Lerot: An Online Learning to Rank Framework

Anne Schuth
ISLA, University of Amsterdam
a.g.schuth@uva.nl

Katja Hofmann
Microsoft Research
Cambridge
katja.hofmann@microsoft.com

Shimon Whiteson
ISLA, University of Amsterdam
s.a.whiteson@uva.nl

Maarten de Rijke
ISLA, University of Amsterdam
derijke@uva.nl

ABSTRACT

Online learning to rank methods for IR allow retrieval systems to optimize their own performance directly from interactions with users via click feedback. In the software package *Lerot*, presented in this paper, we have bundled all ingredients needed for experimenting with online learning to rank for IR. Lerot includes several online learning algorithms, interleaving methods and a full suite of ways to evaluate these methods. In the absence of real users, the evaluation method bundled in the software package is based on simulations of users interacting with the search engine. The software presented here has been used to verify findings of over six papers at major information retrieval venues over the last few years.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: H.3.3 Information Search and Retrieval

Keywords

Information retrieval, experimentation, software, interleaved comparisons, learning to rank

1. INTRODUCTION

Adapting IR systems to a specific user, group of users, or deployment setting has become possible and popular due to *learning to rank* techniques [20]. Generally speaking, a learning to rank method learns the weights of a function that maps a document-query pair described by a feature vector to a value that is used to rank documents for a given query. We refer to such a function with instantiated weights as a *ranker*. Most current approaches learn *offline*, i.e., before deployment rankers are estimated from manually annotated training data.

In contrast, an *online* learning to rank method learns directly from interactions with users, e.g., using click feedback. For instance, the current state-of-the-art online learning to rank approach uses *dueling bandit gradient descent* (DBGD) [14, 25] to find a high quality ranker. In each step, the current best ranker is perturbed, and then both the original and perturbed rankers are compared using an *interleaved comparison method* [23]: the rankings proposed by the two rankers are interleaved and presented to the user, whose clicks determine which ranker wins the comparison. If the perturbed ranker wins, the original ranker is adjusted slightly in its direction.

Lerot, the framework presented in this paper, offers a solution for evaluating and experimenting with online learning to rank algorithms in living labs and simulations. Living labs represent a user-centric research methodology that seeks to test and evaluate emerging technologies in real-world contexts. Therefore, they form the ideal environment for prototyping and assessing online learning to rank methods. Lerot is designed to support such experiments with *online* learning to rank algorithms, or with components of such algorithms in a living lab setup. Lerot also offers the next best thing: simulations of users interacting with a search engine. In contrast to experiments run in a full-blown living lab environment, simulation experiments make it possible to generate a wide range of candidate result lists, without the risk of adversely affecting user experience in a production system. Thus, simulation experiments with Lerot may complement or precede experimentation in a living lab setup for online learning to rank.

While there are several other libraries and frameworks for learning to rank such as SVMRank[1] [18], RankLib[2] and Sofia-ml[3] [24], these all focus on *offline* learning to rank. By contrast, Lerot focuses on *online* learning to rank, implementing DBGD and extensions of DBGD such as candidate preselection (CPS) [13] in an easily decomposable fashion.

In this paper, we present all the components that are included in Lerot. The framework has *all batteries included* (except for the data), to replicate experiments; no code needs to be written.

Lerot and its predecessors have been used to verify the findings in numerous publications [4, 9–14] at major venues. The framework is easily extensible to compare the implemented methods to new online evaluation and online learning approaches.

2. FRAMEWORK

In broad terms, Lerot can be used to run two types of experiments: *learning experiments* and *evaluation experiments*. Learning experiments operate in a continuous space of possible solutions and evolve rankers over time to find the optimal one. Evaluation experiments, on the other hand, operate on a fixed set of rankers and are designed to identify the best ranker among this set using,

LivingLab'13, November 1, 2013, San Francisco, CA, USA.
ACM 978-1-4503-2412-0/13/11
http://dx.doi.org/10.1145/2513150.2513162.

[1] http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html
[2] http://people.cs.umass.edu/~vdang/ranklib.html
[3] https://code.google.com/p/sofia-ml/

Listing 1: Minimal example of an online learning experiment that uses a list wise learning algorithm and a cascade user model to simulate clicks.

```
import sys, random
import retrieval_system, environment, evaluation,
                                                query
learner = retrieval_system.ListwiseLearningSystem(
                                                [...])
user_model = environment.CascadeUserModel([...])
evaluation = evaluation.NdcgEval([...])
train = query.load_queries(sys.argv[1], [...])
test = query.load_queries(sys.argv[2], [...])
while True:
    q = train[random.choice(train.keys())]
    l = learner.get_ranked_list(q)
    c = user_model.get_clicks(l, q.get_labels())
    s = learner.update_solution(c)
    print evaluation.evaluate_all(s, test)
```

for instance, interleaved comparisons. This paper mostly focuses on describing the learning experiments.

A minimal example[4] of a learning algorithm embedded in a simulation with a user model is shown in Listing 1. The example defines a learner (see Section 2.1), a user model (see Section 2.3), an evaluation method (see Section 2.4), and lists of training and test queries with labels. If real users are available, they are the source of the training queries and the clicks. In their absence, the queries come from a dataset and the clicks from a click model that uses relevance judgements. The queries `q` are observed in a random order, a ranked list `l` is produced by the learner, this ranking is sent to the click model and the clicks `c` it produces, in turn, are observed by the learner so that it can update the solution. The updated solution `s` is then evaluated on the test queries. In theory, this process continues indefinitely.

2.1 Learning Algorithms

The `learner` in Listing 1 can be instantiated in many ways. Our framework has implementations for (1) learning from document-pairwise feedback [9, 17, 24, 26]; (2) learning from listwise feedback, such as dueling bandit gradient descent (DBGD) [25]; and (3) extensions of DBGD, such as candidate pre-selection (CPS) [13].

All these methods have the exact same interface; they implement three functions, two of which are called in Listing 1.

- `l = get_ranked_list(q)`
  Returns a list `l` of documents in response to query `q`.
- `s = update_solution(c)`
  Updates the current solution using clicks `c` and returns the updated solution `s`.
- `s = get_solution()`
  Returns the updated solution `s`.

Listing 2 shows the implementation of the learning algorithm used by DBGD.[5] During initialization (which we omitted from the listing) `self.ranker` is randomly initialized. When query `q` is observed, it arrives at `get_ranked_list()`. In DBDG this function creates a `candidate` ranker that is a variation of `self.ranker`. Both these

[4]This code mainly serves as an illustration (but is included in Lerot in `src/scripts/run-example.py`), a version that interprets the configuration files explained in Section 3.2 should be preferred.

[5]The actual implementation of DBGD included in Lerot is slightly more involved as it is configurable.

Listing 2: Listwise learning algorithm, that in combination with Listing 1 constitutes DBGD [25].

```
class ListwiseLearningSystem(AbstractLearningSystem):
    def __init__(self): [...]

    def get_ranked_list(self, q):
        u = utils.sample_unit_sphere()
        candidate = self.ranker + self.delta * u
        l, a = self.comparison.interleave(
                self.ranker, candidate, q, self.n)
        self.l, self.a, self.q, self.u = l, a, q, u
        return l

    def update_solution(self, c):
        o = self.comparison.infer_outcome(
                        self.l, self.a, c, self.q)
        if o > 0:
            self.ranker += self.alpha * self.u
        return self.ranker

    def get_solution(self):
        return self.ranker
```

rankers are given to an interleaved comparison method (see Section 2.2) and after storing all intermediate results, the interleaved list `l` is returned. As soon as the user interacts with this list, the clicks `c` arrive at the function `update_solution()`. This function delegates the computation of the outcome `o` of the interleaved comparison to the interleaving method. Based on `o`, `self.ranker` is updated towards the `candidate`.

2.2 Interleaved Comparison Methods

In recent years, several methods for *interleaved comparisons* have been developed. They can be viewed as online evaluation methods that can be applied—as opposed to TREC style evaluation—without manual labeling of relevant documents. Instead, the clicks of real users (or, in our case, simulated clicks) of the search engine are interpreted to compare two ranking algorithms. In the context of online learning, interleaved comparisons are mainly used to decide whether a candidate ranker is an improvement over the current best ranker or not. In comparison to absolute click metrics typically used in A/B testing, interleaved comparison methods reduce variance (briefly, this is because they perform within-subject as opposed to between-subject comparisons) [23], and make different assumptions about how clicks should be interpreted (as relative, as opposed to absolute feedback) [9].

In the Lerot framework, we have implemented the following interleaving methods: (1) balanced interleave (BI) [19, 23]; (2) team draft interleave (TD) [23]; (3) document constraints interleave (DC) [8]; (4) probabilistic interleave (PI) [11]; (5) optimized interleave (OI) [22]; and (6) vertical aware team draft interleave (TD-VA) [4].

These methods also have the exact same interface; they implement the two functions that are called in Listing 2.

- `l, a = interleave(A, B, q, n)`
  Returns an interleaved list of documents `l` with length `n` of rankings produced by systems `A` and `B` for query `q`. The return value `a` can be used to store, for instance, team assignments in the case TD is used.
- `o = infer_outcome(l, a, c, q)`
  Returns the outcome `o` in $(-1, 1)$ of the interleaved comparison based on clicks `c` for query `q`, and interleaved list `l`. If `o` < 0 then system `A` wins the comparison, else if `o` > 0 then `B` wins the comparison, otherwise the systems tie.

The PI method requires the ranking systems to be probabilistic; the others expect a deterministic ranker. The TD-VA method requires documents to be annotated with the vertical to which they belong.

2.3 User Models

A user model is used to simulate user's clicking behavior. These models are aimed at predicting what users would click on given a result list with relevance judgments in response to a query. The models thus need datasets that are annotated with relevance. In Section 3.2 we list some datasets that are suited. We have implemented the following click models: (1) dependent click model (DCM) [6, 7], a generalization of the cascade click model [5]; (2) random click model (RCM); and (3) federated click model (FCM) [3], in particular the attention bias model.

These models implement the `user_model` in Listing 1 and, again, these models all have the exact same interface; they implement the following function.

- `c = get_clicks(l, r)`
  Returns a list of clicks `c` on documents in result list `l` given a list of relevance labels `r` for these documents.

Like TD-VA, the FCM requires documents to be annotated with their vertical. The click models only model the assumptions regarding how users examine result pages. They still have to be instantiated to match the situation that has to be simulated. For several instantiations of DCM, see [9] and for instantiations of FCM, see [4].

2.4 Evaluation

Evaluation can be done both online and offline. Online evaluation measures what a user experiences; i.e., the quality of interleaved lists that the user (or user model) interacts with. It is measured as a discounted sum over time. Offline evaluation measures how the current best ranker would perform on a held-out dataset. Implemented metrics are NDCG [1] and a slight variation LetorNDCG [15]. Listing 1 illustrates how and when offline metrics would be calculated in a learning setting. Extending Lerot with more metrics is a matter of creating a new class that implements the following two functions.

- `score = evaluate_ranking(l, q)`
  Returns a `score` for the ranking `l` with respect to query `q`.
- `mean_score = evaluate_all(s, queries)`
  Returns the `mean_score` for all `queries` if they where ranked with solution `s`.

3. IMPLEMENTATION

Lerot is implemented in python and consist of several packages (retrieval_system, comparison, evaluation, etc). Each packages has an abstract class that defines the expected interface (as described in Section 2) of the classes that implement it. Extending the framework is a matter of implementing such a class and changing the configuration file (see Section 3.2) to point to the new class. Lerot is available under the GNU Lesser General Public License.

3.1 Installation

Prerequisities for Lerot are:

- Python 2.7;
- PyYaml;
- Numpy;
- Scipy;

Listing 3: Example configuration file for a learning experiment.

```
training_queries: data/MQ2007/Fold1/train.txt
test_queries: data/MQ2007/Fold1/test.txt
feature_count: 46
num_runs: 1
num_queries: 10
query_sampling_method: random
output_dir: outdir
output_prefix: Fold1
user_model: environment.CascadeUserModel
user_model_args:
    --p_click 0:0.0,1:0.5,2:1.0
    --p_stop 0:0.0,1:0.0,2:0.0
system: retrieval_system.ListwiseLearningSystem
system_args:
    --init_weights random
    --sample_weights sample_unit_sphere
    --comparison comparison.ProbabilisticInterleave
    --delta 0.1
    --alpha 0.01
    --ranker ranker.ProbabilisticRankingFunction
    --ranker_arg 3
    --ranker_tie random
evaluation:
    - evaluation.NdcgEval
```

- Celery (when MetaExperiment is used to distribute over several machines, see Section 3.3); and
- Gurobi[6] (when OI is used as the interleaved comparison method).

Once Python (and pip) have been installed, Lerot can be installed using these commands:

```
$ pip install PyYAML numpy scipy celery
$ git clone https://bitbucket.org/ilps/lerot.git
$ cd lerot
$ python setup.py install
```

This will copy the source of Lerot into a directory called `lerot` in your current working directory and it will be available system wide to import into python.

3.2 Configuration

Lerot can be flexibly configured using yaml files. A full example of a configuration file can be found in Listing 3. For instance, to pick DCM as the user model, `user_model` can be pointing to the `environment.CascadeUserModel` class.

Lerot requires training and test query files in SVMLight format (plain or gzipped) [16]. The framework has been shown to run with the LETOR 3.0 and LETOR 4.0 collections [21], and the Yahoo! Learning to Rank Challenge [2] and MSLR-WEB30k data sets. These data sets all consist of a number of query-documents pairs, each represented by a sparse feature vector and the relevance for each document with respect to the query is judged by professional human annotators. Relevance scales can be binary or graded.

The data set mentioned in Listing 3, e.g., MQ2007 from the LETOR 4.0 collections, can be downloaded and unpacked as follows:

```
$ mkdir data
$ wget http://research.microsoft.com/en-us/um/
    beijing/projects/letor/LETOR4.0/Data/MQ2007.rar
    -O data/MQ2007.rar
$ unrar x data/MQ2007.rar data/
```

[6]Download from http://www.gurobi.com with a free academic trial license.

3.3 Running

After a configuration file is created and the data is prepared, a learning experiment can be run as follows:

```
$ python src/scripts/learning-experiment.py
                              -f config/config.yml
```

With `--help`, we can see all the options it accepts. Settings from the configuration file can be overwritten using the command line. When Lerot is run, a backup copy of the actual configuration it runs is always kept alongside the results it produces.

Running experiments with many repetitions over several data sets and user models is computationally very expensive. With Lerot, it is possible to distribute computation over many machines. Lerot uses Celery to handle the bookkeeping of the distribution. The configuration file has to be extended with some additional information regarding the data sets and user models over which the experiment should be run. An example configuration is included.

```
$ python src/scripts/meta-experiment.py
                            -f config/meta-config.yml
```

Again, we can see all the options it accepts with `--help`. In order to rerun the last experiment, e.g., in case some parts failed, we can specify `--rerun`.

4. CONCLUSION

Online learning to rank is a rapidly evolving area in information retrieval. While several libraries exist for offline learning to rank, Lerot is the first framework for online learning to rank. The framework has been used in many recent publications and reproducing results from those papers only requires a user of the framework to run it with the appropriate configuration file. In sum, Lerot is easy to use and extensible. We have described all functions that need to be implemented in order to do so.

In the context of living labs, Lerot supports two directions of development. First, it allows for experiments with simulated users. The user models it currently implements reflect our current understanding of user behavior; they can easily be extended or replaced by evaluations under different sets of assumptions. Second, Letor provides components that implement complete online learning to rank solutions for use as part of complete living lab evaluation setups.

Acknowledgements. This research was supported by the European Community's Seventh Framework Program (FP7/2007-2013) under grant agreements nr 258191 (PROMISE Network of Excellence) and 288024 (LiMoSINe project), the Netherlands Organisation for Scientific Research (NWO) under project nrs 727.011.005, 612.001.116, HOR-11-10, the Center for Creation, Content and Technology (CCCT), the QuaMerdes project funded by the CLARIN-nl program, the TROVe project funded by the CLARIAH program, the Dutch national program COMMIT, the ESF Research Network Program ELIAS, the Elite Network Shifts project funded by the Royal Dutch Academy of Sciences (KNAW), the Netherlands eScience Center under project number 027.012.105 and the Yahoo! Faculty Research and Engagement Program.

REFERENCES

[1] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML '05*, 2005.

[2] O. Chapelle and Y. Chang. Yahoo! Learning to Rank Challenge Overview. *Journal of Machine Learning Research*, 2011.

[3] D. Chen, W. Chen, and H. Wang. Beyond ten blue links: enabling user click modeling in federated web search. In *WSDM '12*, 2012.

[4] A. Chuklin, A. Schuth, K. Hofmann, P. Serdyukov, and M. de Rijke. Evaluating Aggregated Search Using Interleaving. In *CIKM '13*, 2013.

[5] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *WSDM '08*. ACM Press, 2008.

[6] F. Guo, L. Li, and C. Faloutsos. Tailoring click models to user goals. In *WSCD '09*, 2009.

[7] F. Guo, C. Liu, and Y. M. Wang. Efficient multiple-click models in web search. In *WSDM '09*, 2009.

[8] J. He, C. Zhai, and X. Li. Evaluation of methods for relative comparison of retrieval systems based on clickthroughs. In *CIKM '09*. ACM Press, 2009.

[9] K. Hofmann. *Fast and Reliably Online Learning to Rank for Information Retrieval*. PhD thesis, University of Amsterdam, 2013.

[10] K. Hofmann, S. Whiteson, and M. de Rijke. Contextual Bandits for Information Retrieval. *NIPS '11*, 2011.

[11] K. Hofmann, S. Whiteson, and M. de Rijke. A Probabilistic Method for Inferring Preferences from Clicks. In *CIKM '11*. ACM Press, 2011.

[12] K. Hofmann, S. Whiteson, and M. de Rijke. Estimating Interleaved Comparison Outcomes from Historical Click Data. In *CIKM '12*, 2012.

[13] K. Hofmann, A. Schuth, S. Whiteson, and M. de Rijke. Reusing Historical Interaction Data for Faster Online Learning to Rank for IR. In *WSDM '13*, 2013.

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

[15] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4), 2002.

[16] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1999.

[17] T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02*, 2002.

[18] T. Joachims. Training linear SVMs in linear time. In *KDD '06*. ACM Press, 2006.

[19] T. Joachims, L. Granka, B. Pan, H. Hembrooke, F. Radlinski, and G. Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in Web search. *ACM Transactions on Information Systems*, 25(2), 2007.

[20] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3): 225–331, 2009.

[21] T.-Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. LETOR: Benchmark Dataset for Research on Learning to Rank for Information Retrieval. In *LR4IR '07*, 2007.

[22] F. Radlinski and N. Craswell. Optimized interleaving for online retrieval evaluation. In *WSDM '13*. ACM Press, 2013.

[23] F. Radlinski, M. Kurup, and T. Joachims. How does clickthrough data reflect retrieval quality? In *CIKM '08*, 2008.

[24] D. Sculley. Large scale learning to rank. In *NIPS 2009 Workshop on Advances in Ranking*, 2009.

[25] Y. Yue and T. Joachims. Interactively optimizing information retrieval systems as a dueling bandits problem. In *ICML '09*, 2009.

[26] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *ICML '04*. ACM, 2004.