f_t(\cdot;\, \Theta_t)$. We learn F, again, iteratively, but here at the m-th iteration we learn the weak learner f_m to approximate the *residual* error: the negative gradient of the loss $\mathcal{L}_\mathcal{D}$ with respect to the current function $F^{(m-1)} = \sum_{t=1}^{m-1} f_t(x)$. This quantity is also known as the *pseudo-response* and is defined as follows:

$$g_m(x_i) = -\frac{\partial \mathcal{L}_\mathcal{D}(y_i, F)}{\partial F}\Big|_{F=F^{(m-1)}(x_i)}.$$

For example, when $\mathcal{L}_\mathcal{D}$ is the MSE, g_m is simply $y - F^{(m-1)}$. Said differently, f_m is learnt in a supervised manner on a new copy of the dataset $\mathcal{D}^{(m)} = \{(x_i,\, g_m(x_i)) \,|\, x_i \in \mathcal{D}\}$ where the residuals are now the labels. Finally, we scale f_m by a regularizing *shrinkage* factor, ν, and add it to $F^{(m-1)}$ to obtain $F^{(m)}$.

The learning framework just described is known as gradient boosting. It can be thought of as performing gradient descent in the space of functions, where instead of adjusting the parameters of a function at each step to reduce error, we learn an entirely new function that accounts for the leftover error and add it to the ensemble.

Within this framework, when the weak learners are the class of decision trees, we refer to the resulting *forest* as gradient boosted decision trees or GBDTs. Similarly, when the decision trees have real-valued leaves—also known as regression trees—we use GBRTs as a shorthand.

GBRTs are central to a powerful class of LtR algorithms and, as such, are the topic of the next few chapters. But before we proceed, we must explain how GBRTs are adapted to the ranking task.

4.1 GBRTs and learning to rank

It is easy to see that the gradient boosting framework described earlier is quite general and flexible, and that it can extend to specific learning tasks simply through the use of a differentiable loss function that is appropriate for the desired task. In theory, then, LtR with GBRTs is a matter of plugging in a ranking loss function as $\mathcal{L}_\mathcal{D}$ that is applicable to a ranking dataset $\mathcal{D} = \{(q_i, \boldsymbol{x}_i, \boldsymbol{y}_i) \,|\, 1 \leq i \leq |\mathcal{Q}|\}$, which, as a reminder, is a set of tuples with each tuple comprising of a query $q \in \mathcal{Q}$, a set of documents belonging to that query $\boldsymbol{x} = \{x_1, x_2, \ldots, x_{|\boldsymbol{x}|}\}$, and a set of relevance labels $\boldsymbol{y}$ corresponding to those documents. The only challenge specific to LtR, as we explained in Chapter 2, is deriving a surrogate smooth loss function that is more amenable to gradient boosting than ranking metrics.

As we have already seen, the need for surrogate losses is a central question in LtR research whenever the optimization algorithm requires meaningful gradients. It is in no way unique to gradient boosting or GBRTs. Naturally then, any of the ranking loss functions reviewed in Chapter 2 are reasonable candidates and, indeed, some such as the cross-entropy ranking loss (Bruch, 2021) have been implemented with GBRTs.

What makes GBRTs stand out, however, is the observation that all one needs to conduct a boosting step are the residuals: we need not necessarily have a closed-form loss function $\mathcal{L}_{\mathcal{D}}$, so long as its g_m's are known to us. This simple observation inspired LambdaMART (Burges, 2010), an early but influential LtR algorithm.

LambdaMART designs the residuals at each iteration heuristically and leaves the existence of a loss function with those gradients to assumption. Concretely, the residual of document $x_k \in \boldsymbol{x}$ belonging to a query q is a sum of pairwise quantities as follows:

$$g_m(x_k) = \sum_{l:\, x_l \in \boldsymbol{x}} \lambda_{k,l}.$$

Each $\lambda_{k,l}$ is the multiplication of two factors. One measures the distance between the scores of documents x_k and x_l using a sigmoid transformation. The other sorts the documents in $\boldsymbol{x}$ by their scores up to the current iteration (i.e., $F^{(m-1)}(x_i)$), swaps the positions of documents x_k and x_l, and measures the change in the metric we wish to optimize. So, for instance, if our metric of interest is NDCG, $\lambda_{k,l}$ would materialize as follows:

$$\lambda_{k,l} = \frac{1}{1 + e^{-(s_k - s_l)}} \times \Delta\text{NDCG}_{k,l},$$

where $s_o = F^{(m-1)}(x_o)$ and $\Delta\text{NDCG}_{k,l}$ is the change in NDCG when x_k and x_l trade positions in the ranked list. Note that the algorithm uses relevance labels $\boldsymbol{y}$ to compute NDCG.

Once the residuals of every document of every query are computed, we have all that is necessary to finalize one boosting step following the general recipe of gradient boosting. Repeating this process results in a forest of GBRTs that can be readily used as a ranker.

4.2 The prominance of tree-based learning to rank

GBRT-based LtR algorithms rose to prominence not just in academic research, but also in real-world applications in industry thanks to their unrivaled effectiveness. In tasks ranging from Ads Click Prediction at Facebook (He *et al.*, 2014) and Microsoft (Ling *et al.*, 2017), to product and document ranking at Amazon (Sorokina and Cantú-Paz, 2016) and Yahoo! (Yin *et al.*, 2016), to forecasting and recommendations at Yandex, GBRT-based rankers have played a major role. Winning solutions in many machine learning competitions in recent years too were centered around GBRTs. LambdaMART, for example, was the winner of the Yahoo! LtR Challenge (Chapelle and Chang, 2011). In a competition hosted by Kaggle in 2015 too the majority of the winning solutions used GBRTs (Chen and Guestrin, 2016). According to the same source, so did the top 10 teams who qualified in the KDDCup 2015.

The expansion of GBRTs and GBRT-based LtR algorithms to a larger and more varied range of applications has inspired novel implementations of gradient-boosted tree learning

algorithms. Among them XGBoost (Chen and Guestrin, 2016), LightGBM (Ke *et al.*, 2017), and CatBoost (Prokhorenkova *et al.*, 2018) offer state-of-the-art results with comparatively lightweight training routines. That, in turn, led to a variety of evaluation and analysis frameworks for LtR (Lucchese *et al.*, 2020a; MacAvaney *et al.*, 2022; MacAvaney *et al.*, 2020a; Lucchese *et al.*, 2017a).

The instances above highlight the continued importance of GBRTs in machine learning—and, in particular, in LtR—even in the face of the recent successes of deep learning. That is the reason why we study this family of algorithms in the next few chapters of this monograph.

5 Training Efficient Tree-based Models

Is it possible to train a tree-based LtR model that is efficient during inference? In other words, given we have identified and are aware of the factors that challenge the efficiency of an inference algorithm in Chapter 3, can we use that knowledge to train a model that does not incur high efficiency costs during its application? This chapter reviews methods that explore the trade-offs between inference efficiency and effectiveness *while* learning a ranking model. Broadly, these methods approach the problem in two different ways: (1) by presenting variations of the learning algorithm to address efficiency while training the ranking model and (2) by applying post-hoc optimization to a trained model in a post-processing phase. We review these methods in order.

5.1 Optimizing inference efficiency while learning

There is a large class of methods that aim to reach inference-efficiency while learning a ranking model. This research effort dates back to the work of Wang *et al.* (2010b) who first introduced the notion of *temporally-constrained ranked retrieval* as a desirable property of a ranking algorithm. They argued that, equipped with this property, the ranking algorithm can cope with diverse users and information needs, and better manage load and variance in query execution time.

As a way to induce the property above in an LtR model, Wang *et al.* (2010a) proposed a unified framework for jointly optimizing effectiveness *and* inference efficiency during learning. The idea is simple: Instead of optimizing an effectiveness metric during the training of a model, the new framework optimizes a hybrid metric that balances efficiency and effectiveness. Dubbed the "Efficiency-Effectiveness Trade-off" metric (EET) is a weighted harmonic mean of some measure of efficiency and effectiveness, which can be stated more formally as follows:

$$EET(q) = \frac{(1+\beta^2) \cdot \gamma(q) \cdot \sigma(q)}{\beta^2 \cdot (\gamma(q) + \sigma(q))},$$

where $\sigma(q)$ and $\gamma(q)$ are two functions $\mathbb{R} \rightarrow [0, 1]$ mapping efficiency and effectiveness of a model for a given query Q into the unit interval.

Most query effectiveness metrics have the form above, including precision, recall, average precision, and NDCG. The authors use average precision for $\gamma(\cdot)$ in their work, but any of the other metrics too can be plugged into the hybrid metric. As for the measure of efficiency, $\sigma(\cdot)$, the authors take the query execution time (measured in seconds) as input and map it to a efficiency score in the unit interval such that 0 and 1 represent an inefficient and efficient ranking, respectively. For example, consider the following mappings: *constant*, *step*, *exponential*, and *step + exponential*.

The *constant* function always maps the query execution time to the same value independent of the actual execution time. The *step* function computes a score of 1 for all queries whose execution time does not exceed a given threshold (e.g., 300 msec.) and 0 for all other queries. The *exponential decay* function allows for a softer penalization of increasing query execution times. Finally, the *step + exponential decay* function is a combination of the two previous functions that allow for a soft penalization of query execution times above a given threshold. Figure 5.1 shows the four different $\sigma(q)$ functions introduced to quantify the efficiency behavior of a query q.

The EET metric is a weighted harmonic mean. In fact, the contribution of efficiency and effectivenss factors to the final measure is controlled by a hyper-parameter, β, which in the original work of Wang *et al.* (2010a) is set to $\beta = 1$. Note that, EET measures the efficiency-effectiveness trade-off on a per-query basis. As such, the training algorithm optimizes the mean EET, called MEET(q) and defined as $\frac{1}{N}\sum_{i=1}^{N} EET(q_i)$, to learn a ranking model on a set of N training queries.

Suppose we wish to learn a linear function of the following form, as in the work of Wang *et al.* (2010a):

$$S(q, d) = \sum \lambda_i f_i(q, d),$$

where q is a query, d is a document, $S(q, d)$ is the ranking score produced by the function, $f_i(q, d)$ is a feature computation function, and λ_i is the weight assigned to feature i. How may we make the function more efficient to compute during inference? One idea is to introduce an L_1 penalty in the learning objective, thereby encouraging the model to learn sparse weights. That results in a function $S(\cdot, \cdot)$ where a large number of $lambda_i$'s have a value close to 0. It is now reasonable to disregard feature whose weights are close to 0 as their contribution to the final score is small, thereby obviating the need to compute those features for a query-document pair during inference, and as a result improving inference efficiency with little impact on effectiveness.

Wang *et al.* (2010a) put that hypothesis to the test and conduct experiments on three TREC web collections: Wt10g, Gov2 and ClueWeb09 (part B). The empirical results show that models learnt by optimizing MEET achieve a good balance between effectiveness and

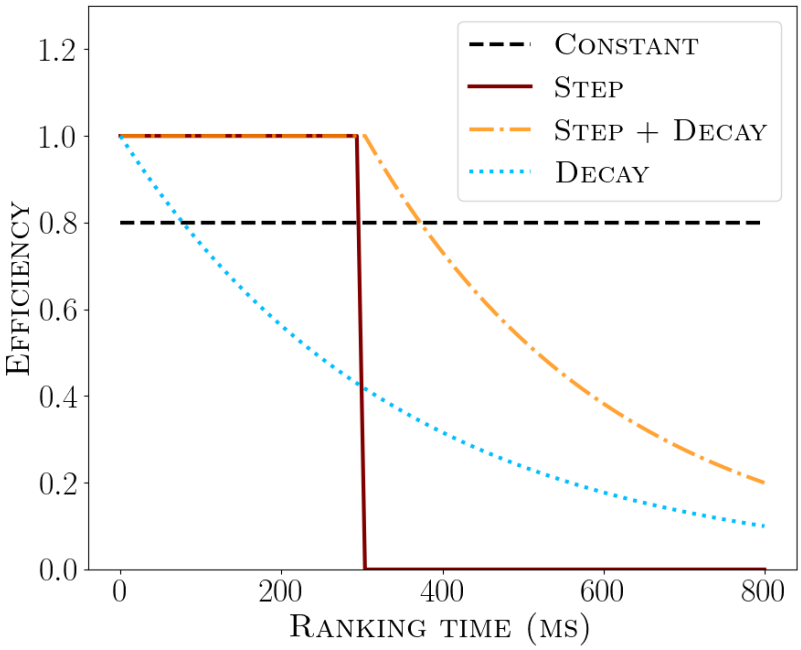


Figure 5.1: The four different $\sigma(q)$ functions measuring efficiency introduced by Wang *et al.*. The figure is redrawn based on (Wang *et al.*, 2010a).

efficiency. A comparison of the query likelihood (QL) model or the sequential dependence model (SD) (Metzler and Croft, 2007) shows that mean query evaluation time for MEET-optimized models is greater than that of QL, but less than that of SD. As expected, increasing the decay rate in a exponential decay flavor of MEET reduces the mean query evaluation time, suggesting that MEET is able to take efficiency into consideration during the learning process.

In a later study, Xu *et al.* (2013) observe that in real-world web search engines the time available for evaluating and applying a machine learning model is budgeted (Kohavi *et al.*, 2013; Lucchese *et al.*, 2015b; Chapelle *et al.*, 2011; Zheng *et al.*, 2008). As discussed before, this time budget is typically spent on computing features and evaluating the model itself. In the models studied by Xu *et al.* (2013), the inference time is often dominated by the computation required to perform feature values. Reducing the number of features required to confidently rank documents for a given query thus should greatly improve efficiency.

In contrast to Wang *et al.* (2010a) and similar works (Efron *et al.*, 2004; Dredze *et al.*, 2007) which discard entire features from the feature set and reduce costs equally across all queries, Xu *et al.* (2013) take a more dynamic, query-dependent approach. Figure 5.2 describes their solution pictorially. The figure on the left shows a model that comprises a cascade of classifiers (CSCC) where each classifier in the cascade terminates the inference for queries that it considers "easy," thereby reducing the total amount of computation necessary to confidently predict a relevance score for a query-document pair. Contrast that with Figure 5.2 (right), which instead illustrates a *tree of classifiers* (CSTC). The tree is used to classify test queries flowing along individual paths from root to leaf nodes. Each path computes different features and is optimized for a specific sub-partition of the input space. This tree structure allows us to decrease the feature computation cost by computing only the features that benefit a given input the most. This is possible because the input space is partitioned by the tree and different features are only computed when they contribute most heavily to the final ranking quality.

Learning such a structure requires an query-dependent feature selection strategy and a dynamic allocation of time budgets for features used in different tree paths (e.g., infrequent paths require a larger share of the inference time budget). As such, we need a new learning objective to optimize. Xu *et al.* (2013) introduce the following loss function to learn a CSTC:

$$\min_{\beta^0,\theta^0,\ldots,\beta^{|V|}\theta^{|V|}} \sum_{v^k \in \mathcal{V}} \underbrace{(\frac{1}{n}\sum_{i=1}^{n} p_i^k \ell_i^k + \rho|\beta^k|)}_{\text{regularized loss}} + \lambda \sum_{v^l \in \mathcal{L}} p^l \underbrace{[\sum_{\alpha} c_\alpha \sqrt{\sum_{v^j \in \pi^l} (\beta_\alpha^j)^2}]}_{\text{test-time cost penalty}} .$$

This loss has two terms: a typical regularized loss and a inference-time cost penalty. These terms together encourage the learning process to re-use features that are already computed along a specific path, rather than computing additional features.

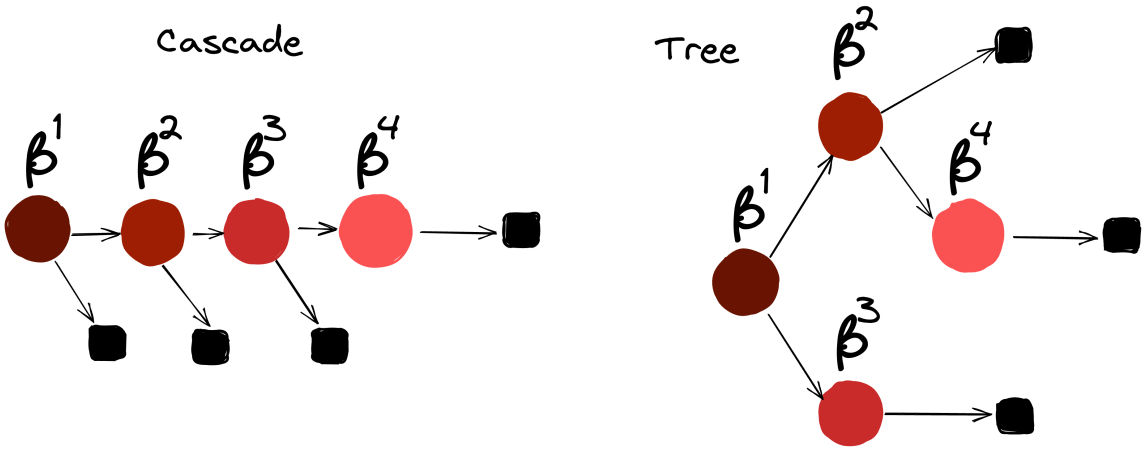


Figure 5.2: Cascade of classifiers (CSCC) vs Tree of classifiers (CSTC). Circular nodes represent classifiers (with their parameters β). Squares represent predictions. The color of each classifier indicates the number of inputs passing through it (darker shades indicate that a larger number of queries are evaluated by that classifier). The figure is redrawn based on (Xu *et al.*, 2013).

Xu *et al.* (2013) provide a comprehensive empirical evaluation of CSTC on the Yahoo! Learning to Rank dataset, with a comparison against several state-of-the-art methods such as stage-wise regression (Friedman, 2001), early-exit strategies (Cambazoglu *et al.*, 2010), and Cronus optimized (Chen *et al.*, 2012) in terms of NDCG@5.

Results confirm that CSTC achieves a higher ranking quality at a small fraction of the computational cost of other methods. The early exit methods achieve limited gains because the inference cost is dominated by feature computation, rather than model evaluation cost. On the other hand, CSTC has the ability to identify features that are most beneficial to different groups of queries, which in turn allows CSTC to achieve a higher NDCG.

The optimization of the loss above was made possible through the use of a mixed-norm relaxation of the L_0 norm, allowing the inference-time cost penalty term to be continuous and differentiable. Later, Kusner *et al.* (2014) observe that the CSTC problem is NP-hard and that Xu *et al.* (2013) developed an approximate solution through the mixed-norm relaxation technique. It turns out that the mixed-norm relaxation is slow to train and requires hyper-parameter tuning. To remedy these problems, Kusner *et al.* (2014) propose an alternative relaxation using approximate submodularity, called *Approximately Submodular Tree of Classifiers* (ASTC), which casts the objective as an approximate submodular set function optimization problem. This new relaxation proved much simpler to implement, yield equivalent results without the need for hyperparameter tuning.

The authors report the results of an empirical evaluation of ASTC on the Yahoo! Learning to Rank dataset as well as three other non-cost-sensitive datasets: Forest (tree type), CIFAR

(image classification), and MiniBooNE (particle identification). Their experiments show that ASTC performs just as well—and sometimes slightly better—than the state-of-the-art CSTC, while its training is up to two orders of magnitude faster.

5.2 Mixed optimization strategies of inference efficiency

In contrast to the methods we reviewed in the previous section which modify the learning process only, mixed strategies can take as input a model that has already been trained to optimize effectiveness and improve its inference efficiency through some form of post-processing without significantly degrading effectiveness. Alternatively, these algorithms could interleave training and pruning-based optimization to achieve the same objective. We study some of these methods in this section.

One work that highlighted the efficacy of post-hoc optimization is a comparative investigation by Asadi and Lin (2013c) of a cost-aware method to train GBRTs and a post-learning pruning of decision trees. The core idea behind their methods was inspired by the simple observation that the cost of traversing a decision tree is proportional to its depth and is a function of its structure, and therefore compact, shallow, and balanced trees must yield faster predictions. They then proposed two solutions as manifestations of that basic idea.

Their first method belongs, in fact, to the previous category of algorithms: Optimization of inference efficiency by modifying the training algorithm. Named "cost-sensitive tree induction," it modifies the splitting strategy used in learning individual decision trees. The idea is that, while growing a decision tree, instead of splitting the node that leads to the maximal gain, G^*, in the effectiveness loss function, we consider the set of all possible splits whose split gain is at least $(1-\tau)G^*$ for some $\tau \in [0,1]$. Of the splits in this set, the algorithm chooses the split that results in the shallowest depth. The hyper-parameter τ trades off efficiency for effectiveness: when $\tau = 0$ the algorithm ignores tree depth and is reduced to effectiveness-maximizing tree learning, but as τ increases splits that are possibly less effective but that do not add to the tree depth are selected.

The other realization of their idea, which is called "pruning while boosting," belongs to the second class. Once a decision tree in a GBRT forest is learnt using the standard tree learning algorithm, a post-processing step prunes some of the nodes so as to reduce the tree depth and create a shallower and more *balanced* tree. The pruning algorithm works as follows: Let $|t|$ be the number of nodes in the tree (including leaf nodes) and d_t be the depth of the tree. The greedy algorithm selects the two deepest leaf nodes and collapses them into their parent node so long as $|t| \geq \alpha \left(2^{d_t+1} - 1\right)$, for some hyper-parameter $\alpha \in [0,1]$ which controls the balance between efficiency for effectiveness. When $\alpha = 0$ no pruning occurs, while when $\alpha = 1$ the output tree is perfectly balanced. Note that the pruning process is

oblivious with respect to the loss function. The rationale behind this pruning is that the loss increment due to the pruning can be counter-balanced by subsequent trees that will be learnt in the GBRT forest.

Asadi and Lin (2013c) present experiments on the MSLR-WEB10K dataset and show that the pruning approach is much more effective. Cost-sensitive tree induction has a modest effect on efficiency (of approximately 1%): trees are slightly less deep, but more trees are generated in the forest and the total number of nodes in the GBRT ensemble is unaffected. The pruning approach, on the other hand, leads to an increase in the number of trees in the ensemble, but both the depth and the total number of nodes in the ensemble is halved. With proper tuning of α, it is possible to reduce the evaluation cost of the forest by approximately 40% while maintaining the same NDCG@5.

As noted earlier, the work of Asadi and Lin (2013c) shows the efficacy of post-processing approaches. The pruning method utilizes the full power of state-of-the-art learning algorithms to learn an effective model, while also reducing their inference cost by inducing a desired model structure. Despite this success, the core idea behind this work has a major caveat: Taking the left or right branch in a decision tree depends on the input data and, as such, paths are taken with different, non-uniform probabilities. Rather than creating balanced trees that minimize the depth of every path, it may make more sense to minimize the depth of the most likely paths. Additionally, as with (Lucchese *et al.*, 2015b; Ye *et al.*, 2018), inference cost may be a non-trivial function of the tree topology, where the tree depth may not be neatly correlated with efficiency.

Pruning at the ensemble level

While the pruning technique of Asadi and Lin (2013c) works at the tree level, Lucchese *et al.* (2016a) propose CLEAVER which instead operates at the ensemble level: instead of pruning nodes, entire decision trees are removed from the ensemble. The motivation is that the iterative GBRT learning algorithm may generate similar and redundant trees, especially when the learning rate is small. It must thus be possible to identify a subset of decision trees that contribute more significantly to the effectiveness.

To that end, given a forest of n trees and with the goal of producing a leaner forest of p trees, Lucchese *et al.* (2016a) consider six pruning strategies:

- RANDOM: A subset p is selected at random;
- LAST: The last $n - p$ trees of the ensemble are discarded;
- SKIP: One tree every $\lceil \frac{n}{p} \rceil$ trees is kept;
- LOW-WEIGHTS: As learning algorithms typically generate a weighted ensemble, the p trees with the largest weights are kept;

- Score-Loss: The p trees that contribute the most to the prediction score of the ensemble are kept; and,
- Quality-Loss: Given a ranking quality metric such as NDCG, the algorithm computes for each tree the degradation in quality if that tree were to be discarded from the ensemble, and subsequently selects the p trees that result in the smallest decrease in quality.

The hyper-parameter p controls the balance between reaching efficiency and effectiveness: When $p = n$, no pruning is performed and the original effectiveness is maintained. When $p < n$, fewer trees will remain in the ensemble but at the risk of degrading effectiveness. As a way to compensate for the possible degradation in quality due to the removal of the $n - p$ trees, the algorithm performs a tree re-weighting step to assign new weights to the surviving p tree. To compute new weights, the algorithm simply uses line search as it allows to locally optimize any given quality measure.

Experiments with λ-Mart on both MSLR-WEB30K and Istella-S show a dramatic reduction of the inference cost. Authors choose p so as to produce the smallest model that provides at least the same effectiveness as the full λ-Mart model. In their experiments, the largest model evaluated on MSLR-WEB30K has 737 trees. The size of the forest was reduced to 369 trees using the Quality-Loss pruning protocol, with a speed-up factor in the inference time of 1.9. Similarly, using Skip, a model of 736 trees on Istella-S was cut down to 368 trees with a speed-up factor of 1.8. Overall, Lucchese *et al.* (2016a) found that Quality-Loss is the most stable strategy across all experiments, due to the fact that it is aware of the target quality metric during the pruning process. The observation that a strategy as simple and cheap as Skip also provides decent boost to inference efficiency suggests that ensembles are typically very redundant, and therefore pruning followed by a re-weighting of trees is a successful way to improve the efficiency of such complex models.

We highlight that the post-processing strategies seen so far—pruning while boosting of (Asadi and Lin, 2013c) and CLEaVER of (Lucchese *et al.*, 2016a)—provide about the same savings in inference costs and result in an approximately 2.0 speed-up factor. While the two ideas are orthogonal, their combined effect on inference efficiency has not yet been explored.

In a follow-up work, Lucchese *et al.* (2018a) proposed X-CLEaVER, an iterative meta-algorithm that is able to learn more efficient and effective ranking ensembles. X-CLEaVER interleaves the iterations of a given gradient boosting learning algorithm with pruning and re-weighting phases. First, redundant trees are removed from the given ensemble, and then the weights of the remaining trees are fine-tuned by optimizing the desired ranking quality metric. The authors propose and analyze several pruning strategies, with a subset borrowed from (Lucchese *et al.*, 2016a). They assess the benefits of interleaving the pruning and re-weighting phases during learning instead of applying it as a single post-learning

optimization step. Experiments on the MSLR-WEB30K and Istella-S datasets show that X-CLEAVER can be successfully applied to several LtR algorithms and optimizes the effectiveness of the learnt ensembles, thereby obtaining more compact forests, making them more efficient at scoring time.

Interestingly, reducing the size of a complex model also brings the advantage of reducing the risk of *over-specialization*. This side of the problem is well investigated in (Vinayak and Gilad-Bachrach, 2015). Authors show empirically that later trees learnt by a GBRT algorithm influence the prediction of a very limited set of training instances and provide a negligible contribution to the rest. This *over-fitting* behavior, it is argued, affects the generalization power of the model to unseen test instances.

To overcome this limitation, the authors borrow the idea of *dropout* from deep learning (Hinton *et al.*, 2012). When dropout is applied to a layer in a neural network, it suppresses a random subset of neurons in that layer during training, with the intuition that the remaining active neurons cannot rely on a limited set of connections to compute their output. In the context of GBRTs, this translates into muting some of the previously learnt trees while learning a new decision tree. Once the new tree is learnt and added to the forest, the algorithm computes its weight as follows: if k trees are muted during training, then the predictions of the newly learnt tree are rescaled by a factor of $1/(k+1)$. The intuition is that, the new tree will likely learn to make large predictions to compensate for the absence of the muted k trees, and that scaling its predictions down attenuates their contribution to the final prediction. Finally, the muted trees are rescaled by a factor of $k/(k+1)$ so as to suppress their contribution to account for the presence of the new tree. In their experiments, Vinayak and Gilad-Bachrach (2015) show that the proposed algorithm, called Dart (Dropouts meet Multiple Additive Regression Trees), can improve the performance of λ-Mart for the ranking task on the MSLR-WEB10K dataset.

This idea was further explored by Lucchese *et al.* (2017b) who propose to replace the *muting* strategy with *pruning*. Their algorithm, named X-Dart, uses the same dropout approach with a crucial difference: At each iteration, either the muted trees are restored into the forest, or are removed from the forest. In particular, when the newly added tree leads to better quality metrics than the k muted trees, the muted trees are dropped, resulting in a forest with $k-1$ fewer trees. Together with an adaptive strategy to fine-tune k, X-Dart produces impressive performance: On the MSLR-WEB30K dataset, an X-Dart model with 500 trees yields the same NDCG@10 as a λ-Mart forest with 1200 trees. Similarly, NDCG@10 of 500 X-Dart trees is on par with a λ-Mart forest of 1500 trees on the Istella-S dataset. This line of work that was inspired by dropout reduces the size of a ranking forest, and improves its efficiency up to a factor of 3 without any loss in accuracy.

Knowledge distillation

Both X-DART and CLEAVER rely on the idea that given a large and effective model, we can find a smaller model that achieves improved efficiency while retaining the same effectiveness. In particular, these algorithms prune an existing model and find a subset of the model with the largest contribution to the final quality metric. But the idea that a small model can approximate a larger one is not new, and it is at the core of *knowledge distillation* (Ba and Caruana, 2014) in the deep learning literature.

In knowledge distillation, a complex *teacher* model is first trained on a given dataset, then a simpler *student* model is trained by minimizing the deviation of its prediction both from the ground-truth labels and from the teacher's predictions. The intuition is that, the teacher model, thanks to its greater complexity, captures complex relationships in the data and potentially removes noise, thereby providing a "cleaner" training signal to the student model. In the deep learning scenario, training a new student model from scratch is a more natural choice than pruning layers or other components of a deep neural network. Nevertheless, we might consider X-DART and CLEAVER as knowledge distillation approaches specifically tailored to decision tree forests.

The idea of knowledge distillation has been applied to LtR in other ways too. In what we regard as *homogeneous* distillation, such as the work of Tang and Wang (2018), the teacher and student models are both from the same hypothesis class. But Cohen *et al.* (2018) show that it may be more appropriate for the two models to come from different classes, which we call *heterogeneous* distillation. Let us dissect these two works as examples of knowledge distillation in LtR.

Ranking Distillation by Tang and Wang (2018) is an example of homogeneous knowledge distillation in the context of LtR. While in that work, the authors fall back to the usual (pointwise or pairwise) logistic loss to train the teacher model, the novelty of their approach rests in the way they train the student model, which has fewer parameters than the teacher: Rather the working on the full training set, the student model is allowed to evaluate only the k documents that received the highest ranking by the teacher model. The prediction error—again, based on the logistic loss—of those k documents is weighted by their rank, and it is adaptively tuned at each training iteration in a manner similar to boosting. Experiments on a recommendation task show that the student model performs even better than the teacher both in the case of convolutional neural networks and matrix factorization. In addition, the student model is about twice as fast as the teacher model. Interestingly, training a student model on the original training dataset provides significantly worse performance figures.

In an interesting turn, Cohen *et al.* (2018) argue that, in the context of LtR, decision tree ensembles like λ-MART are a better choice for the teacher model, but forests may not be all that appropriate for a student model because of the inherent efficiency challenges during inference. Instead, they propose to use a neural network as student model for the

following two reasons: neural networks should be able to learn the predictions of a decision forest thanks to the Universal Approximation Theorem, and modern hardware is highly optimized for the inference of neural networks. While deep networks are still too expensive, the authors show that a *medium*-sized network as a student can be effective and efficient.

In order to make this work, Cohen *et al.* (2018) create a new training dataset that consists of the original training instances used in learning a teacher λ-MART, and a set of randomly generated instances whose feature values are chosen so as to lie in between the different splitting points of the λ-MART forest. The labels of both types of training instances is the output of the teacher λ-MART model. Intuitively, the enhanced training dataset helps the student model approximate the behavior of the teacher on the original training instances as well as points close to its *discontinuities*.

Experiments on the MSLR-WEB30K dataset show that a feed-forward network with two fully connected hidden layers of 500 and 100 neurons, achieves results that are on par with the teacher model with any observed difference being statistically insignificant. Authors explore both CPU and GPU for inference and find that GPUs achieve up to 100× speed-up. While a fair comparison between a multi-threaded implementation of a neural network against a multi-threaded implementation of a forest traversal algorithm was not presented, this work overcomes the limitations of back-propagation for ranking by *mirroring* a λ-MART forest, and, in turn, benefits from advances in hardware that is typically used for computations in neural networks.

Finally, Nardini *et al.* (2022) explore the additivity of knowledge distillation, pruning, and fast matrix multiplication in bringing about inference efficiency. The authors first use the knowledge distillation framework to train shallow neural networks from an ensemble of regression trees. They additionally apply neural network pruning to the learnt network so as to induce more sparsity in its most computationally-intensive layers. The sparse, shallow network is then executed with a optimized sparse matrix multiplication algorithm. Their experiments on two public LtR datasets show that sparse neural networks produced with this approach are competitive at every point of the effectiveness-efficiency trade-off when compared with tree-based ensembles, leading to 4× speed-up during inference without adversely affecting ranking quality.

5.3 Open challenges and future directions

Table 5.1 summarizes the methods discussed in this chapter. We observe two main research directions pursued over the recent years. The first includes novel methods for learning regression forests wherein not only is an effectiveness metric maximized, but their inference cost is also taken into consideration. We identify two different ways of realizing this idea: (1) methods that quantify inference efficiency and optimize it while learning an LtR model,

(2) methods that introduce a more efficient organization of an LtR model so as to reduce the overall feature computation cost. We believe these works can be foundations for new, more complex approaches that learn cost-aware models for web search. Many of these ideas, for example, may be extended to directly learning complex ranking cascades.

The second class of algorithms attempt to condense an existing, effective model into a *smaller* model that is more efficient during inference but just as effective. This reduction in size not only affects a model's inference cost, but as highlighted by Vinayak and Gilad-Bachrach (2015), it can help to prevent over-fitting. Indeed, finding a model of the smallest complexity that achieves high effectiveness is not just an important question in ranking, it is a great challenge in machine learning in general. We expect that research in this area can not only contribute to the specific task of efficient ranking, but that it can also lead to contributions of a wider scope to the data mining community.

We saw that it is possible to borrow and translate ideas from the deep learning literature to design novel learning algorithms (c.f., DART) or use simple feed-forward networks to achieve effective ranking while benefiting from efficient hardware. It is very likely that novel advances in deep learning will affect how we think about LtR and help us design better algorithms. Though, we note that information retrieval systems often have additional requirements and constraints (such as on scale and time complexity) that create non-trivial challenges.

An interesting and relevant line of research which we did not cover in this chapter is that of online LtR. When we need to promptly explore user feedback and fine-tune or re-train a ranking model, the time constraints on the training procedure are even more strict. Oosterhuis and Rijke (2017) consider this problem and propose a cascading model, where a *fast-to-train* model is first optimized to provide reasonable effectiveness from a small number of user interactions, and only when this model converges, it is used to initialize a second *expressive* model. This direction of research highlights, once again, the importance of efficiency and creates new challenges for an LtR system.

6 Efficient Inference of Tree-based Models

In the previous chapter, we described ideas that either learn a model that is efficient during inference or reduce the efficiency cost of an already-trained model. An orthogonal idea is to improve the efficiency of model evaluation itself by reducing its computational complexity. This is particularly important in the context of decision forests because, in order to make a prediction, we must traverse each decision tree in the forest from root to leaf nodes, which involves the evaluation of a decision at every intermediate node and branching to the appropriate sub-tree. While methods from the previous chapter can help reduce the complexity and the size of a decision forest, we are nonetheless faced with the challenge of

evaluating the leaner but still large model.

We study that second problem in this chapter and review data structures and algorithms that help lower the costs of model evaluation. This includes improved traversal of decision trees, producing approximate predictions, and cascading models. The following sections explore these ideas in more depth.

6.1 Efficient traversal of decision forests

Let us depart from the realm of theory, and go back to the basics and consider how we may implement a decision tree from scratch. The most straightforward and perhaps naïve way of materializing a tree is by using some form of if-then-else blocks of code or a conditional or ternary operator depending on the language used. That is, at each node, we compare a feature value against a threshold and decide if we need to take the right or left branch. We continue executing similar decisions until we reach a leaf node.

While this implementation may be the most readable, it is computationally intensive. Yes, the resulting code may be compiled with whatever optimization strategy a compiler can muster, but regardless the size of the resulting code is proportional to the total number of nodes in the ensemble, and it is impossible to successfully leverage the instruction cache due to frequent branching in the code. Conditional blocks have proven to be efficient when the feature set is small (Asadi *et al.*, 2014), but it still suffers from *control hazard*, defined as instruction dependencies introduced by conditional branches.

Asadi *et al.* (2014) improve this implementation in a data structure they call Struct+. This data structure stores the feature id of intermediate nodes and the threshold used in the split, and has pointers to the left and right children. The traversal of the tree starts from the root and moves down to leaves according to the result of a boolean expression on the traversed nodes. But different from the naïve implementation, Struct+ uses an optimized memory layout that linearizes the tree nodes via a breadth-first traversal of the tree.

While the improved memory layout brings about advantages, Struct+ still has a few notable drawbacks. Importantly, the next node to be processed is known only after the boolean decision is evaluated and as such does not address the frequent control hazard. Its efficiency thus is a function of the branch mis-prediction rate. Another caveat is that, due to the unpredictability of the path visited by a given test instance, tree traversal has low temporal and spatial locality, leading to low cache hit ratio and poor CPU cache utilization.

How may we work around these limitations? Asadi *et al.* (2014) offer an algorithm called Pred that rearranges the computation such that control hazard is replaced with *data hazard*, defined as data dependencies introduced when one instruction requires the result of another. The algorithm works by first replacing pointers with node indices within a contiguous array of memory, then using the output of the binary expression directly to

compute the index of the next node. The traversal of a tree of depth d is then statically "un-rolled" into d operations, starting from the root node to the leaves. Leaf nodes are encoded so that their indexes generate self loops. At the end of the traversal, the algorithm identifies the leaf node and uses a look-up table to retrieve the predicted value of the tree.

Pred removes control hazards because the next instruction to be executed is always known. However, it now introduces data dependencies because the output of one instruction is necessary to execute the subsequent one. The algorithm does not address poor memory access patterns of Struct+ either because the path traversed depends on the test instance. Finally, Pred creates yet a new source of overhead: for a tree of depth d, even if a test instance ends in a shallower leaf, the algorithm executes all d instructions anyway. Asadi *et al.* (2014) remedy some of these limitations by creating a vectorized version of the algorithm, named VPred, which interleaves the evaluation of a small batch of documents. VPred was shown to be 25% to 70% faster than Pred on synthetic data, and to outperform other methods we discussed so far.

Feature-major traversal

The algorithms we discussed so far have taken a node-at-a-time view to evaluate a decision tree: When a test instance enters the decision tree at the root, these algorithms evaluate decisions in each recursively until they reach a leaf node. They then move onto evaluating the next document. Lucchese *et al.* (2015b) offer a different traversal pattern in QuickScorer by devising a feature-wise evaluation of decision trees.

QuickScorer traverses a complete forest by evaluating all the nodes that make a decision using the first feature, then all the nodes branching off of the second feature, and so on. Note that, the order in which features are selected is immaterial and may be arbitrary. The algorithm then creates a bit vector for each tree, called *leafindex*, which has as many bits as there are leaves in that tree, and updates it to mark the subset of leaves than will never be reached for the instance under evaluation. That some leaves will never be visited happens because of the structure of the tree: if an intermediate node that evaluates to *false*, then its right subtree is not visited. Each intermediate node too has a bit vector associated with it, which is called a *nodemask*. This bit vector encodes the leaves that are never reached should the condition in that node evaluate to false.

Given these bit vectors and the outcome of the feature-wise evaluation, QuickScorer takes the nodes whose condition evaluated to false, and performs a logical AND of the *leafindex* vector with that node's *nodemask*. After all false nodes have been processed, the *leafindex* bit-vector identifies the exit leaf for the test instance, which the algorithm uses to retrieve a value from a look-up table.

Due to this reorganization of the computation, QuickScorer's data structure can be implemented as a set of contiguous arrays, enabling fast linear scans and bit-wise operations.

Overall, because of these properties, QuickScorer exhibits a cache-efficient behavior. But Tang *et al.* (2014) show that cache utilization can further improve by what is called *blocking*; partitioning the tree ensemble into subsets of limited size, so that each subset can be processed entirely in cache. One can tailor the block sizes based on the different levels of CPU cache (Jin *et al.*, 2016). Lucchese *et al.* (2015b) apply these ideas to QuickScorer with different flavors of blocking (Dato *et al.*, 2016), and make further improvements through vectorization over multiple documents (Lucchese *et al.*, 2016b), multi-core and GPU parallelism (Lettich *et al.*, 2019). More recently, Gil-Costa *et al.* (2022) and Molina *et al.* (2021) propose a novel design of of the QuickScorer algorithm and the application of binning or quantization techniques to tree ensembles to fully leverage novel, energy-efficient field-programmable gate arrays (FPGAs).

Ye *et al.* (2018) take the data structure in QuickScorer and make it more compact in their algorithm, RapidScorer. The first observation was that *nodemasks* are two sequences of 1's separated by a sequence of 0's (i.e., $1^a0^b1^c$ for some $a, b, c \geq 0$), and that only the sequence 0^b is relevant for the logical AND operations. The second observation was that, node tests may be repeated several times throughout a forest, leading to duplication in the original QuickScorer data structure. Consider for instance the case of a binary feature, where the only valid test is $x_i \leq 0$, and this can be repeated hundreds or thousands of times in a large forest. Finally, similar to vectorized QuickScorer (Lucchese *et al.*, 2016b), one may use SIMD instructions to evaluate multiple documents in parallel.

Following these observations, Ye *et al.* (2018) propose a more compact representation of *nodemasks*; a merging mechanism to store and execute repeated node tests only once; and a suitable data structure to allow efficient use of SIMD instructions that operate on 256 (or larger) bit-wide registers. Together this added compactness and parallelism reduces the algorithm's memory footprint and number of operations. These improvements boost inference speed by a factor of 3.5 over QuickScorer on various datasets with up to 870 features, and for a variety of models with up to 400 leaves and 20,000 trees.

6.2 Approximate prediction by partial evaluation

In the previous section—and indeed the previous chapter—we insisted on making exact predictions and evaluating all nodes in a forest. But what if we relaxed this strict requirement and allowed the inference algorithm to produce an *approximate* prediction instead? If we are able to produce inexact scores faster but in such a way that the final quality remains unaffected, then this approximation would be acceptable.

That question motivated the work of Cambazoglu *et al.* (2010). Their work started with the observation that two properties of web search allow one to potentially short-circuit the scoring process in additive ensembles. First, that document relevance follows a skewed

distribution: for most queries, there are very few highly relevant documents, but many non-relevant documents. Second, that most users view only the first few top-ranking documents, and therefore, it may be possible to terminate the scoring of documents that are unlikely to be ranked within the top k. Given these observations, the question before the authors was whether it is possible to terminate the inference midway through the forest (i.e., without consuming every tree) and yet maintain high quality among the top-k documents.

To answer that question we need to first consider the ways in which a tree prediction algorithm scores a set of documents with respect to a query. One obvious approach is the "document-ordered traversal" (DOT) strategy where documents are scored separately by computing predictions from all trees in the forest. Alternatively, we may evaluate each tree (or *scorer*) on all documents at once, resulting in the "scorer-ordered traversal" strategy (SOT). In SOT, we must accumulate and keep track of partial scores for all documents until the entire ensemble has been evaluated. Both strategies have advantages and disadvantages in terms of memory footprint and cache friendliness. We note that, the vectorization methods of QuickScorer and RapidScorer belong to the DOT class, but where small batches of documents are evaluated with a SOT strategy.

Cambazoglu *et al.* (2010) place *exit* points at fixed positions in a given forest (e.g., every 100 trees) and introduce four *early-exit* algorithms that decide at each exit point whether the evaluation of a document can be terminated early. These algorithms are as follows:

- Early Exits Using Score Thresholds (EST): This simple approach filters documents on the basis of a pre-computed threshold and drops documents whose score does not exceed it along the way.
- Early Exits Using Capacity Thresholds (ECT): A more adaptive solution that maintains a heap with the highest scores and drops documents that do not fit in the heap.
- Early Exits Using Rank Thresholds (ERT): Similar to the EST method, except that thresholds are applied to the rank of documents.
- Early Exits Using Proximity Thresholds (EPT): Preserves the top-k documents but additionally keeps all documents whose score is within a range p from the k-th document's, where p is learnt and computed offline.

We highlight that, EST is applicable to both SOT and DOT whereas ECT applies only to the DOT method. ERT and EPT, on the other hand, operate only within the SOT scheme. Experiments showed that the EPT strategy led to the largest gains in inference efficiency (with a speed-up factor of 4×) with only a negligible loss in precision.

Cambazoglu *et al.* (2010) used statistical information from document scores and ranks to decide when to exit early. In contrast, Busolin *et al.* (2021) introduced a learnt technique,

called LEAR, that uses a classifier to predict whether a document should trigger early termination if it is unlikely to be ranked among the final top-k results. The early exit decision occurs at a *sentinel* point (i.e., after having evaluated a limited number of trees) with the partial scores determining if documents should exit. Their experimental evaluation on two public datasets shows that LEAR has a significant impact on the efficiency of the query processing with a speedup of up to 5× with a negligible loss in NDCG@10.

Separately, Lucchese *et al.* (2020b) investigate the problem of query-level early-exit strategies, where the decision to exit depends on the partial scores of all candidate documents for a query. The main finding of the work is that queries exhibit different behaviors as scores are accumulated during the traversal of the ensemble and that query-level early stopping can remarkably improve ranking quality with an overall gain of up to 7.5% in terms of NDCG@10 and a query processing speedup of up to 2.2×.

6.3 Efficient cascades

In the early-exit strategies discussed above, we relied on a partial evaluation of an ensemble to decide whether or not to exclude a document from further evaluation. That detail is analogous to the idea of multi-stage rankers, which we reviewed in Chapter 3. Indeed, early-exiting is similar to having multiple rankers that rank a set of documents sequentially and pass along to subsequent rankers the top-ranking subset.

Wang *et al.* (2011) took that idea and fused together such a cascading model using an additive ranking model. Each ranker in the model is also coupled with a *pruning* function that removes the least promising documents before passing them on to the next ranker. This ranker-pruner pair constitutes one stage in the multi-stage ranker, and documents that are kept by the pruning function keep accumulating partial scores from stages along the way until they end up in the final top-k set or are dropped in later stages.

Wang *et al.* (2011) implement an instance of such a multi-stage ranker following the principles behind AdaRank (Xu and Li, 2007), where each ranker operates on a single feature only. Interestingly, the pruners and rankers are trained jointly with a hyper-parameter that facilitates fine-tuning the balance between quality and efficiency. Their experiments show that this jointly optimized cascade model reduces the inference cost with limited impact on quality. We should note that the gains in efficiency are not as substantial as those achieved by score approximation approaches.

Culpepper *et al.* (2016) take a slightly different approach. They design a multi-stage ranker composed of binary classifiers, where the number of classifiers is equal to the number of relevance grades in the training set. They train the classifier at stage S_i to detect documents with relevance $\leq i$. When a classifier at stage S_i predicts the probability $Pr_i(d)$ for document d, then d *exits* the inference process with label i if $Pr_i(d) < t$, and otherwise

moves to the next stage. This design follows the intuition that a different number of candidates may be required for different queries and that these classifiers can help detect the subset of candidates adaptively. Experiments show that the best configuration of this cascade design can speed up inference by a factor of 2× without an adverse effect on the ranking quality.

Another advantage of the cascade design is that the computation of expensive features can be delayed to later stages, where fewer documents are evaluated. On the other hand, effective features should be used as early as possible in the cascade so that a larger number of documents can be filtered early on. That is precisely what Chen *et al.* (2017) explore in their work. Through extensive experiments, they show that a three-stage cascade with λ-MART in each stage is most effective with a cost reduction of about 50%.

6.4 Open challenges and future directions

We summarize in Table 6.1 the methods reviewed in this chapter which fall naturally into two major research directions. The first covers efficient algorithms for the traversal of decision tree ensembles. This research culminated in the QUICKSCORER and RAPIDSCORER algorithms which today are the *de facto* standard tree traversal implementation, not just for the ranking task but in regression and classification too. But while the existing implementations achieve incredible efficiency in standard computing environments, investigating the inference of complex models in embedded devices remains an open challenge, especially with the rapid rise in the use of machine learning models in resource-constrained devices. Additionally, such non-standard environments define new and unique dimensions of efficiency such as strict bounds on energy consumption among other factors.

The second line of research is an investigation of cascade models which includes early-exit strategies and *ad hoc* training of a multi-stage cascade. There remains a lot in this area that warrants further investigation. The optimal number of stages in a cascade architecture or the value of their hyper-parameters, for example, have proven difficult to determine, which, in turn, limit the applicability of cascade models. Moreover, the observed impact on efficiency is smaller than the gains from efficient traversal techniques. Despite these challenges, we believe that research into cascade models and understanding the trade-offs inherent in their design are promising directions in LtR.

Cascade models, for example, can delay heavy computation to later stages or leverage the benefits of complementary models (e.g., neural and tree models). They may also provide *stability* to an LtR system, where only a few stages may need to be re-trained or updated to improve effectiveness. We thus believe that the construction of an efficient cascade that takes into account feature computation costs, personalization, online training, or offline updates remains an exciting and potentially impactful research direction.

7 Neural Learning to Rank

In Chapter 2, we wrote about the various statistical signals which exist in queries and documents that a ranking model can use to estimate relevance. There, and indeed throughout the past chapters, we took for granted that query-document pairs are given to us in the form of vectors of k pre-computed features that in some way quantize those signals, and instead focused on learning a ranking function from them. Let us now take a step back and reconsider feature vectors.

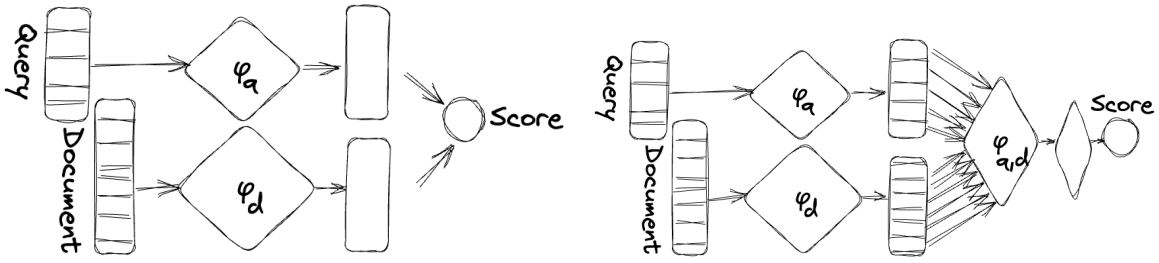
A feature vector is effectively a function that maps queries and documents to a k-dimensional space. Typically, a subset of these features can be viewed as a function of the query alone ($\phi_q : \mathcal{Q} \to \mathbb{R}^{k_q}$), another of the document alone ($\phi_d : \mathcal{D} \to \mathbb{R}^{k_d}$), and the rest form a joint function of query and document pairs ($\phi_{q,d} : \mathcal{Q} \times \mathcal{D} \to \mathbb{R}^{k_{q,d}}$). These functions map their input to a vector of real values, together making up $k = k_q + k_d + k_{q,d}$ features as the representation of a query-document pair: $(\phi_q(\cdot), \phi_d(\cdot), \phi_{q,d}(\cdot, \cdot))$.

If no feature vectors exist and all we have is the raw data, the thinking goes, we must define and build our own input-to-feature mappings. This process of defining and computing mappings from our input space to feature values is known as *feature engineering*. It is often laborious and costly, involving meticulous analysis of the data and making judgment calls on the usefulness of individual features to a machine learning model (Geng *et al.*, 2007; Gigli *et al.*, 2016). Furthermore, our model's ability to learn an effective ranking function from engineered features is tied to and bounded by their richness and discriminative power, which may be limited because we design features following our own intuition and often incomplete understanding of the problem.

Can we avoid the costs and pitfalls of feature engineering and find features that are more helpful to our model? In other words, instead of constructing them by hand, can we learn the functions $\phi_q(\cdot)$, $\phi_d(\cdot)$, and $\phi_{q,d}(\cdot, \cdot)$?

The question above hints at one of the primary reasons behind the emergence of deep learning in information retrieval. After researchers in other communities demonstrated the success of deep neural networks in learning rich representations from raw images and natural texts, it was only natural to consider their application to text ranking. The promise deep learning held for ranking was that it would obviate the need for extensive feature engineering and, instead, it would automatically learn features that give the model the necessary power to estimate relevance.

There are three major directions in the information retrieval literature that explore the role of deep models in learning a representation of queries and documents. We review these briefly in the remainder of this chapter.



(a) Representation-based **(b)** Interaction-based

Figure 7.1: Schematic illustration of pre-Transformer neural rankers. In (a), the two functions ϕ_q and ϕ_d learn representations of query and document such that relevant documents stay "closer" to the query than non-relevant ones, where closeness is measured by a vector similarity function such as cosine similarity. In (b), the representations of query and document terms *interact* with each other in $\phi_{q,d}$ and their relevance is estimated by a subsequent function.

7.1 Representation-based models

The first wave of deep learning models for ranking, aptly known as *representation-based* models, focus squarely on learning $\phi_q(\cdot)$ and $\phi_d(\cdot)$ (with $k_q = k_d$) from queries and documents such that the representations of relevant documents are "closer" than non-relevant ones to the representation of queries. Closeness is typically determined with a simple metric such as inner product or cosine similarity. We illustrate a generic version of this approach in Figure 7.1(a).

An early iteration of this idea called Deep Structure Semantic Model (DSSM) (Huang *et al.*, 2013), for example, uses feed-forward networks to learn ϕ_q and ϕ_d from character n-grams of queries and documents. Subsequent works in this space extend the same idea in different ways. Shen *et al.* (2014), for example, uses convolutional neural networks instead to capture contextual features. Dual Embedding Space Model (DESM) (Mitra *et al.*, 2016) takes as input the pre-trained word2vec (Mikolov *et al.*, 2013) representations instead of character n-grams.

7.2 Interaction-based models

What is left out of the representation-based models is the joint query-document function $\phi_{q,d}(\cdot, \cdot)$; no component of these models captures the interactions between query terms and document terms. Modeling $\phi_{q,d}$ motivated another class of neural rankers that are often known as *interaction-based* models. Specifically, as depicted in Figure 7.1(b), these models create an "interaction" matrix of the representations of query terms and document terms, often in the form of a similarity matrix. The interaction matrix then becomes an input to another function that estimates relevance. Methods in this class include DRMM (Guo

et al., 2016), KNRM (Xiong *et al.*, 2017), and ConvKNRM (Dai *et al.*, 2018).

The distinction between representation- and interaction-based methods lies not just in what features each is capable of learning which affects their effectiveness, but also in their computational efficiency. Because representation-based models learn to map documents to representations with a function ϕ_d that is distinct from ϕ_q, we can store learnt document representations to enable efficient inference. Interaction-based models, in contrast, are more expensive because we must compute $\phi_{q,d}$ during inference as its output depends jointly on the query as well as the document. Despite these differences, the two ideas are not mutually exclusive. In fact, models such as DUET (Mitra *et al.*, 2017), incorporate elements of representation- and interaction-based models to learn all three mappings ϕ_q, ϕ_d, and $\phi_{q,d}$ for effective and efficient ranking.

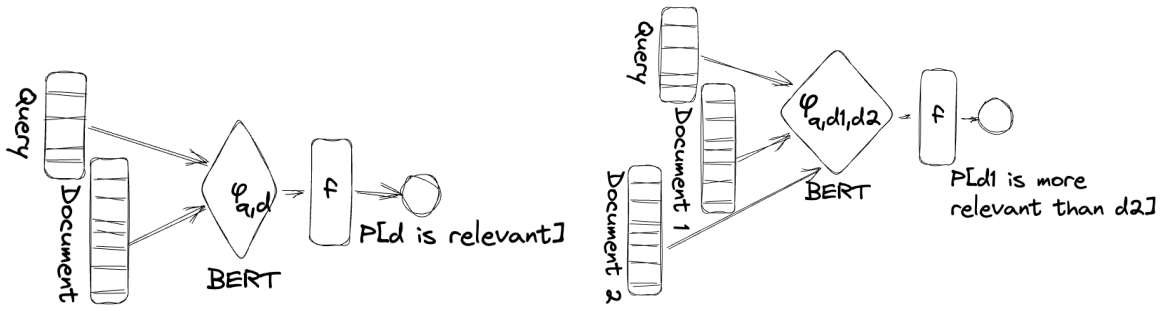
Foregoing feature engineering for representation learning also makes it possible to explore functions beyond the three mappings above. If we can model the interactions between query terms and document terms, for example, why stop there and not capture the interactions among the set of documents being ranked too? In other words, we may extract additional features as a joint function of a set of m documents $\phi_{q,\boldsymbol{d}} : \mathcal{Q} \times \mathcal{D}^m \rightarrow \mathbb{R}^{k_{q,\boldsymbol{d}}}$. Different flavors of this idea were investigated by Ai *et al.* (2019) and Pang *et al.* (2020), where the main challenge is in ensuring that $\phi_{q,\boldsymbol{d}}$ is permutation-invariant (i.e., the output of the function does not depend on the order in which documents are presented to the function).

Our brief review above only sketches an outline of ideas in the early years of neural ranking and leaves out a great deal of details. We refer the interested reader to existing surveys on representation- and interaction-based neural rankers for a more comprehensive review and analysis of these methods (Onal *et al.*, 2018; Mitra and Craswell, 2017; Guo *et al.*, 2020). But even from this outline emerges a clear picture: models grew more and more complex as we sought to enrich the representations of queries and documents. That trend continues to date, with a notable jump in model complexity when rankers based on the Attention mechanism in Transformers (Vaswani *et al.*, 2017) dwarfed many early models.[1]

7.3 Transformer-based models

That began when Nogueira and Cho (2020) reported a dramatic jump in ranking quality by applying Bidirectional Encoder Representations from Transformers (BERT) (Devlin *et al.*, 2019) to the MS MARCO (Nguyen *et al.*, 2016) passage re-ranking task, where short passages are to be ranked with respect to a text query. Their model was later named "monoBERT." In the language of our discussion here, the BERT component in monoBERT serves as the joint function $\phi_{q,d}$, producing a representation for a query-document pair.

[1] We often refer to neural rankers that are based on the Attention mechanism as "Transformer-based" rankers. However, we recognize that "Transformer" implies an encoder *and* a generative decoder neural module, with the latter playing no role in the vast majority of retrieval and ranking systems.



(a) monoBERT **(b)** duoBERT

Figure 7.2: Illustration of the (a) monoBERT and (b) duoBERT rankers. MonoBERT predicts a relevance score of a single query-document pair—a familiar scoring machinery in LtR. DuoBERT takes a pair of documents and predicts whether the first document is more relevant to the query than the second document. Note that, the scoring function itself takes two documents as input, in contrast to *pairwise* LtR where the *loss* function takes the scores of two documents.

From that, a simple feed-forward network learns to estimate relevance by optimizing a pointwise loss function. This is illustrated in Figure 7.2(a). The remarkable effectiveness of this network architecture generated much excitement in the community and led to a flurry of research activity.

Many subsequent works (Akkalyoncu Yilmaz *et al.*, 2019; Li *et al.*, 2020; Dai and Callan, 2019; MacAvaney *et al.*, 2019) seek to address monoBERT's limitation on input size (capped at 512 tokens), which was enough to rank short passages but not sufficiently large to apply to long documents. Others (Nogueira *et al.*, 2019a) extended the model to learn representations for pairs of documents; that is, $\phi_{q,\boldsymbol{d}}$ where $\boldsymbol{d}$ is a set of two documents. The resulting model is known as "duoBERT" and is shown in Figure 7.2(b). Yet others (Nogueira *et al.*, 2020; Pradeep *et al.*, 2021) go beyond BERT and use a sequence-to-sequence model such as Text-to-Text Transfer Transformer (T5) (Raffel *et al.*, 2020) for ranking. We refer the interested reader to a recent survey by Lin *et al.* (2021) on Transformer-based rankers for a detailed discussion of each method and their many existing variants.

As is often the case, this march from basic feed-forward networks to gargantuan stacks of Transformer-based neural modules with millions of parameters has made stunning improvements in quality possible only at the expense of training and inference efficiency. Better accuracy through ever-increasing complexity once again presents a new but familiar challenge: How do we balance the two competing objectives of efficiency and effectiveness? This question has gained even more significance due to the sheer scale of neural rankers and the the multitude of additional efficiency dimensions they introduced, such as sample- and energy-efficiency. Additionally, scaling these models to long documents (as opposed to short "passages") introduces another efficiency challenge that is largely unique to neural

rankers (Lin *et al.*, 2021).

In the next chapter, we will review some of the ideas that explore this trade-off in the context of neural rankers, many of which will, unsurprisingly, look familiar to the reader, just as the question above did. So as we present our summary of each class of methods, we highlight their connection to the first half of this manuscript.

8 Efficiency in Neural Learning to Rank

We have argued in this monograph that neural rankers are just another instantiation of the general LtR framework, where the hypothesis class is the set of deep neural networks. It is therefore not surprising that the ideas that were introduced in previous chapters to manage inefficiency in decision forest models carry over to neural LtR at a high level.

This portability of ideas is easy to see in the case of the *multi-stage* and *cascade* architecture of retrieval and ranking because their general setup is agnostic to the specific choice of models in each stage. In other words, by trimming the candidate list in the first stage (or first few stages), we can reduce the volume of candidates that must be re-ranked by an expensive, neural ranker, thereby improving the inference and training efficiency of the end-to-end ranking system.

This general method was first investigated by Nogueira *et al.* (2019a) in the context of neural rankers. Nogueira *et al.* (2019a) observe that duoBERT, which learns to score pairs of documents jointly as explained in Chapter 7, is more effective than monoBERT but at a much higher inference cost. That is because, given k candidates, duoBERT performs inference on $k(k-1)$ pairs of documents d_i and d_j to estimate the probability $p_{i,j}$ of d_i being more relevant than the other. It then aggregates the probabilities $p_{i,*}$ using one of the many proposed aggregation functions (e.g., sum) to arrive at a single relevance score for individual documents. It is clear that duoBERT, due to its more computationally intensive inference, would fare better for smaller values of k relative to monoBERT.

That prompted the authors to consider a multi-stage ranking pipeline illustrated in Figure 8.1 where candidates, generated by BM25, are first ranked by monoBERT and only then the top candidates are rearranged by a duoBERT ranker. Nogueira *et al.* (2019a) then explore the trade-offs such a setup offers between ranking quality and inference latency, by studying the interplay between quality and the number of candidates retrieved with BM25, along with the number of candidates passed from monoBERT to duoBERT.

Among the many interesting observations, they found that providing a larger pool of candidates to monoBERT helps ranking quality—up to a point, beyond which we see diminishing returns. That indicates that documents with a relatively low BM25 score can indeed be relevant to the query and be placed at higher ranks with monoBERT. Interestingly, it is often enough to apply duoBERT to a handful of top documents ranked by monoBERT

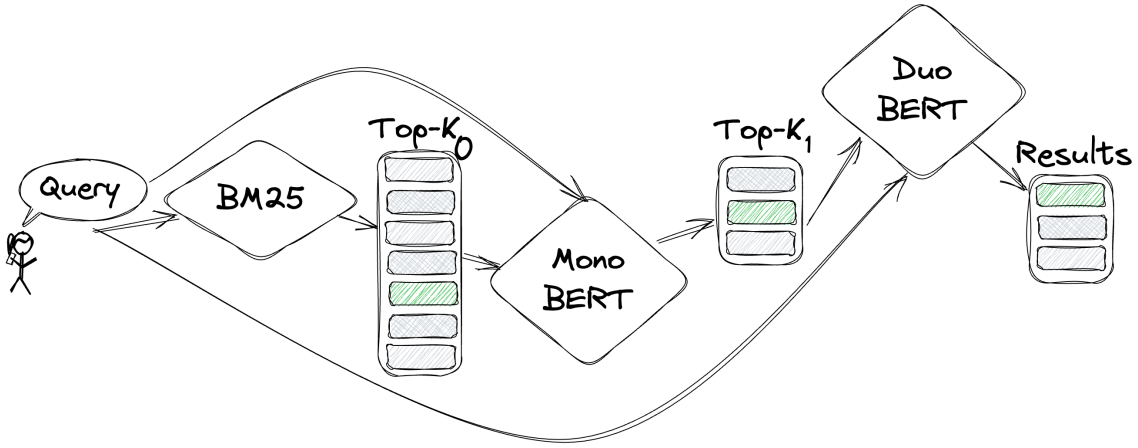


Figure 8.1: Multi-stage search pipeline of Nogueira *et al.* (2019a) consisting of BM25, monoBERT, and duoBERT.

to obtain the highest achievable ranking quality.

In fact, this last point turns into a rather surprising phenomenon in a follow-up study. Pradeep *et al.* (2021) extend the multi-stage pipeline of Figure 8.1 and add one more stage right before BM25: The authors use doc2query-T5 (Nogueira and Lin, 2019), a *generative* model, to expand documents with *predicted* queries, prior to the BM25 stage. They also replace BERT with an adaptation of T5 (Raffel *et al.*, 2020), a sequence-to-sequence model, to the ranking task. They find that, for some but not all monoT5's aggregation functions, passing more candidates between monoT5 and duoT5 re-rankers leads to a drop in quality. Pradeep *et al.* (2021) do not articulate if the quality degradation is statistically significant, nor do they explain why they observe this rather counter-intuitive behavior. It is therefore unclear if this points to a weakness of the duoT5 model itself, or the aggregation functions used to produce relevance scores from probabilities.

Some small details aside, the general observations reported in these works are consistent with the prior literature on multi-stage ranking systems which attests to the robustness of this general and rather intuitive design. In fact, the idea is so natural that others have also investigated similar setups in different contexts, e.g., (Matsubara *et al.*, 2020; Zhang *et al.*, 2021).

In the remainder of this chapter, we summarize other solutions for efficiency in neural LtR whose connection to the earlier literature is less obvious, and discuss other new problems. We revisit, for example, *early-exit strategies* and show how this simple technique can be baked into a neural LtR model. We show how *distillation* can be used to find a small, more efficient model given a large, more effective ranker. Finally, we review the literature on dense or semantic retrieval and describe the challenges this new problem introduces.

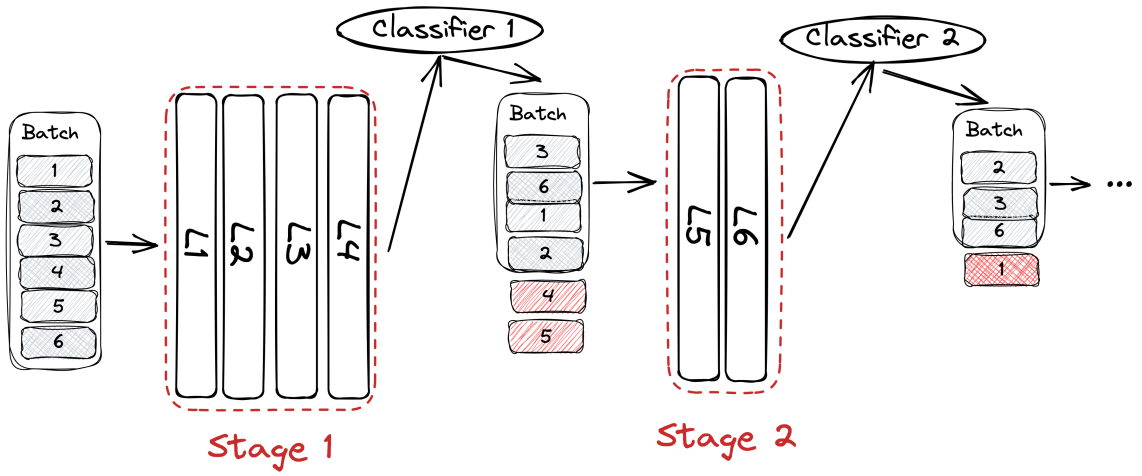


Figure 8.2: Illustration of the Cascade Transformer (Soldaini and Moschitti, 2020). Every group of Transformer layers forms a stage of the cascade and is followed by a classifier, whose output is used to rank candidates in a batch. Between consecutive stages, the model discards $\alpha\%$ of candidates.

8.1 Early exit strategies

What motivated Cambazoglu *et al.* (2010) to only partially evaluate a gradient-boosted decision forest for some documents, thereby exiting the inference algorithm early, was the hypothesis that trees that come later in the forest are there to refine the ranking among the top candidates; the scores of the vast majority of candidates, especially those that are obviously non-relevant, should not change dramatically after the evaluation of the first few trees. So using some form of thresholding, we can exit early and prevent certain candidates from going through an entire forest.

In effect, a gradient-boosted forest can itself be understood as a cascade ranker where each stage is a decision tree. We can therefore trim the candidate list every few stages in the cascade and progressively reduce the cost of inference for any query.

Interestingly, one can view a stack of Transformers much the same way: Each Transformer layer in a multi-layered model such as BERT is akin to a stage in a cascade ranking system! As a candidate list bubbles up the stack of Transformers, analogous to the decision forest scenario, the score of non-relevant documents should change less and less substantially and their position in the ranked list should move up or down less dramatically. As such, it may be possible to exit inference early for a subset of candidates and avoid evaluating the full model on the entire candidate list, all without noticeable impact on effectiveness.

That is, in fact, one major idea that has emerged in the neural information retrieval literature to improve inference efficiency. In the context of an Answer Sentence Selection task for Question Answering, for example, Soldaini and Moschitti (2020) apply that idea to

the multi-layered Transformer model—which they call Monolithic Transformer—to obtain the Cascade Transformer. In particular, they intersperse lightweight classifiers between every few layers of the model, so that layers 4, 6, 8, 10, and 12 become exit points in the cascade. This is illustrated in Figure 8.2.

Conceptually, one can think of each classifier as predicting whether a document should continue to be evaluated or whether it can safely exit the inference altogether. The concrete logic is quite straightforward: Given the predictions of one classifier on a batch of candidates, the cascade ranks the candidates according to their scores and discard $\alpha\%$ of them, before passing the rest onto the next stage in the cascade. This logic repeats until the very last classifier, which produces the final predictions for the remaining candidates. For example, if there are 128 documents in the initial batch and $\alpha = 30\%$, the first exit point drops 38 documents, with the remaining 90 documents moving onto the next exit point. The authors justify this trimming strategy by noting that, discarding a fixed number of items from a batch allows the cascade to know the batch size in each stage *a priori*—a desirable behavior in today's neural network inference engines. Were they to discard candidates by a fixed score or rank threshold as Cambazoglu *et al.* (2010) did, in contrast, batch sizes at each stage of the cascade would become dynamic, possibly leading to sub-optimal throughput.

Another notable difference with earlier work in the context of decision forests is that the model and classifiers can be learnt jointly in an end-to-end manner: At each training iteration, the training procedure selects one stage in the pipeline uniformly at random, computes the value of the loss function, and back-propagates the error throughout earlier stages. This training schedule, the authors claim, makes the later stages of the cascade more robust to noise that stems from high variance in earlier stages.

Soldaini and Moschitti (2020) evaluate the Cascade Transformer in terms of quality and inference cost reduction with respect to a Monolithic Transformer on a number of Question Answering datasets. The results present no surprises: Choosing a larger value of α and discarding candidates more aggressively between stages of the cascade degrades end effectiveness but leads to a larger inference cost reduction (-37% when $\alpha = 30\%$).

While the Cascade Transformer was tailored to improving *throughput*—hence the emphasis on fixed batch sizes throughout the cascade—other works target the inference *latency* (i.e., batch size of 1) using very similar ideas. For example, Xin *et al.* (2020) explored a design that inserts a classifier after every layer. At each exit point (called "off-ramp" in their work), its classifier is evaluated and, depending on the confidence in the classifier's prediction, inference terminates early or the sample goes on to be evaluated by later layers. In a follow-on study, Xin *et al.* (2021) present an evolution of this idea where, instead of deciding to exit early based on the classifiers' confidence, the decision to exit or not is itself learnt at every exit point.

These and other contemporary works on early-exit strategies in Transformer-based models (Liu *et al.*, 2020; Schwartz *et al.*, 2020) are a natural application of the idea of

partial evaluation of Cambazoglu *et al.* (2010) to neural rankers, often resulting in similar trade-offs between effectiveness and efficiency.

8.2 Knowledge distillation and neural compression

We introduced knowledge distillation in Chapter 5, where the idea was to train a small and efficient "student" model that attempts to imitate the behavior and predictions of a larger and effective "teacher" model. The intuition was that, because the teacher model has a greater complexity, it captures nontrivial relationships in the data and cleans up noisy input. In this way, the student model learns from a "clean" version of the data and can focus on learning what is more germane to the task. In that chapter, we also reviewed examples of what we argued may be seen as knowledge distillation in the decision tree-based LtR literature. This included X-Dart, CLEaVER, and X-CLEaVER, which arrive at a student model by *compressing* an existing tree ensemble.

Compression is easy to translate to the world of neural networks: Individual connections or entire layers can be removed, reset in a network if their existence is not "salient" according to some definition of salience, or quantized. The resulting network may become leaner in size or its weight matrix sparser, leading to different and often more efficient computational patterns. We refer the reader to existing surveys (Xu and McAuley, 2022) that cover this growing literature in the context of large language models for a detailed discussion of these methods.

Knowledge distillation (Ba and Caruana, 2014) in the deep learning literature works not by pruning a large model, but by learning a new, more compact model instead. Perhaps the most relevant and pivotal studies from the knowledge distillation literature are TinyBERT (Jiao *et al.*, 2020) and DistilBERT (Sanh *et al.*, 2020). While the specifics of these two works are different, the general idea is rather similar: Given a large pre-trained BERT language model, the two studies explore strategies to train a more compact student model that uses the same Transformer architecture but has fewer parameters. In TinyBERT, for example, the student model learns to fit the individual weight matrices from the attention modules of a large BERT model, with the intuition that linguistic knowledge can be transferred to the student in this way. Additionally, the output of Transformer layers, the embedding layers, and the predictions of the model too become objectives for the student model to attain. As one would expect, distillation leads to models that are several factors faster and lighter than the large teacher models, with little to no loss in effectiveness.

In the context of LtR, we have already called out three works (Tang and Wang, 2018; Cohen *et al.*, 2018; Nardini *et al.*, 2022) in Chapter 5 that demonstrate the usefulness of knowledge distillation for balancing efficiency and effectiveness of ranking models.

Building on that foundation, Gao *et al.* (2020) explore how *ranking-specific* knowledge can be transferred between a teacher and a student BERT-based model. Gao *et al.* (2020) study several distillation strategies. In one, dubbed "Ranker Distill," they train monoBERT over the MS MARCO (Nguyen *et al.*, 2016) dataset, then randomly initialize a smaller student model and ask it to reproduce the teacher model's ranking behavior. In "LM Distill + Fine-tuning", they transfer knowledge from a pre-trained, general-purpose BERT model (but not monoBERT) to the student model, and only then fine-tune the student model for ranking. Finally, in a hybrid of the two methods, they have the student model learn from a general-purpose BERT model first, followed by distillation of ranking behavior from mono-BERT.

Of the three distillation strategies, Gao *et al.* (2020) find that the hybrid approach can produce a student ranking model that is just as effective as monoBERT but that is up to 9 times faster during inference. This is an important finding that goes to suggest that large models may be over-parameterized and that knowledge distillation can substantially reduce inference cost in exchange for additional training of a more compact, student model.

One can even take the idea of distillation a bit further and distill the collective knowledge of an *ensemble* of teacher models into a compact, student model. This is the idea Zhuang *et al.* (2021) studied for ranking, which they claim leads to a model that is more efficient during inference and that displays more stable predictions.

As the authors articulate, one can either have the student predict an *aggregated*, ensemble-level teacher label, or solve a multi-objective optimization problem by predicting all model-level labels simultaneously. In either case, we need to define what the model- or ensemble-level labels are. Zhuang *et al.* (2021) experiment with two model-level labels: (1) the raw score of rankers in the ensemble; and, (2) the reciprocal rank (i.e., $1/(\alpha + \pi_i)$, where α is a constant and π_i is the rank of document i). They define the ensemble-level label as the mean of model-level labels.

Their experiments on the MS MARCO passage ranking dataset appears to corroborate the authors' hypothesis that distilling knowledge from multiple teachers can indeed produce a high-quality student model. It turns out that optimizing a single loss defined on ensemble-level labels is just as good as a multi-objective formulation; and that, using raw scores versus reciprocal ranks makes little difference in the end.

While the idea in this article is interesting in and of itself, it is unclear how generalizable the empirical findings are—as the authors themselves point out too. Perhaps what gives one pause is that the student model is itself a BERT ranker and that the individual rankers in the teacher ensemble are also BERT rankers that are all trained in the same manner but starting out from different initial weights. The authors justify this by suggesting that evaluating a single BERT ranker is cheaper than executing an ensemble of BERT rankers. That may be true, but that small detail may explain the observed insensitivity to the choice of labels and the distillation strategy. More importantly, one wonders if, in this instance,

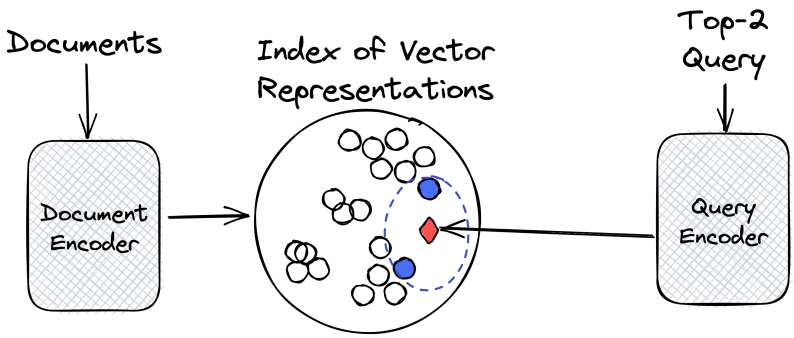


Figure 8.3: The most basic form of dense retrieval produces vector representations (circles) for a collection of documents offline. During inference, the query is similarly transformed into a vector (red rhombus) in the same representational space. An approximate nearest neighbor algorithm then finds the closest document vectors to the query vector (blue circles). By training the encoders in the right way, we can structure the representational space such that closeness in the space implies semantic similarity.

knowledge distillation is not simply related to variance reduction as the student model learns the average of predictions of teacher models.

8.3 Dense retrieval

One of the major innovations in neural LtR research was the evolution of "cross-encoders" such as monoBERT and duoBERT to two-tower or "bi-encoder" models. Instead of learning a parameterized function (like BERT) that takes a query-document pair (or multiple documents) as input and simultaneously learns representations (i.e., features) and predicts their relevance, bi-encoders disentangle the relevance prediction function from representation learning, and simplify the former as much as possible. The idea is to push much of the complex, time-consuming inference operations offline and thereby speed up query processing.

In its most basic form, this design resembles the representation-based neural ranker architecture of Figure 7.1(a), but where ϕ_d and ϕ_q are pre-trained large language models that may be further fine-tuned. Because the representation of documents is independent of queries, we may store the document vectors in an offline index and, during retrieval, compute the representation of the query and find its closest document vectors. This paradigm is often referred to as "dense retrieval" or "semantic search," which is typically solved using a k nearest neighbor (NN) search or k approximate nearest neighbor (ANN) search algorithm. We illustrate this procedure in Figure 8.3.

The feasibility of this basic idea was demonstrated by several works (Zhan *et al.*, 2020; Karpukhin *et al.*, 2020; Ma *et al.*, 2021a; Qu *et al.*, 2021). Karpukhin *et al.* (2020) present what they call Dense Passage Retrieval (DPR) and show that dense vector representations

can indeed be used to perform the retrieval task following the recipe above. Interestingly, effective representations can be learnt from a small number of questions and passages (in a question-answering task) by a simple bi-encoder framework. They evaluate DPR on a wide range of open-domain QA datasets, with the results showing that DPR outperforms a strong BM25 system by 9%–19% absolute points in terms of top-20 passage retrieval accuracy.

In the general dense retrieval framework of Figure 8.3, there are two key factors that contribute to the overall efficiency and effectiveness: the ANN algorithm itself that performs the search over millions or billions of vectors, and, as we will discuss, the nature and quality of the learnt vector representations. In the rest of this section, we pay particular attention to the latter and review a subset of methods that encode queries and documents for dense retrieval.[1]

Existing encoding models fall into one of two categories based on the granularity of the representations they produce: per-document encoders (*single-vector*) and per-term encoders (*multi-vector*). The former class, which includes DPR, learns a single vector representation for a document, whereas the latter produces a contextualized representation for every term in a document.

Single-vector dense retrieval

Intuitively, for a single-vector dense retrieval method to work well, the structure of the representational space must tightly preserve the semantic similarity between pieces of text. In other words, vectors that are close to each other according to some vector distance function, should be semantically similar to each other and vice versa.

A key factor that contributes to the quality of representations and thus the structure of the space is the quality of the negative examples used to train an encoder. This is a challenge because, unlike in standard LtR where the assumption is that a short list of documents with relevant and possibly non-relevant documents exist, in dense retrieval the set of possibly non-relevant documents is extremely large and diverse; everything in a collection minus the very few positive documents is non-relevant to any arbitrary query. Choosing which documents to present to the model as negative examples can have a profound effect on the effectiveness of the final model and its representational space.

In DPR (Karpukhin *et al.*, 2020), for example, negative documents are selected using three different strategies: randomly; by retrieving top-k documents with BM25 and selecting those that do not contain the answer but have significant lexical overlap with the query; or, positive documents for other queries that appear in the same training batch.

[1] We reiterate that our monograph is primarily focused on *ranking*. While recent works on dense retrieval blur the line between retrieval and ranking, we still believe that a deeper discussion of dense retrieval and efficiency and effectiveness trade-offs in that literature is beyond the scope of this monograph.

ANCE (Xiong *et al.*, 2021), which like DPR is a single-vector encoder, continually finds negative examples based on the current structure of the vector space. The way it realizes this idea is by asynchronously updating an ANN index *while* the dense retrieval model is being trained, and retrieving the top-k set for training queries from this index. Every document that appears in the top-k set that is not a positive example is deemed a "hard" negative and used to further train the model. This training procedures continues until convergence.

The authors show that ANCE outperforms other competitive dense and sparse retrievers, with substantial margins in long document retrieval task. Perhaps more importantly, by measuring the success rate of an ANN algorithm in finding relevant documents, they claim that sampling negative examples as done in ANCE improves the quality of the final vector representations. In fact, an ANCE-based dense retrieval model approaches the effectiveness of a cross-encoder, BERT re-ranker in a multi-stage setup. These observations, Xiong *et al.* (2021) state, cast doubt on "a previously-held belief that modeling term-level interactions is necessary in search." Comparing their inference with a BERT re-ranker, they conclude that ANCE brings about a speedup of 100× due to higher quality representations. While some of these claims are not entirely supported and the speedup may be exaggerated, the fact is that sampling negative examples in this way appears to be more effective than previous strategies.

While the relative inference efficiency and effectiveness of ANCE is rather impressive, its training is resource-intensive and time-consuming. This is because an ANN index must be updated with the latest representations and negative examples must be retrieved from this index periodically. To lessen the training cost, Lindgren *et al.* (2021) suggest to maintain a cache of possibly-stale negative examples. Caching, as shown in experiments, allows the training procedure to work more efficiently and scale to a larger pool of negative examples with a lower memory and computational footprint. It turns out, as the authors show in their theoretical analysis, updating a very small fraction of the cache at each iteration ensures fast convergence.

Up to this point, the marriage of ANN search algorithms and learnt vectors from general-purpose language models and encoders led to much success in retrieval quality. Gao and Callan (2021a), however, question the suitability of the latter: Are general-purpose language models optimal for dense retrieval?

Gao and Callan (2021a) argue that existing language models are suboptimal and inefficient because of the way they aggregate and condense information into a single vector, which is designed for tasks (such as next sentence prediction) that are removed from the objectives of dense retrieval. This is because, how information is aggregated and what signals are encoded are determined by how the "attention" mechanism works. In particular, the representation of the CLS token—a token that is prepended to a sequence and whose representation is typically taken as the final vector representation of a query or document—

has weak interactions with other document terms in early layers, and uses too broad of an attention span over document terms in later layers.

Having made this observation, Gao and Callan (2021a) tailor the Transformer architecture in a model they call Condenser by customizing how it *attends* to document terms. Notably, the "head" of the model takes the CLS representation from the later layers as in a Transformer but, additionally, takes *term representations* from early layers. The authors experimentally evaluate their model on two public datasets and show that the use of Condenser improves over standard language models by large margins on several text retrieval and similarity tasks.

In a follow-up study, Gao and Callan (2021b) argue further that for the CLS representation to be effective, it must be transformed by a head, a typically non-linear function. This is different from what takes place in ANN search, where we simply find the closest neighbors with respect to dot product (or other simple distances). There is therefore a disconnect between how representations are generated and how they are used to perform dense retrieval. They then propose augmenting the pre-training loss with an unsupervised corpus-level contrastive loss to "warm start" the embedding space; in effect, the pre-training stage asks the model to learn that similar passages (spans within a document) should have closer representations and dissimilar passages (spans from different documents) should instead be positioned farther. They show the effectiveness of this regime and, as a side-effect, show that it renders unnecessary heavy data engineering efforts such as augmentation, synthesis, and filtering.

The single-vector dense retrieval literature is vast and growing still with many other works that straddle the literature on the ANN as well as the modeling pieces. We do highlight the works of Zhan *et al.* (2021) and Zhan *et al.* (2022) who investigate a joint optimization of vector representation learning *and* the construction of the ANN index for more efficient and effective overall retrieval and a more compact index.

Multi-vector dense retrieval

The methods we reviewed so far learn a single vector representation for a document. Khattab and Zaharia (2020) question whether a simple ANN search over single-vector representations is sufficient to ensure quality and whether relevance estimation would improve by replacing the simple distance function (e.g., dot product) with a more complex function.

ColBERT (Khattab and Zaharia, 2020; Santhanam *et al.*, 2022a) approaches the encoding and similarity estimation problems in two steps: A first step uses a language model (e.g., BERT) to encode query terms and document terms (separately). The output of this step is a sequence of vectors representing terms in a query or document. A subsequent step uses a "late-interaction" function to estimate the overall similarity of query and document terms. This function may be the norm of a matrix whose entry at row i and column j is the inner

product of the query's i^{th} vector and a document's j^{th} vector.

As a result of this two-step process, ColBERT can leverage the expressiveness of deep language models and, at the same time, enable us to pre-compute document representations offline. The authors comprehensively evaluate ColBERT using two passage search datasets (MS MARCO Ranking and TREC Complex Answer Retrieval) and show that it is more effective than non-BERT baselines and competitive with existing BERT-based models, but that ColBERT is two orders of magnitude faster and requires up to four orders of magnitude fewer FLOPs per query.

It should come as no surprise that ColBERT's higher effectiveness comes at a cost. One new challenge is the inflated size of the index: rather than storing a single vector for every document in our index, we must now find room for term-level vector representations. Santhanam *et al.* (2022a) offer a solution to remedy this particular cost by reducing the overall memory footprint of the representations.

Another added cost that pits multi-vector representations against single-vector encoders is that we can no longer perform retrieval in a single step by using an exiting ANN search algorithm: With single-vector representations, once documents are encoded, all we must do to retrieve the top-k documents is to ask for the k nearest neighbors from an ANN index. Tonellotto and Macdonald (2021) proposed to rank terms by their importance and compute the similarity score for query-document pairs only using a subset of query terms instead. Lin *et al.* (2020a) ask, instead, if knowledge distillation makes it possible to learn a model that is just as effective as ColBERT but offers a single-step search like earlier works. With the intuition that tight coupling between the teacher and student models may enable more flexible distillation strategies that yield better representations, the authors show that their distilled model, called TCT-ColBERT, does indeed improve query latency and greatly reduces memory usage with a limited reduction in effectiveness relative to ColBERT.

8.4 Open challenges and future directions

Table 8.1 summarizes the methods we reviewed in this chapter. They naturally fall into three major research directions. The first covers early exit strategies to speed up the inference of Transformer-based neural rankers. We discussed the connection to the literature on tree-based LtR, where early-exit strategies were applied successfully to ensembles of regression trees. With Transformer-based rankers, the approach is similar to tree ensembles: A stack of Transformer layers is equipped with classifiers, placed at different points of the network. These classifiers are in charge of deciding when to stop the inference of a given document. Several works contributed to this direction by proposing how to position the different classifiers and how to decide when to stop the inference.

The second line of research investigates the use of knowledge distillation for ranking,

where we observe two main classes of ideas. The first focuses on applying knowledge distillation to ensemble of regression trees to distill their "knowledge" into a small, more efficient neural networks. The second concerns Transformer-based rankers and attempts to derive networks that are faster during inference without loss in accuracy.

The third category is the literature on dense retrieval methods that concern the efficiency-effectiveness trade-offs with Transformer-based networks. Most proposals use a pre-trained language model to learn representations of documents that can be pre-computed and quickly searched through at query processing time with fast similarity operations. We touched on two main approaches in the literature, single-vector vs. multiple-vector representations, and reviewed how they induce specific time-space trade-offs involving approximate nearest neighbors search.

We note that the dense retrieval literature is still evolving rapidly with new innovative methods being developed actively to learn higher-quality representations and to search for approximate nearest neighbors more efficiently and effectively. As we stated earlier, we believe dense retrieval and the topic of efficiency and effectiveness trade-offs in this specialty deserves its own, more comprehensive survey than what we delivered in our ranking-focused monograph. We therefore refer the reader to a recent survey by Zhao *et al.* (2022) on this topic for a complete treatment.

9 Discussion and Open Challenges

The preceding chapters offered a review of LtR and the many ideas put forward in the literature to understand the efficiency and effectiveness aspects of LtR methods. We reviewed tree-based methods separately from neural network-based methods, but showed how some of the ideas carry from one area to the other. In this chapter, we conclude our monograph by looking ahead and identifying the problems within this space that we anticipate will require significant attention from and research by the community in the coming years.

9.1 Stochastic cascades

Conventionally, ranking functions are deterministic: given a query-document pair, the output of an LtR function is a score that captures the relevance between the input query and document. By sorting candidates by this relevance score, we obtain a final, unique ranked list. It turns out that one may view the set of relevance scores for a list of candidates together as defining a *distribution* from which a ranked list may be *sampled*. This stochastic view of ranking scores, first proposed by Bruch *et al.* (2020), has proven to be a principled perspective and has already led to a flurry of research and many innovations (Oosterhuis, 2021; Diaz *et al.*, 2020; Zamani *et al.*, 2022) due to its flexibility and theoretical properties.

One notable application of this idea that is relevant to the discussion on efficiency and effectiveness is the work of Zamani *et al.* (2022) where the authors take the cascade architecture introduced in Chapter 6 and theoretically analyze the connection between the first-stage retrieval and a second-stage ranker. In particular, by viewing retrieval and subsequent ranking as a stochastic process, they show that, contrary to conventional wisdom, it is not enough for the first-stage retrieval to return a candidate list that maximizes recall. Instead, the retrieved set must maximize *precision*. One implication of this analysis is that retrieving the same number of candidates for all queries in a cascade architecture is not appropriate, and, in fact, individual queries may require a shorter or a longer list of candidates.

The conclusions of Zamani *et al.* (2022) are reminiscent of the work by Wang *et al.* (2011) and help reaffirm the idea of a simultaneous ranking and *pruning* of the candidate list in each stage of the cascade. But more importantly, their work lays the foundation for a more principled construction of cascade ranking models where its end-to-end efficiency and effectiveness may be modeled and optimized. Is it, for example, feasible to construct a cascade system with improved efficiency (by way of pruning candidate lists between stages) *and* enhanced quality (by maximizing precision in early stages)? Can we learn the parameters of such a cascade efficiently? As we stated in our concluding remarks in Chapter 6, we believe an exploration of this question to be important and consequential for efficiency and effectiveness in retrieval and ranking systems.

Going one step further, we ask what implications, if any, this stochastic view of ranking systems has for cascade-like rankers such as decision forests and layered Transformer models. While we often place early exit methods in a category separate from post-hoc pruning algorithms (of nodes, trees, or neural connections), can we unify these methods instead by casting ranking as a stochastic process? If so, what opportunities does such a unified framework bring about insofar as the trade-offs between efficiency and effectiveness? These are open questions that we believe can help shape the future of this topic.

9.2 Retrieval of hybrid vectors

Throughout this monograph, we emphasized the role of cascade architectures in enabling efficient and effective ranking systems. But one thread that has emerged in recent years is whether it is feasible for a cascade ranking system to collapse into a single stage. Can we achieve effectiveness and efficiency (in all its senses) by applying a single function to an entire collection of documents and directly obtain a ranked list? Indeed, this is one of the motivating factors behind the research on "dense retrieval" methods.

As explained earlier, in most dense retrieval methods, we project documents into a vector space where each coordinate is dense (i.e., every coordinate is almost surely non-zero) to

obtain a vector representation (or "embedding"). During inference, queries too are projected into the same vector space. Finding a ranked list of documents that are the most relevant to a query is then equivalent to finding the document vectors that are closest to the query vector. This problem can often be solved efficiently using an Approximate Nearest Neighbor Search algorithm such as FAISS (Johnson *et al.*, 2021) or Hierarchical Navigable Small World Graphs (Malkov and Yashunin, 2016).

While dense retrieval methods produce high-quality ranked lists, they are typically much more inefficient than their inverted index-based counterparts such as BM25. This observation has led researchers to explore *sparse representations*. The crux of the idea is to learn *sparse* representations in a space that has as many dimensions as there are terms in the vocabulary, where each coordinate encodes the "importance" of the corresponding term in the context of a query or document (MacAvaney *et al.*, 2020c). By regularizing the model to encourage sparsity in its output, we can create vector representations that have very few non-zero coordinates relative to the total number of dimensions. Given this sparsity, the thinking goes, we may leverage traditional inverted index-based algorithms for efficient retrieval. Examples of this research include the works of (Lassance and Clinchant, 2022; Formal *et al.*, 2022; Zhuang and Zuccon, 2021; Zhuang and Zuccon, 2022) among others.

While retrieval over learnt sparse representations is often more efficient than dense retrieval, and there is ongoing research on making "sparse retrieval" algorithms more efficient (Mallia *et al.*, 2022), many challenges still remain. For example, if certain coordinates of document vectors are non-zero for a large portion of the collection, the retrieval algorithm will need to visit more documents to obtain the top-k candidates, thereby creating scalability and efficiency issues. Given that existing retrieval algorithms such as (Broder *et al.*, 2003) and its variants, generally assume that queries are much shorter than documents, we face similar scalability and efficiency challenges if a query has a large number of non-zero coordinates in its sparse representation. More research is therefore needed in developing data structures and algorithms that can operate over sparse representations.

Furthermore, there is increasing evidence that a hybrid retrieval framework—where we *fuse* dense and sparse retrieval to obtain a final candidate list—brings about substantial gains in retrieval and ranking quality (Thakur *et al.*, 2021; Luan *et al.*, 2021; Wang *et al.*, 2021b; Chen *et al.*, 2022; Bruch *et al.*, 2022a). While existing studies only consider BM25 for the sparse (also known as "lexical" part), it is in theory possible to extend hybrid retrieval to learnt dense and sparse representations, resulting in hybrid vectors for queries and documents. In fact, as explained in the previous paragraph, learnt sparse representations can themselves be dense in some subspace, thereby taking on a hybrid form in practice. It is as yet unclear how this joint retrieval problem should be addressed and what trade-offs exist in this regime. We believe these research questions to be important to the discussion on efficiency and effectiveness.

9.3 A multi-faceted view of efficiency

Efficiency has historically been taken to mean space- or time-efficiency, primarily in the context of inference. But we should not forget the other factors that contribute to the overall efficiency of a system. For instance, Scells *et al.* (2022) show through an extensive comparison of a range of models from bag-of-words to decision trees to large language model-based rankers, that complex neural models are unsurprisingly energy-hungry, especially during training. This increased energy consumption coupled with the need for larger and larger datasets present new challenges to retrieval and LtR, especially considering the environmental impact of this research.

These new challenges underline the importance of broadening the definition of efficiency to encompass not just time- and space-efficiency as before, but also other related facets such as *sample*-efficiency (i.e., the amount of data required to train an effective model), *resource*-efficiency (e.g., the amount of computational resources needed to train a model), and *energy*-efficiency (i.e., the emissions produced during the course of model training).

This expansion requires the development of formal definitions and standardized metrics for measuring and reporting the efficiency of a retrieval and ranking system. To that end, research is needed to design efficiency-oriented evaluation protocols and guidelines that can help researchers assess the merits of an approach and better understand the trade-offs between various methods. For example, if a work improves efficiency in certain dimensions, but not others, all at the cost of effectiveness, how should we evaluate and interpret the empirical results. This additionally highlights the importance of developing appropriate benchmark datasets. We believe these research questions to be instrumental to the future of efficiency within neural retrieval and ranking.

9.4 Designing multidimensional leaderboards

Existing leaderboards and open challenges in information retrieval that draw much attention and competition from the research community have historically been centered on measures of quality or effectiveness. For example, the MS MARCO (Nguyen *et al.*, 2016) leaderboard orders submitted systems for its various tasks in decreasing order of ranking quality such as MRR@10.

These leaderboards have demonstrably contributed to the progress we have witnessed over the years: MRR@10 for the MS MARCO passage retrieval task, as a representative example, has remarkably gained over 24 points since its debut! But as Santhanam *et al.* (2022b) argue, the emphasis on quality hides the fact that some ranked lists are much more expensive to obtain than others. The authors show this by conducting a *post-hoc* comparison of published works as well as an in-depth cost analysis of representative methods (BM25, Dense Passage Retrieval, SPLADE, and ColBERTv2) to arrive at conclusions that are

broadly consistent with the observations around model inference of Scells *et al.* (2022).

Santhanam *et al.* (2022b) use this fact to encourage the adoption of multidimensional leaderboards and motivate research on metrics that capture the overall utility of a retrieval or ranking method in a single quantity. They point to the *Dynascores* proposed by Ma *et al.* (2021b) as one such measure that allows for a single ranking of a collection of systems. For example, they evaluate the four retrieval methods above in terms of their query latency, accuracy, and dollar cost (as measured on different cloud-based hardware platforms per million queries). By assigning different weights to each dimension (a "policy") and combining the measurements using Dynascores according to the policy, they order retrieval systems by their utility in the context of the given policy.

We too encourage the development of multidimensional leaderboards to incentivize research into efficient and effective systems. In fact, while Santhanam *et al.* (2022b) argue for leaderboards that capture inference efficiency, we believe training efficiency too must be reflected in the overall utility of a retrieval and ranking system. In spite of arguments that training a model incurs a cost that is amortized and thus comparably insignificant, we note that retrieval and ranking models have a relatively short lifetime: As the data distribution shifts, models must often be re-trained or fine-tuned on fresh samples. By incorporating these costs into model evaluation and comparison, a leaderboard could encourage reusability and recyclability of models. How these costs may be measured and factored into a ranking on a leaderboard, however, is an open question.

Acknowledgements

We are grateful to the three anonymous reviewers who perused an earlier version of this monograph meticulously and gave us constructive feedback. This manuscript benefited greatly from their thorough and thoughtful suggestions.

We drew inspiration from discussions we had with participants of the Workshop on Reaching Efficiency in Neural Information Retrieval (ReNeuIR) at ACM SIGIR 2022. We thank them for the topics they brought to our attention and their insight into all aspects of efficiency.

Finally, we extend our sincere gratitude to Maarten de Rijke for his patience, encouragement, and invaluable feedback as we prepared this manuscript.

This research has been partly funded by PNRR - M4C2 - Investimento 1.3, Partenariato Esteso PE00000013 - "FAIR - Future Artificial Intelligence Research" - Spoke 1 "Human-centered AI", funded by the European Commission under the NextGeneration EU programme.

Table 5.1: Highlights of cost-aware learning methods.

Method	Category	Strategy	Inference Speed-up
Joint optimization (Wang *et al.*, 2010a)	Cost-sensitive learning	Learn linear functions with a novel metric (EET) mixing efficiency and effectiveness	2×
CSTC (Xu *et al.*, 2013)	Cost-sensitive learning	Tree of classifiers reducing the number of features extracted per instance	2×
Submodular trees of classifiers (Kusner *et al.*, 2014)	Cost-sensitive learning	Tree of classifiers reducing the number of features extracted per instance. Reduced time for learning the model w.r.t (Xu *et al.*, 2013)	up to 119× (training time)
Pruning while boosting (Asadi and Lin, 2013c)	Node pruning	Collapse leaves to reduce tree depth.	1.6×
CLEAVER (Lucchese *et al.*, 2016a)	Tree pruning	Remove trees and tune weights.	2.6×
X-DART (Lucchese *et al.*, 2017b)	Tree pruning	Mute trees and remove them when appropriate	3×
Ranking Distillation (Tang and Wang, 2018)	Homogeneous knowledge distillation	Student trained on top-ranking instances by teacher.	2×
Fast neural networks from tree-based ensembles (Nardini *et al.*, 2022; Tang and Wang, 2018; Cohen *et al.*, 2018)	Heterogeneous knowledge distillation	Feed-forward networks learned as students approximating λ-MART	4× (CPU), 100× (GPU)

Table 6.1: Highlights of inference methods.

Method	Category	Strategy	Inference Speed-up
Runtime optimizations for tree-based machine learning models (Asadi *et al.*, 2014)	Efficient Traversal	Predication and interleaved multi-document evaluation	2×
QuickScorer (Lucchese *et al.*, 2015b)	Efficient Traversal	Feature-wise traversal and cache-aware data layout	additional 6.5×
RapidScorer (Ye *et al.*, 2018)	Efficient Traversal	Compact data layout	3.5× (over QuickScorer)
Early Exit (Cambazoglu *et al.*, 2010)	Approximate Scoring	Terminate ensemble traversal early	4×
Dynamic cutoff prediction (Culpepper *et al.*, 2016)	Cascade	Query-based prediction of the number of candidate documents	2×
A Cascade Ranking Model (Wang *et al.*, 2011)	Cascade	Joint learning of pruning and ranker stage	tunable trade-off
Cost-Aware Cascade Ranking (Chen *et al.*, 2017)	Cascade	Expensive features are moved to later stages	2×

Table 8.1: Highlights of neural learning to rank.

Method	Category	Strategy
Early-exit on cascade Transformers (Soldaini and Moschitti, 2020)	Early exit	Discard fixed-size subset of documents at each exit point.
Early-exit on Transformers (Schwartz *et al.*, 2020)	Early exit	Early-exit inference of "easy" documents in Transformer-based networks. More layers are executed for "difficult" documents.
Per-layer early-exit for Transformers (Xin *et al.*, 2021; Xin *et al.*, 2020)	Early exit	One "off-ramp" classifier for each layer that terminates inference. Decision taken based on the confidence in the classification (Xin *et al.*, 2020). In a later work, the decision of the classifier (i.e., to "exit" or not) is learnt at every exit point of the network (Xin *et al.*, 2021).
Adaptive inference for distilling fast Transformer-based networks (Liu *et al.*, 2020)	Early exit, Distillation	Student-teacher model applied to BERT. The student model uses classifiers to enable early exits based on confidence.
Distilling smaller models from BERT (Gao *et al.*, 2020)	Distillation	Smaller models learnt from BERT and monoBERT. Hybrid distillation strategies (i.e., a student model learns from a general-purpose BERT model first, followed by distillation of ranking behavior from monoBERT) perform as well as monoBERT but are up to 9× faster at inference.
Single-vector dense retrieval (Karpukhin *et al.*, 2020; Xiong *et al.*, 2021; Lindgren *et al.*, 2021)	Dense retrieval	Documents and passages are encoded as single vectors. Several contributions concern the negative selection strategy that is crucial to generating effective dense representations. While DPR (Karpukhin *et al.*, 2020) selects negative examples in the beginning of the training process, ANCE (Xiong *et al.*, 2021) continually finds negative examples based on the current structure of the vector space by using an ANN index as the model is being trained. To limit the computational burden of the learning process, Lindgren *et al.* (2021) propose to cache negative results during training so to work more efficiently.
Multiple-vector dense retrieval (Khattab and Zaharia, 2020; Santhanam *et al.*, 2022a)	Dense retrieval	Each term of the documents and passages are encoded as a vector. Finer granularity leads to improved performance. Inference is often a two-step process: (1) use a language model to encode query and document terms (separately), (2) use a "late-interaction" function to estimate the overall similarity of query and document terms.

References

Ai, Q., X. Wang, S. Bruch, N. Golbandi, M. Bendersky, and M. Najork. (2019). "Learning Groupwise Multivariate Scoring Functions Using Deep Neural Networks". In: *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*. Santa Clara, CA, USA. 85–92.

Akkalyoncu Yilmaz, Z., S. Wang, W. Yang, H. Zhang, and J. Lin. (2019). "Applying BERT to Document Retrieval with Birch". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*.

Asadi, N. (2013). *Multi-Stage Search Architectures for Streaming Documents*. University of Maryland.

Asadi, N. and J. Lin. (2012). "Fast Candidate Generation for Two-Phase Document Ranking: Postings List Intersection with Bloom Filters". In: *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*. Maui, Hawaii, USA. 2419–2422.

Asadi, N. and J. Lin. (2013a). "Effectiveness/Efficiency Tradeoffs for Candidate Generation in Multi-Stage Retrieval Architectures". In: *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Dublin, Ireland. 997–1000.

Asadi, N. and J. Lin. (2013b). "Fast Candidate Generation for Real-Time Tweet Search with Bloom Filter Chains". *ACM Transactions on Information Systems*. 31(3).

Asadi, N. and J. Lin. (2013c). "Training Efficient Tree-Based Models for Document Ranking". In: *Proceedings of the 35th European Conference on Advances in Information Retrieval*. Moscow, Russia. 146–157.

Asadi, N., J. Lin, and A. P. de Vries. (2014). "Runtime Optimizations for Tree-Based Machine Learning Models." *IEEE Transactions on Knowledge and Data Engineering*. 26(9): 2281–2292.

Ba, J. and R. Caruana. (2014). "Do Deep Nets Really Need to be Deep?" In: *Advances in neural information processing systems*. 2654–2662.

Bendersky, M., W. B. Croft, and Y. Diao. (2011). "Quality-Biased Ranking of Web Documents". In: *Proceedings of the 4th ACM International Conference on Web Search and Data Mining*. Hong Kong, China. 95–104.

Bennett, P. N., K. Svore, and S. T. Dumais. (2010). "Classification-enhanced Ranking". In: *Proceedings of the 19th International Conference on World Wide Web*. 111–120.

Blondel, M., O. Teboul, Q. Berthet, and J. Djolonga. (2020). "Fast Differentiable Sorting and Ranking". In: *Proceedings of the 37th International Conference on Machine Learning*.

Borisov, A., I. Markov, M. de Rijke, and P. Serdyukov. (2016). "A Neural Click Model for Web Search". In: *Proceedings of the 25th International Conference on World Wide Web*. 531–541.

Breiman, L., J. Friedman, C. J. Stone, and R. Olshen. (1984). *Classification and Regression Trees*. Chapman and Hall/CRC.

Broder, A. Z., D. Carmel, M. Herscovici, A. Soffer, and J. Zien. (2003). "Efficient Query Evaluation Using a Two-Level Retrieval Process". In: *Proceedings of the 12th International Conference on Information and Knowledge Management*. New Orleans, LA, USA. 426–434.

Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. (2020). "Language Models are Few-Shot Learners". arXiv: 2005.14165 [cs.CL].

Bruch, S. (2021). "An Alternative Cross Entropy Loss for Learning-to-Rank". In: *Proceedings of the Web Conference 2021*. Ljubljana, Slovenia. 118–126.

Bruch, S., S. Gai, and A. Ingber. (2022a). "An Analysis of Fusion Functions for Hybrid Retrieval". arXiv: 2210.11934 [cs.IR].

Bruch, S., S. Han, M. Bendersky, and M. Najork. (2020). "A Stochastic Treatment of Learning to Rank Scoring Functions". In: *Proceedings of the 13th International Conference on Web Search and Data Mining*. 61–69.

Bruch, S., C. Lucchese, and F. M. Nardini. (2022b). "ReNeuIR: Reaching Efficiency in Neural Information Retrieval". In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Madrid, Spain. 3462–3465.

Bruch, S., C. Lucchese, and F. M. Nardini. (2023). "Report on the 1st Workshop on Reaching Efficiency in Neural Information Retrieval (ReNeuIR 2022) at SIGIR 2022". *SIGIR Forum*. 56(2).

Bruch, S., X. Wang, M. Bendersky, and M. Najork. (2019a). "An Analysis of the Softmax Cross Entropy Loss for Learning-to-Rank with Binary Relevance". In: *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*. Santa Clara, CA, USA. 75–78.

Bruch, S., M. Zoghi, M. Bendersky, and M. Najork. (2019b). "Revisiting Approximate Metric Optimization in the Age of Deep Neural Networks". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. Paris, France. 1241–1244.

Buckley, C. and E. Voorhees. (2005). "Retrieval System Evaluation". In: *TREC: Experiment and Evaluation in Information Retrieval*. MIT Press. Chap. 3.

Burges, C., T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. (2005). "Learning to Rank using Gradient Descent". In: *Proceedings of the 22nd international conference on Machine learning*. ACM. 89–96.

Burges, C. J. (2010). "From RankNet to LambdaRank to LambdaMART: An Overview". *Tech. rep.* No. MSR-TR-2010-82.

Busolin, F., C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, and S. Trani. (2021). "Learning Early Exit Strategies for Additive Ranking Ensembles". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada*. 2217–2221.

Cambazoglu, B. B., H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt. (2010). "Early Exit Optimizations for Additive Machine Learned Ranking Systems". In: *Proceedings of the 3rd International Conference on Web Search and Web Data Mining (WSDM)*. ACM. 411–420.

Cao, Z., T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. (2007). "Learning to Rank: from Pairwise Approach to Listwise Approach". In: *Proceedings of the 24th International Conference on Machine learning*. ACM. 129–136.

Capannini, G., C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, and N. Tonellotto. (2016). "Quality Versus Efficiency in Document Scoring with Learning-to-rank Models". *Information Processing & Management*. 52(6): 1161–1177.

Carterette, B., P. Bennett, D. Chickering, and S. Dumais. (2008). "Here or there". *Advances in Information Retrieval*: 16–27.

Chapelle, O. and Y. Chang. (2011). "Yahoo! Learning to Rank Challenge Overview". In: *Proceedings of the Learning to Rank Challenge*. 1–24.

Chapelle, O., T. Joachims, F. Radlinski, and Y. Yue. (2012). "Large-scale Validation and Analysis of Interleaved Search Evaluation". *ACM Transactions on Information Systems*. 30(1): 6.

Chapelle, O., D. Metlzer, Y. Zhang, and P. Grinspan. (2009). "Expected Reciprocal Rank for Graded Relevance". In: *Proceedings of the 18th ACM conference on Information and knowledge management*. 621–630.

Chapelle, O., P. Shivaswamy, S. Vadrevu, K. Weinberger, Y. Zhang, and B. Tseng. (2011). "Boosted Multi-task Learning". *Machine learning*. 85(1-2): 149–173.

Chen, M., Z. Xu, K. Weinberger, O. Chapelle, and D. Kedem. (2012). "Classifier cascade for minimizing feature evaluation cost". In: *Artificial Intelligence and Statistics*. 218–226.

Chen, R.-C., L. Gallagher, R. Blanco, and J. S. Culpepper. (2017). "Efficient Cost-Aware Cascade Ranking in Multi-Stage Retrieval". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Shinjuku, Tokyo, Japan. 445–454.

Chen, T., M. Zhang, J. Lu, M. Bendersky, and M. Najork. (2022). "Out-of-Domain Semantics to the Rescue! Zero-Shot Hybrid Retrieval Models". In: *Advances in Information Retrieval: 44th European Conference on IR Research, ECIR 2022, Stavanger, Norway, April 10–14, 2022, Proceedings, Part I*. Stavanger, Norway. 95–110.

Chen, T. and C. Guestrin. (2016). "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco, California, USA. 785–794.

Chuklin, A., I. Markov, and M. de Rijke. (2015). *Click Models for Web Search*. Morgan & Claypool. ISBN: 9781627056489.

Cohen, D., J. Foley, H. Zamani, J. Allan, and W. B. Croft. (2018). "Universal Approximation Functions for Fast Learning to Rank: Replacing Expensive Regression Forests with Simple Feed-forward Networks". In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM. 1017–1020.

Cormack, G. V., M. D. Smucker, and C. L. Clarke. (2011). "Efficient and Effective Spam Filtering and Re-ranking for Large Web Datasets". *Information Retrieval*. 14: 441–465.

Culpepper, J. S., C. L. Clarke, and J. Lin. (2016). "Dynamic Cutoff Prediction in Multi-stage Retrieval Systems". In: *Proceedings of the 21st Australasian Document Computing Symposium*. ACM. 17–24.

Cuturi, M., O. Teboul, and J.-P. Vert. (2019). "Differentiable Ranking and Sorting using Optimal Transport". In: *Advances in Neural Information Processing Systems*. Vol. 32.

Dai, Z. and J. Callan. (2019). "Deeper Text Understanding for IR with Contextual Neural Language Modeling". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. Paris, France. 985–988.

Dai, Z., C. Xiong, J. Callan, and Z. Liu. (2018). "Convolutional Neural Networks for Soft-Matching N-Grams in Ad-Hoc Search". In: *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*. Marina Del Rey, CA, USA. 126–134.

Dang, V., M. Bendersky, and W. B. Croft. (2013). "Two-Stage learning to rank for information retrieval". In: *Advances in Information Retrieval*. Springer. 423–434.

Dato, D., C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. (2016). "Fast Ranking with Additive Ensembles of Oblivious and Non-Oblivious Regression Trees". *ACM Transactions on Information Systems*. 35(2): 15:1–15:31.

Dehghani, M., H. Zamani, A. Severyn, J. Kamps, and W. B. Croft. (2017). "Neural Ranking Models with Weak Supervision". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Tokyo, Japan. 65–74.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 4171–4186.

Diaz, F., B. Mitra, M. D. Ekstrand, A. J. Biega, and B. Carterette. (2020). "Evaluating Stochastic Rankings with Expected Exposure". In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. Virtual Event, Ireland. 275–284.

Ding, S. and T. Suel. (2011). "Faster Top-k Document Retrieval Using Block-Max Indexes". In: *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Beijing, China. 993–1002.

Dredze, M., R. Gevaryahu, and A. Elias-Bachrach. (2007). "Learning Fast Classifiers for Image Spam." In: *CEAS*. 2007–487.

Efron, B., T. Hastie, I. Johnstone, R. Tibshirani, *et al.* (2004). "Least Angle Regression". *The Annals of Statistics*. 32(2): 407–499.

Formal, T., C. Lassance, B. Piwowarski, and S. Clinchant. (2022). "From Distillation to Hard Negative Sampling: Making Sparse Neural IR Models More Effective". In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Madrid, Spain. 2353–2359.

Freund, Y., R. Iyer, R. E. Schapire, and Y. Singer. (2003). "An Efficient Boosting Algorithm for Combining Preferences". *Journal of Machine Learning Research*. 4(Nov): 933–969.

Friedman, J. H. (2001). "Greedy Function Approximation: a Gradient Boosting Machine". *Annals of Statistics*: 1189–1232.

Gallagher, L., R.-C. Chen, R. Blanco, and J. S. Culpepper. (2019). "Joint Optimization of Cascade Ranking Models". In: *Proceedings of the 12th ACM International Conference on Web Search and Data Mining*. Melbourne VIC, Australia. 15–23.

Ganjisaffar, Y., R. Caruana, and C. V. Lopes. (2011). "Bagging Gradient-boosted Trees for High Precision, Low Variance Ranking Models". In: *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. 85–94.

Gao, L. and J. Callan. (2021a). "Condenser: a Pre-training Architecture for Dense Retrieval". arXiv: 2104.08253 [cs.CL].

Gao, L. and J. Callan. (2021b). "Unsupervised Corpus Aware Language Model Pre-training for Dense Passage Retrieval". arXiv: 2108.05540 [cs.IR].

Gao, L., Z. Dai, and J. Callan. (2020). "Understanding BERT Rankers Under Distillation". In: *Proceedings of the 2020 ACM SIGIR on International Conference on Theory of Information Retrieval*. Virtual Event, Norway. 149–152.

Geng, X., T.-Y. Liu, T. Qin, and H. Li. (2007). "Feature Selection for Ranking". In: *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Amsterdam, The Netherlands. 407–414.

Gigli, A., C. Lucchese, F. M. Nardini, and R. Perego. (2016). "Fast Feature Selection for Learning to Rank". In: *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*. Newark, Delaware, USA. 167–170.

Gil-Costa, V., F. Loor, R. Molina, F. M. Nardini, R. Perego, and S. Trani. (2022). "Ensemble Model Compression for Fast and Energy-Efficient Ranking on FPGAs". In: *Advances in Information Retrieval*. Springer. 260–273.

Gomes, B. (2017). "Our Latest Quality Improvements for Search". URL: https://www.blog.google/products/search/our-latest-quality-improvements-search/.

Gordon, M., K. Duh, and N. Andrews. (2020). "Compressing BERT: Studying the Effects of Weight Pruning on Transfer Learning". In: *Proceedings of the 5th Workshop on Representation Learning for NLP*. 143–155.

Guo, J., Y. Fan, Q. Ai, and W. B. Croft. (2016). "A Deep Relevance Matching Model for Ad-Hoc Retrieval". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. Indianapolis, Indiana, USA. 55–64.

Guo, J., Y. Fan, L. Pang, L. Yang, Q. Ai, H. Zamani, C. Wu, W. B. Croft, and X. Cheng. (2020). "A Deep Look into Neural Ranking Models for Information Retrieval". *Information Processing & Management*. 57(6).

He, X., J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, *et al.* (2014). "Practical Lessons from Predicting Clicks on Ads at Facebook". In: *Proceedings of the 8th International Workshop on Data Mining for Online Advertising*. 1–9.

Henzinger, M. R. *et al.* (2000). "Link Analysis in Web Information Retrieval". *IEEE Data Engineering Bulletin*. 23(3): 3–8.

Hinton, G. E., N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. (2012). "Improving Neural Networks by Preventing Co-adaptation of Feature Detectors". arXiv: 1207.0580 [cs.NE].

Hoerl, A. E. and R. W. Kennard. (1970). "Ridge Regression: Biased Estimation for Nonorthogonal Problems". *Technometrics*. 12(1): 55–67.

Hofmann, K., A. Schuth, S. Whiteson, and M. de Rijke. (2013a). "Reusing Historical Interaction Data for Faster Online Learning to Rank for IR". In: *Proceedings of the 6th ACM International Conference on Web Search and Data Mining*. 183–192.

Hofmann, K., S. Whiteson, and M. de Rijke. (2013b). "Balancing Exploration and Exploitation in Listwise and Pairwise Online Learning to Rank for Information Retrieval". *Information Retrieval*. 16(1): 63–90.

Hofstätter, S., H. Zamani, B. Mitra, N. Craswell, and A. Hanbury. (2020). "Local Self-Attention over Long Text for Efficient Document Retrieval". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. Virtual Event, China. 2021–2024.

Huang, P.-S., X. He, J. Gao, L. Deng, A. Acero, and L. Heck. (2013). "Learning Deep Structured Semantic Models for Web Search using Clickthrough Data". In: *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*. ACM. 2333–2338.

Jagerman, R., Z. Qin, X. Wang, M. Bendersky, and M. Najork. (2022). "On Optimizing Top-K Metrics for Neural Ranking Models". In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Madrid, Spain. 2303–2307.

Järvelin, K. and J. Kekäläinen. (2000). "IR Evaluation Methods for Retrieving Highly Relevant Documents". In: *Proceedings of the 23rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 41–48.

Jiang, D., K. W.-T. Leung, and W. Ng. (2016). "Query Intent Mining with Multiple Dimensions of Web Search Data". *Proceedings of the 25th International Conference on World Wide Web*. 19(3): 475–497.

Jiao, X., Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu. (2020). "TinyBERT: Distilling BERT for Natural Language Understanding". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*.

Jin, X., T. Yang, and X. Tang. (2016). "A Comparison of Cache Blocking Methods for Fast Execution of Ensemble-based Score Computation". In: *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Pisa, Italy. 629–638.

Joachims, T. (2002). "Optimizing Search Engines using Clickthrough Data". In: *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 133–142.

Joachims, T., L. Granka, B. Pan, H. Hembrooke, and G. Gay. (2005). "Accurately Interpreting Clickthrough Data as Implicit Feedback". In: *Proceedings of the 28th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 154–161.

Joachims, T., A. Swaminathan, and T. Schnabel. (2017). "Unbiased Learning-to-rank with Biased Feedback". In: *Proceedings of the 10th ACM International Conference on Web Search and Data Mining*. 781–789.

Johnson, J., M. Douze, and H. Jégou. (2021). "Billion-Scale Similarity Search with GPUs". *IEEE Transactions on Big Data*. 7: 535–547.

Jones, K. S., S. Walker, and S. E. Robertson. (2000). "A Probabilistic Model of Information Retrieval: Development and Comparative Experiments: Part 2". *Information processing & management*. 36(6): 809–840.

Karpukhin, V., B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. (2020). "Dense Passage Retrieval for Open-Domain Question Answering". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 6769–6781.

Ke, G., Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. (2017). "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Long Beach, California, USA. 3149–3157.

Khattab, O. and M. Zaharia. (2020). "ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. Virtual Event, China. 39–48.

Kohavi, R., A. Deng, B. Frasca, T. Walker, Y. Xu, and N. Pohlmann. (2013). "Online Controlled Experiments at Large Scale". In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1168–1176.

Kusner, M. J., W. Chen, Q. Zhou, Z. E. Xu, K. Q. Weinberger, and Y. Chen. (2014). "Feature-Cost Sensitive Learning with Submodular Trees of Classifiers". In: *AAAI*. 1939–1945.

Kveton, B., C. Szepesvari, Z. Wen, and A. Ashkan. (2015). "Cascading Bandits: Learning to Rank in the Cascade Model". In: *International Conference on Machine Learning*. 767–776.

Lassance, C. and S. Clinchant. (2022). "An Efficiency Study for SPLADE Models". In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Madrid, Spain. 2220–2226.

Lettich, F., C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. (2019). "Parallel Traversal of Large Ensembles of Decision Trees". *IEEE Transactions on Parallel and Distributed Systems*. 30(9): 2075–2089.

Li, C., A. Yates, S. MacAvaney, B. He, and Y. Sun. (2020). "PARADE: Passage Representation Aggregation for Document Reranking". arXiv: 2008.09093 [cs.IR].

Lin, J., R. Nogueira, and A. Yates. (2021). "Pretrained Transformers for Text Ranking: BERT and Beyond". arXiv: 2010.06467 [cs.IR].

Lin, S.-C., J.-H. Yang, and J. Lin. (2020a). "Distilling Dense Representations for Ranking using Tightly-coupled Teachers". arXiv: 2010.11386 [cs.IR].

Lin, Z., J. Liu, Z. Yang, N. Hua, and D. Roth. (2020b). "Pruning Redundant Mappings in Transformer Models via Spectral-Normalized Identity Prior". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*.

Lindgren, E., S. Reddi, R. Guo, and S. Kumar. (2021). "Efficient Training of Retrieval Models using Negative Cache". *Advances in Neural Information Processing Systems*. 34: 4134–4146.

Ling, X., W. Deng, C. Gu, H. Zhou, C. Li, and F. Sun. (2017). "Model Ensemble for Click Prediction in Bing Search Ads". In: *Proceedings of the 26th International Conference on World Wide Web Companion*. 689–698.

Liu, S., F. Xiao, W. Ou, and L. Si. (2017). "Cascade Ranking for Operational E-commerce Search". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 1557–1565.

Liu, T.-Y. (2009). "Learning to Rank for Information Retrieval". *Foundations and Trends in Information Retrieval*. 3(3): 225–331.

Liu, W., P. Zhou, Z. Wang, Z. Zhao, H. Deng, and Q. Ju. (2020). "FastBERT: a Self-distilling BERT with Adaptive Inference Time". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 6035–6044.

Liu, Z., F. Li, G. Li, and J. Cheng. (2021). "EBERT: Efficient BERT Inference with Dynamic Structured Pruning". In: *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. 4814–4823.

Long, B., O. Chapelle, Y. Zhang, Y. Chang, Z. Zheng, and B. Tseng. (2010). "Active Learning for Ranking through Expected Loss Optimization". In: *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 267–274.

Luan, Y., J. Eisenstein, K. Toutanova, and M. Collins. (2021). "Sparse, Dense, and Attentional Representations for Text Retrieval". *Transactions of the Association for Computational Linguistics*. 9: 329–345.

Lucchese, C., C. I. Muntean, F. M. Nardini, R. Perego, and S. Trani. (2017a). "RankEval: An Evaluation and Analysis Framework for Learning-to-Rank Solutions". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Shinjuku, Tokyo, Japan. 1281–1284.

Lucchese, C., C. I. Muntean, F. M. Nardini, R. Perego, and S. Trani. (2020a). "RankEval: Evaluation and Investigation of Ranking Models". *SoftwareX*. 12: 100614. ISSN: 2352-7110.

Lucchese, C., F. M. Nardini, S. Orlando, R. Perego, F. Silvestri, and S. Trani. (2016a). "Post-Learning Optimization of Tree Ensembles for Efficient Ranking". In: *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Pisa, Italy. 949–952.

Lucchese, C., F. M. Nardini, S. Orlando, R. Perego, F. Silvestri, and S. Trani. (2018a). "X-CLEaVER: Learning Ranking Ensembles by Growing and Pruning Trees". *ACM Transactions on Intelligent Systems and Technology*. 9(6).

Lucchese, C., F. M. Nardini, S. Orlando, R. Perego, and N. Tonellotto. (2015a). "Speeding up Document Ranking with Rank-based Features". In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. 895–898.

Lucchese, C., F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. (2015b). "QuickScorer: A Fast Algorithm to Rank Documents with Additive Ensembles of Regression Trees". In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 73–82.

Lucchese, C., F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. (2016b). "Exploiting CPU SIMD Extensions to Speed-up Document Scoring with Tree Ensembles". In: *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Pisa, Italy. 833–836.

Lucchese, C., F. M. Nardini, S. Orlando, R. Perego, and S. Trani. (2017b). "X-DART: Blending Dropout and Pruning for Efficient Learning to Rank". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Shinjuku, Tokyo, Japan. 1077–1080.

Lucchese, C., F. M. Nardini, S. Orlando, R. Perego, and S. Trani. (2020b). "Query-Level Early Exit for Additive Learning-to-Rank Ensembles". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '20*. Virtual Event, China: ACM. 2033–2036.

Lucchese, C., F. M. Nardini, R. Perego, S. Orlando, and S. Trani. (2018b). "Selective Gradient Boosting for Effective Learning to Rank". In: *The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*. Ann Arbor, MI, USA. 155–164.

Lucchese, C., S. Orlando, R. Perego, F. Silvestri, and G. Tolomei. (2013). "Discovering Tasks from Search Engine Query Logs". *ACM Transactions on Information Systems*. 31(3): 14.

Ma, X., K. Sun, R. Pradeep, and J. Lin. (2021a). "A Replication Study of Dense Passage Retriever". arXiv: 2104.05740 [cs.IR].

Ma, Z., K. Ethayarajh, T. Thrush, S. Jain, L. Y. Wu, R. Jia, C. Potts, A. Williams, and D. Kiela. (2021b). "Dynaboard: An Evaluation-As-A-Service Platform for Holistic Next-Generation Benchmarking". In: *Neural Information Processing Systems*.

MacAvaney, S., S. Feldman, N. Goharian, D. Downey, and A. Cohan. (2020a). "ABNIRML: Analyzing the Behavior of Neural IR Models". *arXiv*. abs/2011.00696. URL: https://arxiv.org/abs/2011.00696.

MacAvaney, S., C. Macdonald, and I. Ounis. (2022). "Streamlining Evaluation with ir-measures". In: *Advances in Information Retrieval*. Springer International Publishing.

MacAvaney, S., F. M. Nardini, R. Perego, N. Tonellotto, N. Goharian, and O. Frieder. (2020b). "Efficient Document Re-Ranking for Transformers by Precomputing Term Representations". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. Virtual Event, China. 49–58.

MacAvaney, S., F. M. Nardini, R. Perego, N. Tonellotto, N. Goharian, and O. Frieder. (2020c). "Expansion via Prediction of Importance with Contextualization". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1573–1576.

MacAvaney, S., A. Yates, A. Cohan, and N. Goharian. (2019). "CEDR: Contextualized Embeddings for Document Ranking". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. Paris, France. 1101–1104.

Macdonald, C., R. L. Santos, and I. Ounis. (2012). "On the Usefulness of Query Features for Learning to Rank". In: *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*. 2559–2562.

Macdonald, C., R. L. Santos, and I. Ounis. (2013). "The Whens and Hows of Learning to Rank for Web Search". *Information Retrieval*. 16(5): 584–628.

Mackenzie, J., J. S. Culpepper, R. Blanco, M. Crane, C. L. Clarke, and J. Lin. (2018). "Query Driven Algorithm Selection in Early Stage Retrieval". In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM. 396–404.

Mackenzie, J., M. Petri, and A. Moffat. (2021). "Anytime Ranking on Document-Ordered Indexes". *ACM Transactions on Information Systems*. 40(1).

Malkov, Y. A. and D. A. Yashunin. (2016). "Efficient and Robust Approximate Nearest Neighbor Search using Hierarchical Navigable Small World graphs". arXiv: 1603.09320 [cs.DS].

Mallia, A., J. Mackenzie, T. Suel, and N. Tonellotto. (2022). "Faster Learned Sparse Retrieval with Guided Traversal". In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Madrid, Spain. 1901–1905.

Matsubara, Y., T. Vu, and A. Moschitti. (2020). "Reranking for Efficient Transformer-Based Answer Selection". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1577–1580.

McCarley, J. S., R. Chakravarti, and A. Sil. (2021). "Structured Pruning of a BERT-based Question Answering Model". arXiv: 1910.06360 [cs.CL].

Metzler, D. and W. B. Croft. (2007). "Linear Feature-based Models for Information Retrieval". *Information Retrieval*. 10(3): 257–274.

Mikolov, T., K. Chen, G. S. Corrado, and J. Dean. (2013). "Efficient Estimation of Word Representations in Vector Space". arXiv: 1301.3781 [cs.CL].

Mitra, B. and N. Craswell. (2017). “Neural Models for Information Retrieval”. arXiv: 1705.01509 [cs.IR].

Mitra, B., F. Diaz, and N. Craswell. (2017). “Learning to Match using Local and Distributed Representations of Text for Web Search”. In: *Proceedings of the 26th International Conference on World Wide Web*. 1291–1299.

Mitra, B., S. Hofstätter, H. Zamani, and N. Craswell. (2021). “Improving Transformer-Kernel Ranking Model Using Conformer and Query Term Independence”. In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1697–1702.

Mitra, B., E. Nalisnick, N. Craswell, and R. Caruana. (2016). “A Dual Embedding Space Model for Document Ranking”. arXiv: 1602.01137 [cs.IR].

Moffat, A. and J. Zobel. (2008). “Rank-biased Precision for Measurement of Retrieval Effectiveness”. *ACM Transactions on Information Systems*. 27(1): 2.

Mohan, A., Z. Chen, and K. Weinberger. (2011). “Web-search Ranking with Initialized Gradient Boosted Regression Trees”. In: *Proceedings of the learning to rank challenge*. 77–89.

Molina, R., F. Loor, V. Gil-Costa, F. M. Nardini, R. Perego, and S. Trani. (2021). “Efficient Traversal of Decision Tree Ensembles with FPGAs”. *Journal of Parallel and Distributed Computing*. 155: 38–49.

Nardini, F. M., C. Rulli, S. Trani, and R. Venturini. (2022). “Distilled Neural Networks for Efficient Learning to Rank”. *IEEE Transactions on Knowledge and Data Engineering*: 1–1.

Nguyen, T., M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng. (2016). “MS MARCO: A Human Generated MAchine Reading COmprehension Dataset”. Nov.

Nogueira, R. and K. Cho. (2020). “Passage Re-ranking with BERT”. arXiv: 1901.04085 [cs.IR].

Nogueira, R., Z. Jiang, R. Pradeep, and J. Lin. (2020). “Document Ranking with a Pre-trained Sequence-to-Sequence Model”. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. 708–718.

Nogueira, R. and J. Lin. (2019). “From doc2query to docTTTTTquery”.

Nogueira, R., W. Yang, K. Cho, and J. Lin. (2019a). “Multi-stage document ranking with BERT”. arXiv: 1910.14424 [cs.IR].

Nogueira, R., W. Yang, J. Lin, and K. Cho. (2019b). “Document Expansion by Query Prediction”. arXiv: 1904.08375 [cs.IR].

Onal, K. D., Y. Zhang, I. S. Altingovde, M. M. Rahman, P. Karagoz, A. Braylan, B. Dang, H.-L. Chang, H. Kim, Q. Mcnamara, A. Angert, E. Banner, V. Khetan, T. Mcdonnell, A. T. Nguyen, D. Xu, B. C. Wallace, M. Rijke, and M. Lease. (2018). “Neural Information Retrieval: At the End of the Early Years”. *Information Retrieval*. 21(2–3): 111–182.

Oosterhuis, H. (2021). "Computationally Efficient Optimization of Plackett-Luce Ranking Models for Relevance and Fairness". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM.

Oosterhuis, H., R. Jagerman, and M. de Rijke. (2020). "Unbiased Learning to Rank: Counterfactual and Online Approaches". In: *Companion Proceedings of the Web Conference 2020*. Taipei, Taiwan. 299–300.

Oosterhuis, H. and M. de Rijke. (2017). "Balancing Speed and Quality in Online Learning to Rank for Information Retrieval". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM. 277–286.

Pang, L., J. Xu, Q. Ai, Y. Lan, X. Cheng, and J. Wen. (2020). "SetRank: Learning a Permutation-Invariant Ranking Model for Information Retrieval". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Petri, M., A. Moffat, J. Mackenzie, J. S. Culpepper, and D. Beck. (2019). "Accelerated Query Processing Via Similarity Score Prediction". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. Paris, France. 485–494.

Ponte, J. M. and W. B. Croft. (1998). "A Language Modeling Approach to Information Retrieval". In: *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. 275–281.

Pradeep, R., R. Nogueira, and J. Lin. (2021). "The Expando-Mono-Duo Design Pattern for Text Ranking with Pretrained Sequence-to-Sequence Models". arXiv: 2101.05667 [cs.IR].

Prokhorenkova, L., G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. (2018). "CatBoost: Unbiased Boosting with Categorical Features". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. Montréal, Canada. 6639–6649.

Qin, T., T.-Y. Liu, and H. Li. (2010). "A General Approximation Framework for Direct Optimization of Information Retrieval Measures". *Information Retrieval*. 13(4): 375–397.

Qu, Y., Y. Ding, J. Liu, K. Liu, R. Ren, W. X. Zhao, D. Dong, H. Wu, and H. Wang. (2021). "RocketQA: An Optimized Training Approach to Dense Passage Retrieval for Open-Domain Question Answering". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 5835–5847.

Radlinski, F. and T. Joachims. (2005). "Query Chains: Learning to Rank from Implicit Feedback". In: *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. ACM. 239–248.

Radlinski, F., R. Kleinberg, and T. Joachims. (2008). "Learning Diverse Rankings with Multi-armed Bandits". In: *Proceedings of the 25th International Conference on Machine Learning*. 784–791.

Raffel, C., N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. (2020). "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". *Journal of Machine Learning Research*. 21(140): 1–67.

Rasolofo, Y. and J. Savoy. (2003). "Term Proximity Scoring for Keyword-based Retrieval Systems". *Advances in Information Retrieval*: 79–79.

Robertson, S., H. Zaragoza, and M. Taylor. (2004). "Simple BM25 Extension to Multiple Weighted Fields". In: *Proceedings of the 13th ACM International Conference on Information and Knowledge Management*. 42–49.

Salton, G. and C. Buckley. (1988). "Term-weighting approaches in automatic text retrieval". *Information Processing & Management*. 24(5): 513–523.

Sanh, V., L. Debut, J. Chaumond, and T. Wolf. (2020). "DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter". arXiv: 1910.01108 [cs.CL].

Santhanam, K., O. Khattab, J. Saad-Falcon, C. Potts, and M. Zaharia. (2022a). "ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction". In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 3715–3734.

Santhanam, K., J. Saad-Falcon, M. Franz, O. Khattab, A. Sil, R. Florian, M. A. Sultan, S. Roukos, M. Zaharia, and C. Potts. (2022b). "Moving Beyond Downstream Task Accuracy for Information Retrieval Benchmarking". arXiv: 2212.01340 [cs.IR].

Scells, H., S. Zhuang, and G. Zuccon. (2022). "Reduce, Reuse, Recycle: Green Information Retrieval Research". In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Madrid, Spain. 2825–2837.

Schwartz, R., G. Stanovsky, S. Swayamdipta, J. Dodge, and N. A. Smith. (2020). "The Right Tool for the Job: Matching Model and Instance Complexities". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 6640–6651.

Severyn, A. and A. Moschitti. (2015). "Learning to Rank Short Text Pairs with Convolutional Deep Neural Networks". In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. 373–382.

Shen, Y., X. He, J. Gao, L. Deng, and G. Mesnil. (2014). "Learning Semantic Representations using Convolutional Neural Networks for Web Search". In: *Proceedings of the 23rd International Conference on World Wide Web*. ACM. 373–374.

Soldaini, L. and A. Moschitti. (2020). "The Cascade Transformer: an Application for Efficient Answer Sentence Selection". In: *ACL*.

Sorokina, D. and E. Cantú-Paz. (2016). "Amazon Search: The Joy of Ranking Products". In: *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 459–460.

Sparck Jones, K. (1972). "A Statistical Interpretation of Term Specificity and its Application in Retrieval". *Journal of documentation*. 28(1): 11–21.

Strubell, E., A. Ganesh, and A. McCallum. (2019). "Energy and Policy Considerations for Deep Learning in NLP". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy. 3645–3650.

Swezey, R., A. Grover, B. Charron, and S. Ermon. (2021). "PiRank: Scalable Learning To Rank via Differentiable Sorting". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan. Vol. 34. 21644–21654.

Szummer, M. and E. Yilmaz. (2011). "Semi-supervised Learning to Rank with Preference Regularization". In: *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*. 269–278.

Tang, J. and K. Wang. (2018). "Ranking Distillation: Learning Compact Ranking Models With High Performance for Recommender System". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2289–2298.

Tang, X., X. Jin, and T. Yang. (2014). "Cache-conscious Runtime Optimization for Ranking Ensembles". In: *Proceedings of the 37th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 1123–1126.

Tax, N., S. Bockting, and D. Hiemstra. (2015). "A Cross-benchmark Comparison of 87 Learning to Rank Methods". *Information Processing & Management*. 51(6): 757–772.

Taylor, M., J. Guiver, S. Robertson, and T. Minka. (2008). "SoftRank: Optimizing Non-Smooth Rank Metrics". In: *Proceedings of the 2008 International Conference on Web Search and Data Mining*. Palo Alto, California, USA. 77–86.

Thakur, N., N. Reimers, A. Rücklé, A. Srivastava, and I. Gurevych. (2021). "BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models". In: *35th Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

The Guardian. (2017). "Google tells Army of 'Quality Raters' to Flag Holocaust denial". URL: https://www.theguardian.com/technology/2017/mar/15/google-quality-raters-flag-holocaust-denial-fake-news.

Tonellotto, N. and C. Macdonald. (2021). "Query Embedding Pruning for Dense Retrieval". In: *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*. Virtual Event, Queensland, Australia. 3453–3457.

Tseng, P. *et al.* (1988). "Coordinate Ascent for Maximizing Nondifferentiable Concave Functions".

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. (2017). "Attention is All You Need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Long Beach, California, USA. 6000–6010.

Vinayak, R. K. and R. Gilad-Bachrach. (2015). "DART: Dropouts meet Multiple Additive Regression Trees". In: *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*. Ed. by G. Lebanon and S. V. N. Vishwanathan. Vol. 38. *Proceedings of Machine Learning Research*. San Diego, California, USA: PMLR. 489–497.

Wang, L., J. Lin, and D. Metzler. (2011). "A Cascade Ranking Model for Efficient Ranked Retrieval". In: *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Beijing, China. 105–114.

Wang, L., J. J. Lin, and D. Metzler. (2010a). "Learning to Efficiently Rank". In: *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 138–145.

Wang, L., D. Metzler, and J. Lin. (2010b). "Ranking Under Temporal Constraints". In: *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*. Toronto, ON, Canada. 79–88.

Wang, M., X. Xu, Q. Yue, and Y. Wang. (2021a). "A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search". *Proc. VLDB Endow.* 14(11): 1964–1978.

Wang, S., S. Zhuang, and G. Zuccon. (2021b). "BERT-Based Dense Retrievers Require Interpolation with BM25 for Effective Passage Retrieval". In: *Proceedings of the 2021 ACM SIGIR International Conference on Theory of Information Retrieval*. Virtual Event, Canada. 317–324.

Xia, F., T.-Y. Liu, J. Wang, W. Zhang, and H. Li. (2008). "Listwise Approach to Learning to Rank: Theory and Algorithm". In: *Proceedings of the 25th International Conference on Machine Learning*. Helsinki, Finland. 1192–1199.

Xie, Y., H. Dai, M. Chen, B. Dai, T. Zhao, H. Zha, W. Wei, and T. Pfister. (2020). "Differentiable Top-k with Optimal Transport". In: *Advances in Neural Information Processing Systems*. Vol. 33. 20520–20531.

Xin, J., R. Tang, J. Lee, Y. Yu, and J. Lin. (2020). "DeeBERT: Dynamic Early Exiting for Accelerating BERT Inference". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Xin, J., R. Tang, Y. Yu, and J. Lin. (2021). "BERxiT: Early Exiting for BERT with Better Fine-Tuning and Extension to Regression". In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. 91–104.

Xiong, C., Z. Dai, J. Callan, Z. Liu, and R. Power. (2017). "End-to-End Neural Ad-Hoc Ranking with Kernel Pooling". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Shinjuku, Tokyo, Japan. 55–64.

Xiong, L., C. Xiong, Y. Li, K.-F. Tang, J. Liu, P. Bennett, J. Ahmed, and A. Overwijk. (2021). "Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval". In: *International Conference on Learning Representations*.

Xu, C. and J. McAuley. (2022). "A Survey on Model Compression and Acceleration for Pretrained Language Models". arXiv: 2202.07105 [cs.CL].

Xu, J., W. Zhou, Z. Fu, H. Zhou, and L. Li. (2021). "A Survey on Green Deep Learning". arXiv: 2111.05193 [cs.CL].

Xu, J. and H. Li. (2007). "AdaRank: a Boosting Algorithm for Information Retrieval". In: *Proceedings of the 30th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 391–398.

Xu, Z., M. J. Kusner, K. Q. Weinberger, and M. Chen. (2013). "Cost-sensitive Tree of Classifiers". In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. Atlanta, GA, USA. I-133–I-141.

Ye, T., H. Zhou, W. Y. Zou, B. Gao, and R. Zhang. (2018). "RapidScorer: Fast Tree Ensemble Evaluation by Maximizing Compactness in Data Level Parallelization". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 941–950.

Yilmaz, E. and S. Robertson. (2009). "Deep versus Shallow Judgments in Learning to Rank". In: *Proceedings of the 32nd international ACM SIGIR Conference on Research and Development in Information Retrieval*. 662–663.

Yin, D., Y. Hu, J. Tang, T. D. Jr., M. Zhou, H. Ouyang, J. Chen, C. Kang, H. Deng, C. Nobata, J.-M. Langlois, and Y. Chang. (2016). "Ranking Relevance in Yahoo Search". In: *Proceedings of the 22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Yue, Y., J. Broder, R. Kleinberg, and T. Joachims. (2012). "The k-armed Dueling Bandits Problem". *Journal of Computer and System Sciences*. 78(5): 1538–1556.

Yue, Y. and T. Joachims. (2009). "Interactively Optimizing Information Retrieval Systems as a Dueling Bandits Problem". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. 1201–1208.

Zamani, H., M. Bendersky, D. Metzler, H. Zhuang, and M. Najork. (2022). "Stochastic Retrieval-Conditioned Reranking". In: *Proceedings of the 2022 ACM SIGIR International Conference on the Theory of Information Retrieval*. Madrid, Spain.

Zhan, J., J. Mao, Y. Liu, J. Guo, M. Zhang, and S. Ma. (2021). "Jointly Optimizing Query Encoder and Product Quantization to Improve Retrieval Performance". In: *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*. Virtual Event, Queensland, Australia. 2487–2496.

Zhan, J., J. Mao, Y. Liu, J. Guo, M. Zhang, and S. Ma. (2022). "Learning Discrete Representations via Constrained Clustering for Effective and Efficient Dense Retrieval". In: *Proceedings of the 15th ACM International Conference on Web Search and Data Mining*. Virtual Event, AZ, USA. 1328–1336.

Zhan, J., J. Mao, Y. Liu, M. Zhang, and S. Ma. (2020). "RepBERT: Contextualized Text Embeddings for First-Stage Retrieval". arXiv: 2006.15498 [cs.IR].

Zhang, Y., C. Hu, Y. Liu, H. Fang, and J. Lin. (2021). "Learning to Rank in the Age of Muppets: Effectiveness–Efficiency Tradeoffs in Multi-Stage Ranking". In: *Proceedings of the 2nd Workshop on Simple and Efficient Natural Language Processing*. 64–73.

Zhao, W. X., J. Liu, R. Ren, and J.-R. Wen. (2022). "Dense Text Retrieval based on Pretrained Language Models: A Survey". arXiv: 2211.14876 [cs.IR].

Zheng, Z., H. Zha, T. Zhang, O. Chapelle, K. Chen, and G. Sun. (2008). "A General Boosting Method and its Application to Learning Ranking Functions for Web Search". In: *Advances in Neural Information Processing Systems*. 1697–1704.

Zhuang, H., Z. Qin, S. Han, X. Wang, M. Bendersky, and M. Najork. (2021). "Ensemble Distillation for BERT-Based Ranking Models". In: *Proceedings of the 2021 ACM SIGIR International Conference on Theory of Information Retrieval*. Virtual Event, Canada. 131–136.

Zhuang, S. and G. Zuccon. (2021). "TILDE: Term Independent Likelihood MoDEl for Passage Re-Ranking". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Virtual Event, Canada. 1483–1492.

Zhuang, S. and G. Zuccon. (2022). "Fast Passage Re-ranking with Contextualized Exact Term Matching and Efficient Passage Expansion". In: *Workshop on Reaching Efficiency in Neural Information Retrieval, the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*.