Feature Extraction for Large-Scale Text Collections

Luke Gallagher
RMIT University
luke.gallagher@rmit.edu.au

Antonio Mallia
New York University
antonio.mallia@nyu.edu

J. Shane Culpepper
RMIT University
shane.culpepper@rmit.edu.au

Torsten Suel
New York University
torsten.suel@nyu.edu

B. Barla Cambazoglu
RMIT University
barla.cambazoglu@rmit.edu.au

ABSTRACT

Feature engineering is a fundamental but poorly documented component in Learning-to-Rank (LTR) search engines. Such features are commonly used to construct learning models for web and product search engines, recommender systems, and question-answering tasks. In each of these domains, there is a growing interest in the creation of open-access test collections that promote reproducible research. However, there are still few open-source software packages capable of extracting high-quality machine learning features from large text collections. Instead, most feature-based LTR research relies on “canned” test collections, which often do not expose critical details about the underlying collection or implementation details of the extracted features. Both of these are crucial to collection creation and deployment of a search engine into production. So in this regard, the experiments are rarely reproducible with new features or collections, or helpful for companies wishing to deploy LTR systems.

In this paper, we introduce Fxt, an open-source framework to perform efficient and scalable feature extraction. Fxt can easily be integrated into complex, high-performance software applications to help solve a wide variety of text-based machine learning problems. To demonstrate the software’s utility, we build and document a reproducible feature extraction pipeline and show how to recreate several common LTR experiments using the ClueWeb09B collection. Researchers and practitioners can benefit from Fxt to extend their machine learning pipelines for various text-based retrieval tasks, and learn how some static document features and query-specific features are implemented.

ACM Reference Format:
Luke Gallagher, Antonio Mallia, J. Shane Culpepper, Torsten Suel, and B. Barla Cambazoglu. 2020. Feature Extraction for Large-Scale Text Collections. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM ’20), October 19–23, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3340531.3412773

1 INTRODUCTION

Features derived from text have been a core component of research in machine learning for many years, and are widely applied in natural language processing, information retrieval, data mining, and text analytics. Classic text-based features derived from cosine similarity and TF-IDF based measures, which capture term clustering effects in a document or passage w.r.t. how common a term is in the entire collection, have been used pervasively in the IR and NLP research communities.

In recent years, a number of benchmark collections have been made available in domains such as Learning-to-Rank (LTR) [9, 28, 33], recommender systems [31], question answering [3, 14], passage re-ranking [3], and conversational search [19, 34, 39]. These collections provide pre-computed features or pre-selected candidate documents, and greatly simplify the research community’s ability to focus on creating new machine learning algorithms without having to worry about feature engineering problems, or managing and processing massive text collections. However, these “canned” collections can also limit our ability to reproduce the experiments with new features or test data as several of the most widely used collections used in IR do not clearly define the queries, features, or the collections used to create them. This can greatly limit our ability to build end-to-end prototype systems, to perform ablation studies for *interpretable machine learning* [38], or even to create new test collections that aim to capture a similar environment.

The focus of this work is to present a new open source feature extraction toolkit—Fxt. We release this toolkit free and open for use by the research community.[1] One key aim of Fxt is to fill the gap within the research and open-source communities and help improve scalable, text-based machine learning system design. Fxt consists of several carefully engineered components. A fast, scalable indexing engine for the efficient storage and retrieval of text documents, passages, or text snippets; an efficient feature extraction API; and a configurable collection of 448 features engineered for a variety of text-based machine learning tasks, which can easily be extended to include custom features for other target domains. Fxt includes clean-room algorithmic implementations of several hundred features that have been shown to be valuable in LTR applications over the last twenty years. Fxt simplifies test collection construction, replicable research, and the study of end-to-end, large-scale inference pipelines. All of the algorithms can be readily used in production-quality search systems that are scalable and efficient.

CIKM ’20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3412773

[1] github.com/ten-blue-links/fxt

2 RELATED SOFTWARE

Many state-of-the-art algorithms rely on "classic" feature extraction techniques [16]. A wide variety of tree-based and neural machine learning models can benefit from classic feature extraction techniques to help with training data curation, weak supervision [13], or candidate generation [10] in tasks such as multi-hop question answering [42] or product search at Amazon and eBay. Yet no community-based open source repository of text-based feature extraction tools is currently available.

Most commercial search engines adopt a two-stage ranking architecture. In the first stage, a small subset of potentially good documents are selected by a simple yet fast ranking heuristic. In the second stage, the selected documents are re-ranked via a more sophisticated, machine-learned ranking model [25]. The sophisticated ranking models rely on a large number of features extracted from various sources. The number of features used by the learning models deployed in commercial search systems are known to be in the order of several hundreds. The most important features are usually the relevance features extracted from the query-document pair. Also, some query-independent features can be extracted from the document content (e.g., various features obtained by NLP techniques). These can be coupled with static document popularity or spam features that can be obtained from the hyperlink structure of the Web or various document classifiers as well as public datasets (e.g., Alexa's top sites dataset). Finally, if available, features extracted from user clicks on search results also play an important role in the quality of trained learning models.

Despite the common use of LTR systems in commercial search or recommendation settings, only a few systems are publicly available. Below we provide a brief overview of open-source systems available in the context of information retrieval. We note that all these systems are implemented in Java, while Fxt is implemented in C++, which may offer efficiency advantages.

ElasticSearch[2] is a popular open-source search platform which comes with an LTR plugin that can use learning models generated by XGBoost and Ranklib libraries. The plugin also provides an extendable framework for extraction of features from queries and documents. However, the plugin does not provide any implementation for specific features. That is, additional feature engineering and extraction is required to generate individual features used by the ranking model within the plugin.

Solr[3] is another commonly used open-source search platform that supports LTR in its default distribution. Additive decision tree ensembles and neural networks are among the supported learning models. It also provides a framework for feature extraction and storage. However, similar to ElasticSearch, no implementation is provided for individual features, i.e., the burden of feature engineering and extraction remains on the developer.

Anserini[4] [41] is an open-source information retrieval system whose development was motivated by reproducibility studies in IR [23]. This toolkit provides various re-ranking strategies as well as code to extract a number of features from queries and documents. The extracted features include certain length/size features, query-document similarity features, various query term statistics, and term proximity features. Compared to Anserini, our toolkit provides the implementations of a much larger set of features.

Terrier[5] [30] is another open-source information retrieval platform for rapid development and evaluation of large-scale retrieval applications. It features an extendable plugin architecture, where new ranking features can be easily added. An arbitrary number of static document features can be imported from an external text file. Also, a large number of relevance weighting models are implemented as part of this software. The feature extractors in Terrier are tightly coupled with the query processing pipeline, while our toolkit can be run in standalone mode.

3 FEATURE EXTRACTION TOOLKIT

Many current benchmark datasets used for state-of-the-art research omit certain details such as the candidate generation method, the query terms used, a detailed description of the features used, or the hyperparameters applied to each of the respective algorithms. These decisions are sometimes necessary—valid reasons may include the protection of user privacy, or company trade secrets. However from an academic viewpoint, such decisions can widen the gap between theory and practice, and hinder research progress.

3.1 Overview

The primary objective of Fxt is to provide a repository of high-quality text feature implementations which are scalable and easily integrated into many different search architectures. It is a plug-in component that can be used in a more complex retrieval pipeline, but it is also self-contained enough to be used independently to create new "canned" test collections by researchers who wish to focus on one aspect of an LTR system. The key benefit for research end-to-end prototyping is to not have to "reinvent the wheel" of feature engineering—an arduous task that prompted development of Fxt in the first place [11]. On the other hand, building "canned" collections with known features can be used for the advancement of LTR algorithms. Fxt is freely available for anyone to use, and is self contained enough to be framework agnostic. The toolkit focuses on two core components—*indexing* and *feature extraction.* Figure 1 shows how these two components are used by Fxt.

Implementation. The toolkit is written in C++, in order to maximize efficient and scalable feature extraction from large text corpora. The idea is that Fxt resides as a component within a larger multi-stage system that adheres to the Direct Index paradigm [1]. The approach relies on a *forward* index used to compute certain query-dependent features in a document-centric manner. To mitigate space issues, a highly compressible document vector format is used to encode intermediate document data.

Fxt is fully configurable and supports custom feature selection so that only features needed by an application are extracted from the collection. This functionality enables multiple, increasingly complex re-ranking stages to be deployed with multiple configurations, and minimizes computational cost (only features wanted are computed). That is, different features can be easily extracted and used in different stages of an LTR pipeline.

[2] elastic.co

[3] lucene.apache.org/solr

[4] github.com/castorini/anserini

[5] terrier.org

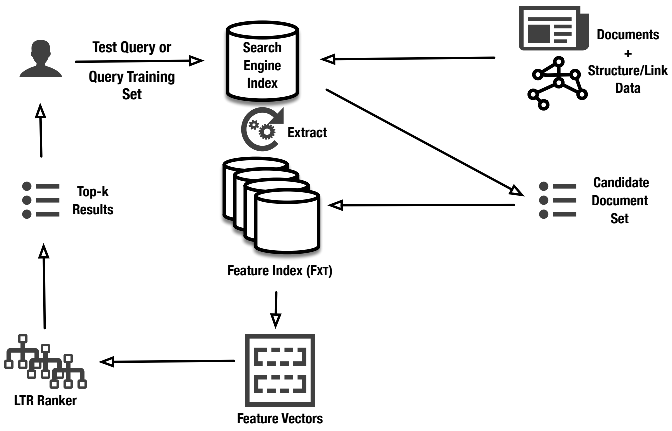

Figure 1: Query Lifecycle: A target query is submitted to a search engine (Indri). The search engine's index returns a candidate document set which is sent to Fxt, and based on the configuration, Fxt returns a feature vector. The feature vector can be processed by an LTR ranker such as LightGBM. A similar process can be used to train an LTR ranker by instead submitting a batch set of labeled training queries.

.

3.2 Indexing

The role of the indexing subsystem is to separate the feature repository requirements from the query processing pipeline, providing greater flexibility to search engine developers in accessing features with minimal changes in system architecture. Fxt currently uses an existing Indri[6] index to construct the feature repository. Our current experiments use Indri for the initial parsing and indexing, which allows us to focus on more of the domain specific problems for the feature extraction toolkit. However, any search engine can be used to generate the initial index, and we are currently working to support the *common index file format* (CIFF) [24], which will allow Fxt to be integrated into any search engine supporting the CIFF format.

The `indexer` reads from an Indri index and generates an index that is optimized for on-the-fly feature extraction and in-memory query processing operations. The Fxt index contains common data structures (lexicon, postings), along with structures to support more complex feature extraction operations efficiently—document vector forward index, static document features and document field information. Compression is used for the postings and document vector data structures. Posting lists are encoded with SIMD-FastPFOR [21]. Document vectors are encoded with StreamVByte [22]. We use the FastPFor[7] library for both implementations.

3.3 Feature Extraction

The primary goal of the framework is provide efficient and effective feature extraction from queries, documents, and query-document pairs. Feature extraction is handled by the `extractor` program. Figure 1 shows a typical use case. The `extractor` takes as input, a set of queries, their corresponding documents as a TREC run file, a path to the index, and an optional feature configuration file to construct a training set. Once the pipeline has been initiated, a similar process is used to execute a single query. The query processing stage extracts features for each query-document pair in the intermediate candidate set. A configuration file or command line invocation can be used to define the features that should be included in a feature vector. The resulting intermediate feature vector output is then returned in CSV format. This output format is the preferred intermediate format in many publicly available machine learning frameworks, and can be converted directly into another format, such as SVMLight or NPZ formats. We are also investigating how to support a CIFF extension that can easily be integrated into any search engine architecture.

Table 1: Summary of features available in Fxt.

Description	No. Features
Term Score Aggregation (Unigram)	159
Term Score Aggregation (Bigram)	147
Query Document Score (Unigram)	106
Query Document Score (Bigram)	4
Static Document Quality	19
Query (Document Independent)	13
Total	448

3.4 Features

There are already a large number of query, document and query-document features provided by Fxt. A total of 448 are available, while more features will be added over time. Table 1 summarizes the taxonomy of feature types that are currently implemented.[8]

We now broadly outline some features from these classes and their origins within the literature. Table 2 is used for this purpose. A large portion of the static document features are from a study conducted by Bendersky et al. [4]. Other static document features originate from the seminal work of Liu [25]—for example counts of inlinks/outlinks and other elements within a document. The term score aggregation features are mainly derived from the work of Culpepper et al. [12]. These are closely related to other query-performance predictors surveyed by Carmel and Yom-Tov [5]. Currently, a small number of these query-only features are available for use.

In terms of scoring features, i.e., those that perform term matching between the query and document—many were derived from the documentation of LTR collections such as Yahoo! [9] and LETOR [33]. The query-document scoring features include popular methods such as BM25, QL, and DFR. If field information is available, feature extraction can be performed on segments such as the document *title* and *inlink*. A small number of proximity-based scoring functions that target bigram matching between the query and document are implemented, the most notable of which

[6] lemurproject.org/indri
[7] github.com/lemire/FastPFor

[8] See the full list of 448 features that are currently in the toolkit at github.com/ten-blue-links/fxt

Table 2: Summary of prior work describing LTR features implemented in Fxt.

Name	Feature Type	Reference	Description
Stop Cover/Ratio	Static Document	Bendersky et al. [4]	Percentage of stop words in a candidate document.
Link Count	Static Document	Liu [25]	Number of inlinks/outlinks pointing to/from a candidate document.
Query Difficulty	Query Specific	Culpepper et al. [12]	Commonly used pre-retrieval query performance prediction features such as the average maximum score of the query terms.
Count-based	Query-Document	Qin and Liu [33]	Query-document count statistics of *sum*, *min*, *mean*, *max*, *var* over whole document and document fields.
Score-based	Query-Document	Liu [25]	Unigram retrieval models over whole document and document fields.

Table 3: Summary of test and training sets within the dataset.

Test Queries	Training/Validation Queries
WT09	MQ09
WT10	Web Tracks 2009, 2011, 2012
WT11	Web Tracks 2009, 2010, 2012
WT12	Web Tracks 2009, 2010, 2011

is SDM [32]. Much of the inspiration for implementing the query-document features comes from open and accessible descriptions of features found in the LTR literature [25, 29, 30, 33]. With the growing collection of available features across the different feature classes there is potential for this toolkit to be applied in many areas beyond IR-based LTR.

4 LEARNING-TO-RANK DATASET

In this section, we show how to use Fxt to create a LTR dataset using the ClueWeb09B collection. Relevant scripts and instructions to reproduce the data files can be found at github.com/ten-blue-links/cikm20.

4.1 Dataset Construction

There are four datasets available—one for each of the Web Tracks held through 2009–2012. Table 3 shows the query set combinations used in our experiments. We now describe each of these in detail.

Collection and Queries. In our experiments, we use the ClueWeb09B collection, which contains 50,220,423 documents. This collection is the result of a large web crawl performed in 2009. We make use of query sets from the 2009 Million Query Track (MQ09), and the 2009–2012 Web Tracks (WT09–WT12).

Judgments and Evaluation. Human judgments from the MQ09 and WT09 have several important differences from the Web Tracks which ran between 2010 and 2012. In 2009, relevance labels were created using a shallow, sampling-based document pool, which was devised in order to get judgments for as many queries as possible (the Million Query Track). As it was the first year that NIST had used the new ClueWeb collection, along with the return of the Web Track, it was decided that the WT09 would be a subset of MQ09 in order to avoid duplicated judgment efforts [7] (i.e. 20001–20050, and 1–50 are equivalent).

Another important issue to consider when aggregating data across the Web Tracks of ClueWeb09B is that the relevance grades used during human adjudication differ from year to year, with the exception of WT10 and WT11, which have the same relevance grading scheme. We discuss the implications of these differences in more detail in Section 5.1 when we show how to perform the final effectiveness comparisons.

Web Track 2009. As alluded to earlier, the way in which the training queries are constructed for WT09 is handled differently to the Web Tracks of 2010–2012. The key difference is that the MQ09 query set is used for training, as the same human assessment process was used for both MQ09 and WT09 [7].

There are 687 judged queries from MQ09, but the first 50 queries (and judgments) are also used in WT09. Therefore, we omit these topics from the training stage. This leaves 637 queries in the training set. These are then randomly shuffled and split into training and validation sets of 572 and 64 queries respectively (i.e. a 90/10 split).[9] The primary reason to use MQ09 for training in this case, was that the relevance grades are consistent with WT09, while this is not true for the remaining Web Tracks which we discuss next.

Web Tracks 2010–2012. Table 3 summarizes the query sets used which are from the four Web Tracks of 2009–2012. The WT09 queries were included here as the number of judgments and the pooling depth are similar to the query sets from 2010–2012 albeit through a PREL process. An amendment was made to the training queries by removing topic 70 "`to be or not to be that is the question`" because the query is computationally expensive. The training queries for each year were randomly shuffled into sets of 120 and 30 for training and validation respectively.

Index Configuration. An index was constructed using Indri with Krovetz stemming and with no stopping applied. Then an Fxt index was built using the Indri index as input. A common practice on the

[9] There is one less query in the training data since the topic 20705 "choreathetosi" returned no results.

ClueWeb09B collection is the removal of SPAM documents [18]. This was not performed initially as it may be applied as a processing step during retrieval.

Document Sampling. To sample documents for the re-ranking task, we ran Indri with a BM25 retrieval across all queries using parameters $k_1 = 0.9$ and $b = 0.4$. The retrieval depth was set to 2,000, which was previously suggested by Macdonald et al. [29] during a similar LTR exercise. For each of the test sets (50 topics), we use this as our initial sample, the aim being to reflect a candidate generation phase that would likely be seen in a production environment. The training samples were retrieved in a similar manner, but also included all of the judged documents in the QRELs. This ensures that all viable query-document pairs are used by the training sets and generally improves the final effectiveness of the learned models.

4.2 Features

We now provide an overview of the 134 features that are included in the dataset. A subset of features was chosen as some of the features offered by Fxt may not be useful for the ClueWeb web collections. For example the term score aggregation and query features from Table 1 were not used because they are less useful in a post-retrieval context [35]. The exact details of the features and their parameters can be found in the publicly available repository already discussed. In the following, we discuss the features categorized as they appear in Table 1.

Query-Document Unigram. These features consist of common scoring functions including BM25, Query Likelihood, TF-IDF, BB2, DPH and DFR—the last three are from the Divergence From Randomness family of models [20]. Each of the scoring functions were computed for the following document fields: whole document, body, title, heading, anchor and inlink.

Various frequency statistics were computed for the query document pairs as well. These features were originally described in the MSLR LETOR datasets [33]. They include of the sum, min, max, mean, and variance arithmetic functions, and are also computed over document fields.

Query-Document Bigram. Proximity based methods are a useful feature for ranking functions that compute the distance between a set of terms within a document. However, they can be computationally expensive, and Robertson and Zaragoza [36] suggest this can be somewhat alleviated by restricting methods to use bigrams only. The methods we provide as bigram features for this are SDM [32], and BM25-TP [26].

Static Document Quality. There are many ways to characterize document quality. It can be something as simple as the length of a given document field. In this dataset, we compute simple statistics like this over the title, URL and whole document fields.

Other features of interest and available as part of the dataset come from other prior work. One of these is the fraction of text within a document field relative to the rest of the document. The fields for this feature include the anchor and table fields, and an aggregate version that combines multiple fields. Turning to stop words, features that use this information are the *stop ratio*, and *stop cover*. The stop ratio is the ratio of stop words to non-stop words, while stop cover can be described as the fraction of stop words relative to the document. The interested reader may find more details from the prior work in this area [4, 33].

Simple link statistics are available and include the number of inlinks and outlinks within the document. Accompanying these are domain level features that are often used as a source for evaluating a document's quality. These features are the AlexaRank score and whether or not the document originated from Wikipedia. The Wayback Machine was used to attain AlexaRank data that is temporally close to the time of the ClueWeb09B crawl.[10]

5 EXPERIMENTS

The theme of this section is to empirically analyze the LTR dataset described in Section 4. The relevant details regarding the collection, queries and index configurations were discussed in Section 4.1. The code and data to reproduce the experiments can be found in the GitHub repository.[11]

5.1 Evaluation

We compare the effectiveness using three common evaluation metrics—RBP with persistence $p = 0.9$, NDCG at cut-offs $\{5, 20\}$, and AP. The effectiveness scores were computed using `rbp_eval`,[12] `gdeval.pl`,[13] and `trec_eval`,[14] each of which represents the original implementation for the respective measure. Early precision metrics were favored due to the nature of the collection and judgment pooling depth. They are also more common when using LTR algorithms, where typically only a few of the highest ranking documents are required for most users. We also report AP since one of the baselines is the initial candidate set, which tends to be much deeper than the final top-k document set returned to a user.

Note that each year of the Web Track (50 topics each year) were evaluated separately. Results are reported in this way since the relevance grades over the four years the track ran on ClueWeb09B were not consistent. For clarity, the relevance grades were $\{0, 1, 2\}$ in 2009; 2010 and 2011 used the same relevance grades $\{-2, 0, 1, 2, 3\}$; and in 2012 relevance levels changed again having $\{-2, 0, 1, 2, 3, 4\}$.

Recall from the previous discussion that judgments in WT09 did not use the traditional TREC pooling and assessment methodology. The judgment process for topics 1–50 of the WT09 were shared with the MQ09 track. The Million Query Track used two document selection algorithms designed to aid the judgment process over large sets of queries. Due to the statistical estimation strategies of the document selection algorithms MTC [6] and statAP [2], the judgments resulting from this process are known as *PRELs*. That is, the judgment file includes for each entry, the *probability of inclusion*. This probability value was set to 1 when a judgment was deterministic, by an assessor, before any sampling was applied to select additional documents for assessment [2]. Evaluation comparisons using WT09 should be interpreted cautiously given the differences in judgments. For a more detailed discussion on evaluation and pooling depth pitfalls when using the 2009 ClueWeb collection, see Lu et al. [27].

[10] archive.org/web

[11] github.com/ten-blue-links/cikm20

[12] people.eng.unimelb.edu.au/ammoffat/rbp_eval-0.2.tar.gz

[13] github.com/trec-web/trec-web-2013

[14] trec.nist.gov/trec_eval/trec_eval-9.0.7.tar.gz

Before evaluating the results reported for WT09, all judgments that do not have an inclusion probability of 1 were removed from the Category B PREL file.[15] This ensures that we are correctly evaluating systems on the judgments made that were part of the original pooling depth of 12 for WT09, before any sampled documents were judged and included in the final set. The resulting QREL file can be found along with the released dataset.

5.2 Experimental Setup

LambdaMART. We use the LambdaMART implementation found in the LightGBM framework.[16] Model hyperparameters were optimized using grid search to select the best combination of learning rate $\eta = \{0.05, 0.06, \ldots, 0.1\}$, number of leaves $\{16, 32, 64\}$, feature sub-sampling rate $\{0.5, 0.6, \ldots, 1.0\}$, and the minimum data per leaf $\{13, 20, 100\}$. Given the relatively small size of the query sets, we applied early stopping after 40 iterations. Table 4 lists system parameters selected for each Web Track task.

BM25. The BM25 ranking function is used as a bag-of-words baseline, and it is often deployed for candidate generation within large-scale search systems. The parameters k_1 and b were chosen via grid search over the training queries with $k_1 = \{0.5, 0.6, \ldots, 2.0\}$, and $b = \{0.1, 0.2, \ldots, 1.0\}$.

SDM. This retrieval model was used as a baseline as it is known to be more effective than a simple bag-of-words ranking. Additionally, variants of SDM are often included as a feature within re-ranking datasets, and so the expectation is that LTR models including this as a feature should consistently outperform SDM alone. The parameters for SDM were also selected using grid search, but were performed in two stages. First, the feature weights for independent terms w_t, ordered terms w_o, and unordered terms w_u were enumerated from the following set of tuples $\{(w'_t, w'_o, w'_u) \in S \times S \times S \mid w'_t + w'_o + w'_u = 1.0\}$, where the step set $S = \{0.1, 0.2, \ldots, 1.0\}$. The feature weights selected from this search were then used to tune smoothing parameters for both terms and phrases (μ_t and μ_p). The Cartesian product of the set $\{500, 1000, 1500, 2000, 2500, 3000, 3500, 4000\}$ combined with itself was used as the search space.

TREC Best. To provide some intuition about where an upper bound on effectiveness for each test collection might be, we select the best performing TREC submission from each of the Web Tracks reported. "Best" systems were selected based on NDCG@20 scores of the original track submissions.

5.3 Results Discussion

Effectiveness results are shown in Table 5. Overall, there is a similar trend across the four sets of queries. The systems BM25 and SDM are less effective than LambdaMART, and in turn, the (pooled) TREC systems are the most effective. These results are somewhat expected as the baseline systems are also features available to the LTR model. LambdaMART has an advantage, it is a *supervised* learning algorithm, while the baselines are of course *unsupervised.*

When compared against the best TREC submissions, we see that LambdaMART can attain a higher AP score in three of the four query sets. However, only two of the years show statistical significance. There is another interesting observation for WT10. The NDCG@5 score for LambdaMART is the top performing result. However, for the other metrics the system irra10b is clearly superior for RBP and NDCG@20, while AP shows no noticeable difference. As our goal is simply to provide a toolkit to enable such comparisons, we leave deeper comparisons of strengths and weaknesses of common LTR algorithms as future work.

Another aspect of interest is the absence of statistical significance on the results shown for WT11. It is not clear why this is the case, but less variance between the average scores was observed when doing a query by query qualitative comparison. This may suggest that this set of topics are more difficult, or there is insufficient signals being learned from the training set for these queries. This also warrants further investigation.

5.4 Feature Analysis

We now turn our investigation to the features used by the LambdaMART models. Figure 2 shows four graphs, one for each year of the Web Tracks discussed. Shown within each graph are the 10 most important features from LambdaMART. Feature importance is calculated as the number of times a feature is used within the model. This provides some intuition as to how useful a feature may be and could also be a useful aid when explaining model behavior.

There are some commonalities between the features used across all of the graphs shown. We can see that the feature `lm_dir_2500_title`, which is the Query Likelihood model over the title field, is one of the most informative features in all cases. Furthermore, other Query Likelihood features appear multiple times throughout these results. These are interesting outcomes, and warrant further investigation to determine if this trend applies more generally across different query sets and collections, or whether it is localized to ClueWeb09B.

When focusing only on static document features, we see that stop cover consistently performs well in all of the collections tested. This may be related to the fact that the ClueWeb collection is now known to contain a large number of spam documents, and this feature is a strong signal for models to prune out low quality documents. The AlexaRank feature also performs well in three of the collections (2010–2012) and shows that for this particular dataset, traffic analysis is also a simple but useful feature to LTR models on web collections.

6 CONCLUSION

In this paper, we have presented Fxt, a standalone text indexing and feature extraction component useful for end-to-end systems prototyping, dataset construction, and feature extraction. We also presented a detailed walk-through on how the toolkit can be used to create a custom LTR dataset using the ClueWeb09B collection.

There are a number of interesting opportunities for academic and industry-based researchers to use Fxt in their future work. The toolset has slowly evolved over the last three years as our research efforts became more heavily invested in applying machine learning in IR, and will continue to improve as these techniques are now consistently being applied successfully to solve a wide variety of problems in the IR community.

[15] trec.nist.gov/data/web/09/prels.catB.1-50.gz

[16] lightgbm.readthedocs.io

Table 4: Summary of baselines and learning-to-rank parameters for each of the Web Tracks 2009–2012.

Track	System	Parameters
WT09	BM25	$k_1 = 2.0$, $b = 0.4$.
	SDM	Smoothing $\mu_t = 4000$, $\mu_p = 2000$. Feature weights $w_t = 0.3$, $w_o = 0.1$, $w_u = 0.6$.
	LambdaMART	99 trees, 64 leaves, $\eta = 0.07$, feature sub-sample 1.0 and min sample per leaf 20.
WT10	BM25	$k_1 = 1.6$, $b = 0.2$.
	SDM	Smoothing $\mu_t = 4000$, $\mu_p = 2000$. Feature weights $w_t = 0.3$, $w_o = 0.1$, $w_u = 0.6$.
	LambdaMART	101 trees, 16 leaves, $\eta = 0.1$, column sample 1.0 and min sample per leaf 13.
WT11	BM25	$k_1 = 1.2$, $b = 0.1$.
	SDM	Smoothing $\mu_t = 4000$, $\mu_p = 2000$. Feature weights $w_t = 0.6$, $w_o = 0.1$, $w_u = 0.3$.
	LambdaMART	89 trees, 16 leaves, $\eta = 0.08$, column sample 1.0 and min sample per leaf 100.
WT12	BM25	$k_1 = 1.8$, $b = 0.4$.
	SDM	Smoothing $\mu_t = 4000$, $\mu_p = 2000$. Feature weights $w_t = 0.6$, $w_o = 0.2$, $w_u = 0.2$.
	LambdaMART	31 trees, 64 leaves, $\eta = 0.1$, column sample 1.0 and min sample per leaf 100.

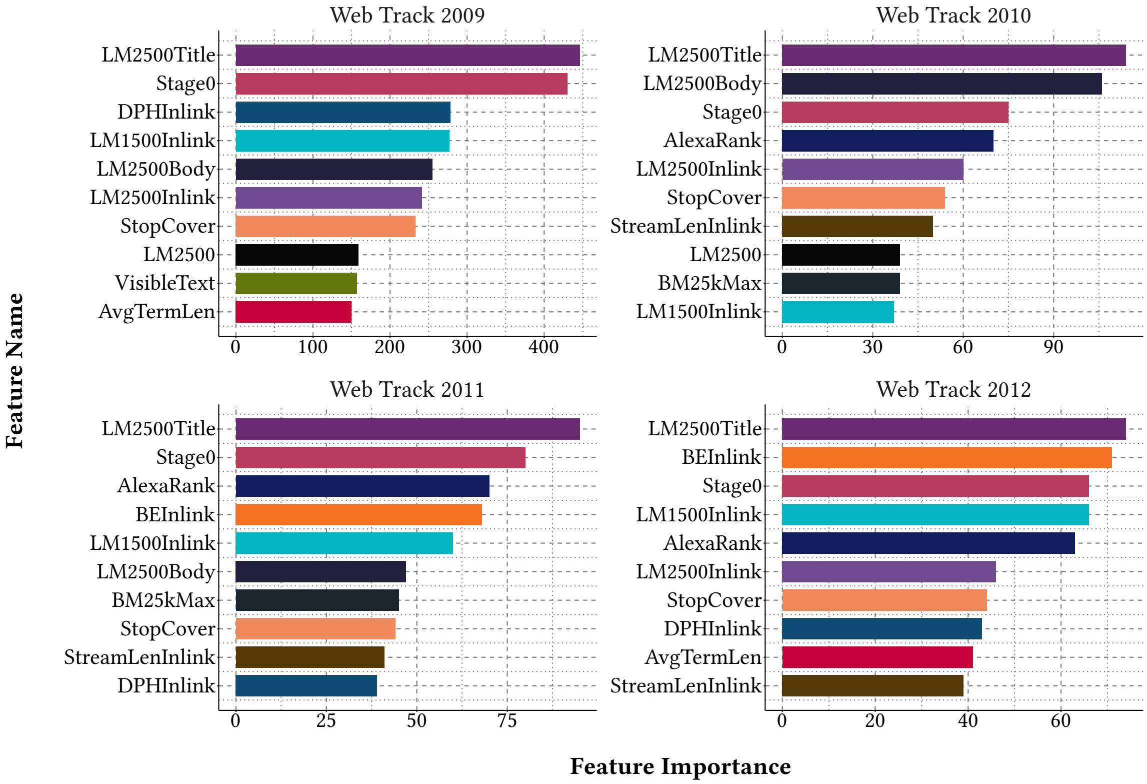

Figure 2: Feature importance of the four LambdaMART models through 2009–2012. The top 10 most important features were selected from each model, and therefore the set of features shown may differ between them.

Planned future improvements to Fxt include the addition of a benchmark tool to measure the computational cost of feature extraction so that research exploring efficiency and effectiveness trade-offs, such as in cascade ranking [17, 40], can be more accessible to new researchers in the community. Another interesting line of work involves adding passage level feature extraction, as they are useful for passage-based retrieval and in document retrieval [8, 15, 37]. Ablation studies over many more test collections could provide additional insights into interactions between queries and document collections and ranking model choices. Fxt will ideally extend and improve everyone's ability to more confidently undertake much deeper and extensive comparative studies of different LTR models within a controlled environment.

Table 5: Effectiveness scores for Web Tracks 2009–2012. A † indicates statistical significance with Bonferroni correction at $p < 0.05$ compared to BM25.

System	RBP 0.9	NDCG@5	NDCG@20	AP
ClueWeb 2009 Topics 1-50				
BM25	0.204 +0.307	0.193	0.202	0.162
SDM	0.229 +0.264	0.222	0.228	0.189
LambdaMART	0.286 +0.344†	0.298†	0.296†	0.219†
uogTrdphCEwP	0.298 +0.355†	0.336†	0.302†	0.200
ClueWeb 2010 Topics 51-100				
BM25	0.098 +0.186	0.100	0.120	0.102
SDM	0.091 +0.118	0.098	0.115	0.095
LambdaMART	0.187 +0.295†	0.224†	0.245†	0.131†
irra10b	0.199 +0.043†	0.193†	0.260†	0.133
ClueWeb 2011 Topics 101-150				
BM25	0.122 +0.170	0.182	0.176	0.097
SDM	0.126 +0.072	0.180	0.189	0.117
LambdaMART	0.132 +0.139	0.235	0.199	0.117
srchvrs11b	0.154 +0.031	0.271	0.233	0.110
ClueWeb 2012 Topics 151-200				
BM25	0.102 +0.217	0.087	0.102	0.104
SDM	0.104 +0.161	0.087	0.101	0.113
LambdaMART	0.193 +0.185†	0.193†	0.189†	0.164†
DFalah121A	0.220 +0.069†	0.198†	0.213†	0.120

ACKNOWLEDGEMENTS

This work was supported by the Australian Research Council's *Discovery Projects* Scheme (DP190101113), the NSF Grant IIS-1718680, an Amazon Research Award, and an Australian Government Research Training Program Scholarship.

REFERENCES

[1] N. Asadi and J. Lin. 2013. Document Vector Representations for Feature Extraction in Multi-Stage Document Ranking. *Inf. Retr.* 16, 6 (2013), 747–768.

[2] J. Aslam and V. Pavlu. 2007. *A Practical Sampling Strategy for Efficient Retrieval Evaluation.* Technical Report. Northeastern University.

[3] P. Bajaj, D. Campos, N. Craswell, L. Deng, J. Gao, X. Liu, R. Majumder, A. McNamara, B. Mitra, T. Nguyen, M. Rosenberg, X. Song, A. Stoica, S. Tiwary, and T. Wang. 2016. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. *arXiv preprint arXiv:1611.09268* (2016).

[4] M. Bendersky, W. B. Croft, and Y. Diao. 2011. Quality-Biased Ranking of Web Documents. In *Proc. WSDM.* 95–104.

[5] D. Carmel and E. Yom-Tov. 2010. *Estimating the Query Difficulty for Information Retrieval.* Morgan & Claypool.

[6] B. Carterette, J. Allan, and R. Sitaraman. 2006. Minimal Test Collections for Retrieval Evaluation. In *Proc. SIGIR.* 268–275.

[7] B. Carterette, V. Pavlu, H. Fang, and E. Kanoulas. 2009. Million Query Track 2009 Overview. In *Proc. TREC.*

[8] M. Catena, O. Frieder, C. I. Muntean, F. M. Nardini, R. Perego, and N. Tonellotto. 2019. Enhanced News Retrieval: Passages Lead the Way!. In *Proc. SIGIR.* 1269–1272.

[9] O. Chapelle and Y. Chang. 2011. Yahoo! Learning to Rank Challenge Overview. *J. Mach. Learn. Res.* 14 (2011), 1–24.

[10] D. Chen, A. Fisch, J. Weston, and A. Bordes. 2017. Reading Wikipedia to Answer Open-Domain Questions. In *Proc. ACL.* 1870–1879.

[11] R-C. Chen, L. Gallagher, R. Blanco, and J. S. Culpepper. 2017. Efficient Cost-Aware Cascade Ranking in Multi-Stage Retrieval. In *Proc. SIGIR.* 445–454.

[12] J. S. Culpepper, C. L. A. Clarke, and J. Lin. 2016. Dynamic Cutoff Prediction in Multi-Stage Retrieval Systems. In *Proc. ADCS.* 17–24.

[13] M. Dehghani, H. Zamani, A. Severyn, J. Kamps, and W. B. Croft. 2017. Neural Ranking Models with Weak Supervision. In *Proc. SIGIR.* 65–74.

[14] M. Dunn, L. Sagun, M. Higgins, V. Ugur Guney, V. Cirik, and K. Cho. 2017. SearchQA: A New Q&A Dataset Augmented with Context from a Search Engine. *arXiv preprint arXiv:1704.05179* (2017).

[15] Y. Fan, J. Guo, Y. Lan, J. Xu, C. Zhai, and X. Cheng. 2018. Modeling Diverse Relevance Patterns in Ad-hoc Retrieval. In *Proc. SIGIR.* 375–384.

[16] J. Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[17] L. Gallagher, R-C. Chen, R. Blanco, and J. S. Culpepper. 2019. Joint Optimization of Cascade Ranking Models. (2019), 15–23.

[18] L. Gallagher, J. Mackenzie, and J. S. Culpepper. 2018. Revisiting Spam Filtering in Web Search. In *Proc. ADCS.*

[19] J. Gao, M. Galley, and L. Li. 2018. Neural Approaches to Conversational AI. *arXiv preprint arXiv:1809.08267* (2018).

[20] B. He and I. Ounis. 2005. Term Frequency Normalisation Tuning for BM25 and DFR Models. In *Advances in Information Retrieval.* 200–214.

[21] Daniel Lemire and Leonid Boytsov. 2015. Decoding billions of integers per second through vectorization. *Software: Practice and Experience* 45, 1 (2015), 1–29.

[22] Daniel Lemire, Nathan Kurz, and Christoph Rupp. 2018. Stream VByte: Faster byte-oriented integer compression. *Inform. Process. Lett.* 130 (2018), 1–6.

[23] J. Lin, M. Crane, A. Trotman, J. Callan, I. Chattopadhyaya, J. Foley, G. Ingersoll, C. Macdonald, and S. Vigna. 2016. Toward Reproducible Baselines: The Open-Source IR Reproducibility Challenge. In *Proc. ECIR.* 408–420.

[24] J. Lin, J. Mackenzie, C. Kamphuis, C. Macdonald, A. Mallia, M. Siedlaczek, A. Trotman, and A. de Vries. 2020. Supporting Interoperability Between Open-Source Search Engines with the Common Index File Format. In *Proc. SIGIR.* To Appear.

[25] T-Y. Liu. 2009. Learning to Rank for Information Retrieval. *Found. Trends in Inf. Ret.* 3, 3 (2009), 225–331.

[26] X. Lu, A. Moffat, and J. S. Culpepper. 2015. On the Cost of Extracting Proximity Features for Term-Dependency Models. In *Proc. CIKM.* 293–302.

[27] X. Lu, A. Moffat, and J. S. Culpepper. 2016. The Effect of Pooling and Evaluation Depth on IR Metrics. *Inf. Retr.* 19, 4 (2016), 416–445.

[28] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, F. Silvestri, and S. Trani. 2016. Post-learning optimization of tree ensembles for efficient ranking. In *Proc. SIGIR.* 949–952.

[29] C. Macdonald, R. L. Santos, and I. Ounis. 2013. The Whens and Hows of Learning to Rank for Web Search. *Inf. Retr.* 16, 5 (2013), 584–628.

[30] C. Macdonald, R. L. Santos, I. Ounis, and B. He. 2013. About Learning Models with Multiple Query-Dependent Features. *ACM Trans. Information Systems* 31, 3 (2013), 11:1–11:39.

[31] J. McAuley and A. Yang. 2016. Addressing Complex and Subjective Product-Related Queries with Customer Reviews. In *Proc. WWW.* 625–635.

[32] D. Metzler and W. B. Croft. 2005. A Markov Random Field Model for Term Dependencies. In *Proc. SIGIR.* 472–479.

[33] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 Datasets. *arXiv preprint arXiv:1306:2597* (2013).

[34] F. Radlinski and N. Craswell. 2017. A Theoretical Framework for Conversational Search. In *Proc. CHIIR.* 117–126.

[35] F. Raiber and O. Kurland. 2014. Query-performance Prediction: Setting the Expectations Straight. In *Proc. SIGIR.* 13–22.

[36] S. E. Robertson and H. Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends in Inf. Ret.* 3 (2009), 333–389.

[37] E. Sheetrit, A. Shtok, and O. Kurland. 2019. A Passage-Based Approach to Learning to Rank Documents. *Inf. Retr.* 23, 2 (2019), 159–186.

[38] S. Sonnenburg, M. L. Braun, C. S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K-R. Müller, F. Pereira, C. E. Rasmussen, G. Rätsch, B. Schölkopf, A. Smola, P. Vincent, J. Weston, and R. Williamson. 2007. The Need for Open Source Software in Machine Learning. *J. Mach. Learn. Res.* 8 (2007), 2443–2466.

[39] P. Thomas, M. Czerwinski, D. McDuff, N. Craswell, and G. Mark. 2018. Style and Alignment in Information-Seeking Conversation. In *Proc. CHIIR.* 42–51.

[40] L. Wang, J. Lin, and D. Metzler. 2011. A Cascade Ranking Model for Efficient Ranked Retrieval. In *Proc. SIGIR.* 105–114.

[41] P. Yang, H. Fang, and J. Lin. 2018. Anserini: Reproducible Ranking Baselines using Lucene. *J. Data and Information Quality* 10, 4 (2018).

[42] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. W. Cohen, R. Salakhutdinov, and C. D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Proc. EMNLP.* 2369–2380.