

INTERACTIVE LEARNING TO RANK AND VISUAL RANK
INTERPRETATION

by

Mateus Malvessi Pereira

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
March 2020

© Copyright by Mateus Malvessi Pereira, 2020

Table of Contents

List of Tables	iv
List of Figures	v
Abstract	viii
List of Abbreviations and Symbols Used	ix
Acknowledgements	xi
Chapter 1 Introduction	1
1.1 Contributions	3
1.2 Organization	3
Chapter 2 Iterative Learning to Rank from Explicit Relevance Feedback	5
2.1 Introduction	5
2.2 Related Work	6
2.3 Proposed Method	9
2.3.1 Information Retrieval	9
2.3.2 Learning to Rank	10
2.3.3 Relevance Feedback	11
2.3.4 Re-Ranking	12
2.4 Experimental Results	12
2.4.1 Simulated Oracle	14
2.4.2 Ablation Study	15
2.4.3 Case Study: Community Question Answering	18
2.4.4 Evaluation	19
2.5 Conclusion	21
Chapter 3 Visualizations for Learning to Rank Interpretation	22
3.1 Introduction	22
3.2 Related Work	23
3.2.1 General Models for Interpretation	24
3.2.2 Ranking Interpretation	24

3.3	Method	25
3.3.1	Ranking Comparison Visualization	27
3.3.2	Cumulative Feature Gain Visualization	28
3.3.3	Cumulative Rule Gain Visualization	31
3.4	Case Study: Ranking Model Evolution	32
3.4.1	Dataset	33
3.4.2	Usage Scenario	33
3.5	Improving the Learning to Rank Method	36
3.5.1	Problem Analysis	36
3.5.2	Solution Evaluation	37
3.6	Case Study: Understanding Ranking Order	38
3.6.1	Dataset	38
3.6.2	Usage Scenario	39
3.7	Discussion and Limitations	42
3.8	Conclusion	43
Chapter 4	Discussion	47
4.1	Learning to Rank Models & Features	47
4.2	Visualizations Design Process	49
4.3	Limitations and Future Work	51
4.4	Conclusion	53
Appendix A	Metrics used in Evaluations	54
A.1	Recall	54
A.2	Mean Average Precision	54
A.3	Normalized Discounted Cumulative Gain	55
Appendix B	ACM Copyright Letter	57
Bibliography	62

List of Tables

2.1	The parameters used for evaluating various features of the proposed system.	12
2.2	The MAP calculated on SemEval 2016 and 2017 datasets compared with results of two baseline models and the top 3 results from the SemEval competition. Our result represents the configuration with least amount of iterations to achieve a higher result.	19
2.3	NDCG for MQ2007/2008 compared with perfect click model from [22]. Our result represents the configuration with least amount of iterations to achieve a higher result.	20
4.1	Size of feature sets and average execution time for one iteration for different feature sets for LambdaMART and Rank SVM, averaged after 7 iterations of feedback.	49

List of Figures

2.1	The effect of variations of different parameters of the system on the quality of the retrieved results in NDCG@10 over 8 iterations. Iteration 0 is the initial set of results, feedback instances start on iteration number 1. Every graph shows two static baselines, the non-pretrained that uses TF-IDF and the pretrained that considers a LambdaMART model trained on the training set. (a) Effect of usage of novel documents. (b) Varying starting point from a non-pretrained model and a pretrained one. (c) Varying type of reinforcement of previous feedback. (d) Effect of feedback provided considering AL instances from model uncertainty and using the top 10 documents.	13
2.2	Effect of Question and Answer feedback on validation set of SemEval 2016. Questions using {0, 1, 4} and Answers using {0, 1, 2, 4} feedback per iteration. Question feedback improve results faster than feedback only on Answers.	15
2.3	Results on SemEval 2016 and 2017, MQ2007 and MQ2008 considering 2 feedbacks per iteration on MQ2007 and MQ2008 and 1 question feedback and 2 answers on SemEval.	16
3.1	Overview of interface and proposed visualizations and navigation flow. a) Search bar. b) Results from search. c) Cumulative Feature Gain. d) Cumulative Rule Gain. e) RankDiff visualization.	26
3.2	Example feature cell utilized in the Cumulative Feature Gain visualization. a) Elements marked as relevant. b,d) Arrows represent score influence of this feature over the instance. c) Division line, on the left are the top 20 relevant items, and on the right the bottom 10. e) An element highlighted by selection. f) The right most division line is used to separate instances of interest that did not appear in the previous lists. g) Elements that were marked as irrelevant. Elements are positioned sequentially based on their ranking order in the X-Axis. The position on the Y-Axis encodes the feature value for each element.	29
3.3	Cumulative Rule Gain, looking only at 3 ranked elements (E1, E2, E3) and 6 ranking rules (R1 to R6). Color red and green of the rules indicate a negative or positive contribution, respectively.	32

3.4	First iteration result on the RankDiff case study. The squares represent items sorted by ranking score, from left to right in descending order.	32
3.5	Behavior of the Learning to Rank algorithm over 4 iterations. a) the behavior of the original algorithm. b) the behavior after changes to the algorithm from the intuition taken from the previous result. c,d) Feedback that did not change position.	33
3.6	Evaluation on MQ2007 average for all folds on test set. <i>Before</i> represents the results before the changes to the method, and <i>After</i> changes to the method, developed after insights taken from the RankDiff visualization.	36
3.7	Evaluation of different combinations of parameters j and l in the proposed solution to avoid the stagnation problem.	38
3.8	Evaluation on the SemEval 2016 and 2017 and on MQ2008 averaged for all folds on test set, average of 30 executions. <i>Before</i> represents the results before the changes to the method, and <i>After</i> changes to the method, developed after insights taken from the RankDiff visualization.	39
3.9	Search results for “pirates” keyword. The left is a list of entries matching the query term, and on the right the system recommendations based on the left list.	41
3.10	Cumulative Feature Gain view for the recommendations after the initial search. The rules on the top were shown after hovering the first element in cell #1.	42
3.11	Cumulative Feature Gain view for the recommendations in the second iteration. The rule on the top relate to the first element in cell #5.	43
3.12	Cumulative Rule Gain view for the recommendations, displaying all rules that are applied to score the documents on the first iteration. On the top, the selected rules ($R1$, $R2$, $R3$ and $R4$) are displayed in detail.	44
3.13	Cumulative Rule Gain view for the recommendations, displaying the second iteration. The rules related to the current iteration have a black indication on the left side. Rules with black borders are selected and shown in detail on the top.	45
3.14	RankDiff in the middle of the transition to results of the second iteration of the ”Understanding Ranking Order” Case Study, showing how new documents enter the view.	46

4.1	Comparison of LambdaMART and Rank SVM performance on NDCG@10 for different feature sets.	48
4.2	Initial visualizations developed for the dashboard with RankDiff as the central visualization, along with other surrounding visualizations. a) Correlation Matrix displaying pairwise cosine similarity between elements. b) Feature Heat-Map displaying element features. c) Leaf Scores visualization, displaying active leaf scores for every element.	50
4.3	Hierarchical Forest visualization, which was initially considered for the dashboard, here showing five levels of the regression forest. Each black rectangle represents a feature. Topmost connections represent a tree, connecting to its first feature for decision. Connections to the feature nodes are positioned relative to their decision threshold. Each line represents a branch of a tree. The red and blue rectangles represent the leaves, which are a shade from red to blue representing a negative to positive contribution.	52

Abstract

Many algorithms in the Information Retrieval domain have been developed considering training models using vast amounts of data. The acquisition of this data, however, is time-consuming and requires lots of human effort. Active Learning techniques try to solve this problem by reducing the number of instances needed in the training phase by selecting relevant instances to be labelled. Although such an approach has been proved to be effective, it is still hard to understand how the model is changing after every relevance feedback. As a potential solution, the use of visualizations to help users to understand models is becoming a widespread approach both to understand the overall behaviour of a model and to analyze individual data instances. In this thesis, I explore the utilization of a Learning to Rank algorithm in a relevance feedback scenario and the use of visualizations to understand the reasoning behind the model's ranking decisions.

List of Abbreviations and Symbols Used

B_{fe}	Rules related to feature f and element e .
D_{fe}	Contribution of feature f for element e .
F_e	Array of feature values for element e .
F_{max}	Max value for a feature.
F_{min}	Min value for a feature.
I_f	Importance of feature f .
P_r	Array of element positions in a given ranker r .
S_{acc}	Accumulated score used in Cumulative Rule Gain visualization.
U	Visible set of documents in the Cumulative Rule Gain visualization.
V_{max}	Max value for the accumulated score of the documents in Cumulative Rule Gain visualization.
V_{min}	Min value for the accumulated score of the documents in Cumulative Rule Gain visualization.
\Re	Real numbers.
λ	Size of the gap used in the cells of the Cumulative Feature Gain visualization.
μ	Size of elements in the visualizations.
θ	Proportion of the vertical space used by the arrows in the Cumulative Feature Gain visualization.
e	An element.
e_x	Position of element in the X-axis.
e_y	Position of element in the Y-axis.
h	Height size available for visualization.
l	Length of arrows in Cumulative Feature Gain.
w	Width size available for visualization.
AL	Active Learning.

BOW Bag of Words.

CFG Cumulative Feature Gain.

CQA Community Question Answering.

CRG Cumulative Rule Gain.

IIR Interactive Information Retrieval.

IR Information Retrieval.

LtR Learning to Rank.

MAP Mean Average Precision.

NDCG Normalized Discounted Cumulative Gain.

SVM Support Vector Machine.

TFIDF Term Frequency-Inverse Document Frequency.

Acknowledgements

I would like to show my gratitude to my supervisor, Dr. Fernando Paulovich, for encouraging me to seek a graduate degree. For his his support and guidance provided throughout this project. Thank you for this opportunity and all the experiences that came from it. I also want to acknowledge Dr. Elham Etemad, that was my co-supervisor during this project, with whom I had many interesting discussions about research, our cultures and many other things.

I want to thank all the people from Waterford Energy Services Inc., partner organization responsible for part of the motivation of this project. For the friends that I made and the learning opportunities. This work was supported by Mitacs through the Mitacs Accelerate program.

Special thanks to the many friends that were around the lab during these two years. For the wings nights, for the movies, for other events and all stuff that we did together. Specially the wings nights. Thanks to my family and friends from home, that were far away from be, but still kept in touch.

Chapter 1

Introduction

Since the boom of the internet, unprecedented amounts of data have been created and stored in such large quantities that it is now impossible for a standard user to extract any information from it quickly. This created the need to have machines to do part of the work for us, leading to the development of Information Retrieval (IR) techniques, which are utilized to retrieve relevant documents in a sea of information.

IR became a massive field of study, being originally aimed to find relevant information in small collections of data in text domains, which quickly started to grow for larger collections and other media with the popularization of the internet and search engines. Like any other area of research, IR began with simple models, initially based on keyword matching, then evolved to more complex mathematical models that considered the overall term frequencies of the entire document and collection, such as TFIDF [56, 39] and BM25 [53]. Machine Learning (ML) is another big area of research that is applied to all kinds of tasks, including IR. The use of ML in IR leads to the development of more complex models and document representations, such as Bag of Words (BoW) and Word Embeddings [40, 9].

Although Information Retrieval solves many problems when trying to find relevant documents, there are still difficulties in properly ranking the results in an optimal way. In this context, the research field of Learning to Rank (LtR) emerges in an attempt to solve the need to create better rankings [70]. The intention of LtR algorithms is not to classify or predict values but to find optimal ordering for a set of items. These algorithms, however, also require some labeled samples to learn patterns in the data and be able to generate meaningful rankings.

Learning to Rank algorithms are usually trained considering labeled query-document pairs, which indicate the importance of the document to a given query. The acquisition of training data is always an expensive and time-consuming process in most domains, requiring lots of manual effort to label instances. In learning to rank tasks, the labeling process requires the annotator to label the importance of a group of instances in relation to a query,

which should have at least two relevance levels, including relevant and irrelevant, but more levels can also be used.

A solution that reduces the need for loads of training data is the so-called Active Learning, where the system returns to the user instances that, if labeled, should help in improving the prediction behavior using far fewer training instances. Active Learning in LtR is not new and has already been applied before [58, 10].

Interactive Learning to Rank is a field of research that joins Machine Learning, Human-Computer Interaction, and other information retrieval related areas, to improve ranking functions [7]. Usage of user interaction in the form of feedback is a common approach of including the user in the information retrieval process. Two types of user feedback are possible: implicit and explicit. Explicit feedback comes in the form of a conscious input given by a user, such as a review or a grade score. The implicit feedback is extracted from other indirect user interactions, such as clicks and mouse movements. However, implicit feedback is often more noisy and biased, usually requiring many users to generate enough data. Because this work assumes a system with a small number of users and no initial training data, the explicit feedback approach is used.

The necessity of having a ranking on a new system with a small number of users motivates the use of Interactive Learning to Rank in this work. Such need is from a partner organization that supported this research, in which the data lacks the previous existence of query/document relations and only contains documents. I experimented with a solution to interactively learn to rank on a query on the fly, assuming no other existing query/document groups are available. It is worth mentioning that, over time, the system would accumulate enough data to be used as training, but this is out of the scope of this work.

For every ML task, including LtR, there are numerous ways of solving the same problem, and the usage of such models in a production environment is taken with care since often, it is hard to know what is happening inside the model. This motivated many studies in trying to explain and find the reasoning behind machine learning models and is a promising research direction [67, 20, 28]. Machine Learning interpretation can be considered, overall, in two levels: transparency of the model and post-hoc interpretability. The former considers a global picture and overall view of how the model sees the data, and the latter assumes a local interpretation where it is possible to understand a single or compare a few instances [13, 35, 14].

In LtR, as in other tasks, the understanding of what information the system is using to find relevant documents is crucial to gain user trust. Besides that, as other studies have already shown [3, 1], understanding how the ML model is behaving can reveal weaknesses and flaws that can then be fixed. This also motivates this work, in which I propose visualizations to assist in understanding the behaviour of the LtR model.

1.1 Contributions

This work has two main contributions, the first related to a iterative Learning to Rank method, and the second in the interpretability of LtR models. The first contribution is a novel iterative Learning to Rank method. This method is utilized in an interactive tool, that improves the search results through the user’s relevance feedback provided at each iteration. This method is deeply evaluated through several ablation studies on standard LtR datasets.

The second contribution is the development of three new visualization techniques to enhance learning to rank model understanding and explanation. The first visualization technique, RankDiff, allows the comparison between different rankings results, independently of their underlying ranking models. The second, Cumulative Feature Gain (CFG), allows an overall understanding of which features are important to the ranking model and how they affect the ranking of the top instances. The third, Cumulative Rule Gain (CRG), offers a way to compare the rules that cause difference in ranking of a selection of documents. Besides these visualizations, I also proposed a new metric to evaluate the Iterative Learning to Rank models, based on the insights acquired from analyzing them. Finally, a Case Study is performed to show the value of the proposed visualizations and how to interpret their results.

1.2 Organization

This work is organized in three chapters, two of which are based on papers developed during the Master’s program, and the last which finalizes this work with some discussions. Chapter 2 is a reproduction of the paper “*Iterative Learning to Rank from Explicit Relevance Feedback*” accepted in the ACM/SIGAPP Symposium on Applied Computing 2020¹

¹<https://www.sigapp.org/sac/sac2020>

conference, in the Information Access and Retrieval Track. This chapter develops the Iterative Learning to Rank with user feedback, starting with a literature review in LtR, followed by the description of the proposed method, its evaluations and results.

Chapter 3 is an extended version of the research paper “*Visualizations for Learning to Rank Interpretation*”, submitted to the Computer Graphics International 2020², which discusses the development of the visualizations techniques for interpretation of the ML model applied in LtR. This chapter starts with a discussion of the related works, followed by elaboration of the proposed visualizations and their evaluations.

Lastly, Chapter 4 discusses early experiments conducted that complement the previous chapters. It also discusses some future work, limitations and final conclusions.

²<http://www.cgs-network.org/cgi20>

Chapter 2

Iterative Learning to Rank from Explicit Relevance Feedback

This chapter is a reproduction of the paper [47] submitted and accepted by the ACM Symposium on Applied Computing 2020. The thesis writer is the main author and main contributor to the respective conference paper, being responsible for implementation, evaluations and most of the analysis and writing of the paper. A copy of the ACM Copyright agreement is attached in Appendix B.

2.1 Introduction

Given the increasing amount of information accessible through the internet, many machine learning approaches have aimed to improve the quality of search engines and Information Retrieval (IR) techniques. Besides, personalizing the search results has a significant impact on users' satisfaction [71]. Interactive Information Retrieval (IIR) systems try to address both demands by capturing users' feedback to improve the order of the retrieved results based on personal preferences and intents when they are presented.

Various IIR approaches have been developed for improving the ranking order of the retrieved information using Learning to Rank (LtR) techniques, and for capturing the relevance feedback from users [7]. LtR techniques are supervised learning approaches that train a model using pairs of queries and documents with their relevance scores as labels. This relevance can be explicitly expressed by or implicitly perceived from user's interactions. The trained model is then used to rank documents given a new query [8]. The performance of LtR techniques depends on the amount of available training data, which is costly to obtain, giving rise to active learning techniques for LtR [10].

Other techniques have been developed for collecting relevance feedback to improve users' experience while working with IIR systems. Researchers have studied implicit and explicit feedback from users and considered the effects of positive and negative feedback separately and in combination [7]. Implicit feedback is obtained by capturing general user interactions like user clicks, while explicit feedback happens when the user provides direct

relevance feedback. These techniques have adopted the user feedback for exploration in the search space [46], finding users' intent [54], and re-ranking the retrieved documents [31].

Most LtR and relevance feedback techniques have been applied in IIR systems independently. Their combination is mostly considered in cases of implicit feedback on search engines [50, 51]. Methods based on implicit feedback often assume the existence of many users whose interactions with the system are required for training the model. This is infeasible for enterprise-specific systems with less users looking for specific information.

Explicitly capturing user's feedback is only studied for finding the relevant documents from PubMed [72] which considers multi-level relevance feedback. Although their technique is similar to the proposed method, their performance is much lower in the benchmark dataset considering our base LtR model.

This work describes an online learning to rank approach to personalize and improve the quality of search results through the user's explicit feedback. We propose and evaluate novel approaches on how to use each user's feedback to improve the ranking of the results. We have utilized the LambdaMART algorithm [69] for ranking the retrieved documents and fine-tuning the model based on the accumulated feedback from the user. The proposed method is evaluated in the scenarios with or without available initial training data. Besides evaluating our performance on general learning to rank scenarios using LETOR 4.0 [48] datasets, we have also applied our proposed method in the specific test case of Community Question Answering forums and evaluated its ranking performance, using the SemEval 2016 and SemEval 2017 datasets [44, 43].

We have provided a brief review of the related works on Learning to Rank and Relevance Feedback in Section 2.2. The details of the proposed method and Learning to Rank module are elaborated in Section 2.3. The proposed system is evaluated in Section 2.4, and we conclude this chapter in Section 2.5.

2.2 Related Work

In this section, we present a brief review of the existing Learning to Rank algorithms and active learning heuristics applied in this domain. We show some of the existing systems for capturing relevance feedback in general and for improving LtR algorithms. Also, we discuss techniques used in community question answering forums for finding relevant questions and answers.

Learning to Rank: In general, LtR methods train a model on labeled pairs of query and document and use it to provide a better ranking for the retrieved documents for a query, usually measured by Normalized Discounted Cumulative Gain (NDCG) metric. There are three main categories of LtR methods: pointwise [38], pairwise [69], and listwise [68]. They differ in the number of training instances used for updating their loss functions varying from a single training instance (pointwise), a pair of training instances (pairwise), and all pairs associated (listwise) [25]. All these techniques require training data in order to learn the relevance between query and documents, but there are many cases where none or a small training data is available. This has created a new field of research for active learning techniques for LtR.

Active Learning (AL) techniques are used to find the most informative documents for the LtR so they can be labeled by the user. Various heuristics such as *Query by Committee* [10], *Minimizing Hinge Rank Loss* [12] and *Expected Loss Optimization* [37] have been applied in the literature for finding these informative training instances. In this work we evaluated two ways of selecting instances to be labeled, selecting the most uncertain instances for the model and selecting from the actual top- n ranked documents.

Relevance Feedback: The relevance feedback has been used more often in the context of exploratory searches. Contrary to the lookup search (focus of this work) in which the user precisely formulates the query, in the exploratory search, the user may not know his intent or its exact wording [29]. Various tools have considered the exploratory search scenario to model the concept drift [29], user intent [54], and to evaluate the effect of negative feedback on the search result [46]. Some other methods have utilized the user's feedback to automatically decide whether the user is performing lookup or exploratory search [2]. Some of these methods rely on the term-based relevance feedback for ranking the results to show the most relevant documents to the users [46]. The main issue with these techniques is the fact that asking for term-based feedback from the user has a high cognitive load comparing to typing new queries and results in context trap for the user [2]. To avoid this context trap, we are capturing document-level feedback where the user (simulated) provides positive or negative feedback on the retrieved documents.

None of these methods has utilized the user's feedback to improve their LtR algorithms. However; there are techniques which have used the implicit feedback that is captured in the system's logs. This implicit feedback is utilized to recognize the user's behavior and

personalize the list of retrieved documents [51], or to predict the next query of the user based on the previous chain of user's queries [50]. The explicit feedback which affects the LtR algorithm was studied in searching for scientific papers where the user can select the relevance scores of retrieved papers [72].

Some research have been done on Online LtR, where the system uses implicit feedback, user clicks on the retrieved results, to improve their LtR module [22, 76]. These techniques usually balance the returned results by having a mix of documents obtained by an exploitative and a more relaxed model. The exploitative model retrieves the most relevant documents while the relaxed model returns documents with the intent of better exploration of the document space. While these approaches consider the use of implicit feedback and a considerable amount of iterations with different user queries, our proposed method utilizes explicit feedback on retrieved documents for a single user query over a handful of iterations.

Community Question Answering: The Community Question Answering (CQA) forums are playing an essential role in collaborative learning and knowledge sharing on the web [59]. Websites such as Stack Overflow are tools for internet users seeking answers to their questions. These forums constantly capture explicit relevance feedback from user's votes and receive new questions and answers.

The ability to accurately retrieve the relevant questions and answers to the user's query in a Community Question Answering forum reduces the redundancy of capturing answers and low-quality questions in the forum [59]. This unwanted redundancy decreases the sustainability of the forum and its effectiveness in responding to the users' queries [60].

In the last decade, much research has considered solving the underlying challenges of community question answering systems. The Natural Language Processing (NLP) approaches have their focus on retrieving the most similar questions to the input query [75, 49], and the best-provided answer in the existing threads [4, 64]. These techniques mostly focus on bridging the semantic and lexical gap between the new query and the existing questions and answers in the database. While they are beneficial in enhancing the IR performance, they do not consider usage of relevance feedback to improve the LtR model.

Algorithm 1 Interactive Learning to Rank

Variables:

D, Document Collection; *Q*, Current Query;
RD, Retrieved Documents; *RL*, Ranked List;
F, Relevance Feedback; *PF*, Previous Feedback;
Model, LambdaMART Model;

end Variables:**function** ILTR(*D*, *Q*)

RD = **RETRIEVE**(*Q*, *D*) ▷ 2.3.1

RL = **RANK**(*Model*, *RD*, *Q*) ▷ 2.3.2

while **FINISHED**(*D*, *Q*) == *False* **do**

F = **FEEDBACK**(*RL*) ▷ 2.3.3

RL = **RE-RANK**(*Model*, *Q*, *RD*, *F*, *PF*) ▷ 2.3.4

PF = [*PF*; *F*]

end while

end function

2.3 Proposed Method

Given an input query, our system employs an active learning to rank approach to rank documents based on the user's feedback. Starting with a list of documents retrieved and initially ranked, the model learns from the user through an iterative feedback process to refine the ranking of the results until the user is satisfied. The overall system consists of five main modules: information retrieval, Learning to Rank, relevance feedback, data preparation, and fine-tuning the ranking model whose calling procedure is illustrated in Algorithm 1.

2.3.1 Information Retrieval

The main goal of the IR module is fetching all the relevant documents to a given query, attaining a high recall rate. In this research, a base retrieval system retrieves the candidate relevant documents to be ranked. This retrieval module is responsible to provide documents associated with the user's query to the ranking module. The retrieval module generates an initial ranking of the results using a base ranker trained on the available data (pretrained model). When there is no training data available, this retrieval module is considered as

a simple ranking method using the similarity score between the query and the documents (non-pretrained model).

When training data is available, the ranking for initial results is obtained from the pre-trained LambdaMART model. The Term Frequency-Inverse Document Frequency (TF-IDF)[56] similarity score is utilized as the base ranking of the retrieved documents for the non-pretrained model.

2.3.2 Learning to Rank

The initial ranking results from the IR module do not capture the relevance between query and documents, and the user intent. To improve such ranking, we have utilized our adaptation of the LambdaMART learning-to-rank model [69] which learns through feedback iterations. LambdaMART is a supervised LtR model trained on existing pairs of $(query, document)$ labelled by their degree of relevancy.

LambdaMART [69] combines the LambdaRank [6] and McRank [34] algorithms to achieve a better performance/speed trade-off. This model optimizes the ranking of documents by training a forest of boosted regression trees on the gradients of the loss function. This algorithm optimizes the ranking of results measured by the NDCG, a discrete-valued and non derivable function, by choosing the lambda values that reduce the pairwise errors. These lambdas are found by swapping the document pairs in the ranked list and monitoring the gain imposed on the NDCG metric.

To calculate NDCG gain, two weak learners of R and \hat{R} are used for sorting the training data. Each weak learner is a regression tree, that contributes to the final score of a document. The model looks for $\alpha \in [0,1]$ which maximizes the score defined by $s_j = (1 - \alpha)s_j^R + \alpha s_j^{\hat{R}}$. In this equation, s_j is the score for the document D_j in the list, and the s_j^R and $s_j^{\hat{R}}$ are the scores of the weak learners R and \hat{R} for this document. This allows an optimal combination of the weak learns to build the forest of regression trees.

To use the LtR model, not only we have to extract features from the query and the document independently, but we also have to extract features depending on both the query and the document. These dependent features are essential to encode the relevance between the query and document pairs.

In the first iteration, the model is initialized by training on the top and bottom n results returned by the retrieval module. This initial training approximates the retrieval module's

ranking with a smaller model that is fine-tuned in the following iterations with the feedback instances.

2.3.3 Relevance Feedback

The user is able to provide positive or negative feedback f on the retrieved documents. Since this is a binary operation, we propose to interpolate the current document score with the relevance feedback value. We consider the value of relevance score $f_n = \min_{l \in L} l$ and $f_p = \max_{l \in L} l$, where L is the range of relevance scores in the dataset, for the negative or positive feedback respectively. The document’s relevance score in iteration i , $Y_{d(i)}$ is calculated according to equation 2.1 in which α is a parameter of the model, and $Y_{d(0)} = 0$.

$$Y_{d(i)} = Y_{d(i-1)} + \alpha(f - Y_{d(i-1)}), \quad f \in \{f_n, f_p\} \quad (2.1)$$

The main purpose of this interpolation is to diversify the relevance scores of the instances receiving feedback and still keep some weight of the community scoring when available. This diversification also addresses the LambdaMART’s requirement of having various relevance scores for training. As discussed in Section 2.3.2, LambdaMART [69] is only able to learn the ranking of documents when its training data has instances with different relevance scores.

The feedback instances from previous iterations are considered as training instances in the following iterations to reinforce their relevance scores. In case different feedback values are provided on the same item over the iterations, the latest feedback overrides the value received on previous iteration.

We evaluated the feedback from the user considering two options: the current top- n results or the instances selected by active learning. In the case of active learning, a group of documents are selected by applying the Query by Committee [10] technique to the weak rankers in the LambdaMART model, as the committee. The standard deviation over the scores of individual rankers for each document is calculated and the documents with higher values are considered as uncertain instances for LambdaMART. Higher value of standard deviation shows higher disagreement between the LambdaMART rankers. These selected documents are added to the list of candidates for active learning if they have not already received feedback.

After each iteration of feedback, the LambdaMART model is fine-tuned, considering

Table 2.1: The parameters used for evaluating various features of the proposed system.

Parameters	Fixed Setting	Variable Setting
Include novelty	True	True, False
Reinforcement	Both	Negative only, Positive only, Both, None
Pre-trained model	False	True, False
Usage of AL	False	True or False

the training data obtained from the relevance feedback to improve the model’s performance. In each iteration, several new regression trees are added as weak learners for LambdaMART. This helps the model to keep updating its weights to capture the relevance score of the new training data. The process of fine-tuning the model using relevance feedback continues until the maximum iteration number is reached.

2.3.4 Re-Ranking

All documents that have received feedback in iteration i are considered as new training data to fine-tune the LambdaMART model. For each instance of feedback, we have a training sample consisting of the current query, the document with feedback, and the value of feedback $Y_{d(i)}$. In case there is variety on the values of $Y_{d(i)}$ for the training instances, this list is utilized to fine-tune the model.

If all $Y_{d(i)}$ s obtained from the relevance feedback module are equal, LambdaMART is not able to learn from them. To tackle this limitation, we include one new training sample, called **novelty** document. This sample is obtained by looking for the most dissimilar document to the average feature vector obtained from the relevance feedback instances. Only documents without feedback history are candidates to be selected as the opposite sample.

The novel document is attributed an opposite relevance score of the documents that came from the relevance feedback module $Y_{d(i)} = \max(L) - Y_{d(i)}$, in which $\max(L)$ is the maximum relevance score considered in the dataset.

2.4 Experimental Results

In this section, we cover the implementation details of the system and the structure of the datasets used for evaluation. We have used a simulated Oracle to evaluate the performance

and effectiveness of the system with various parameter settings. We have utilized pyltr¹ as the LambdaMART implementation.

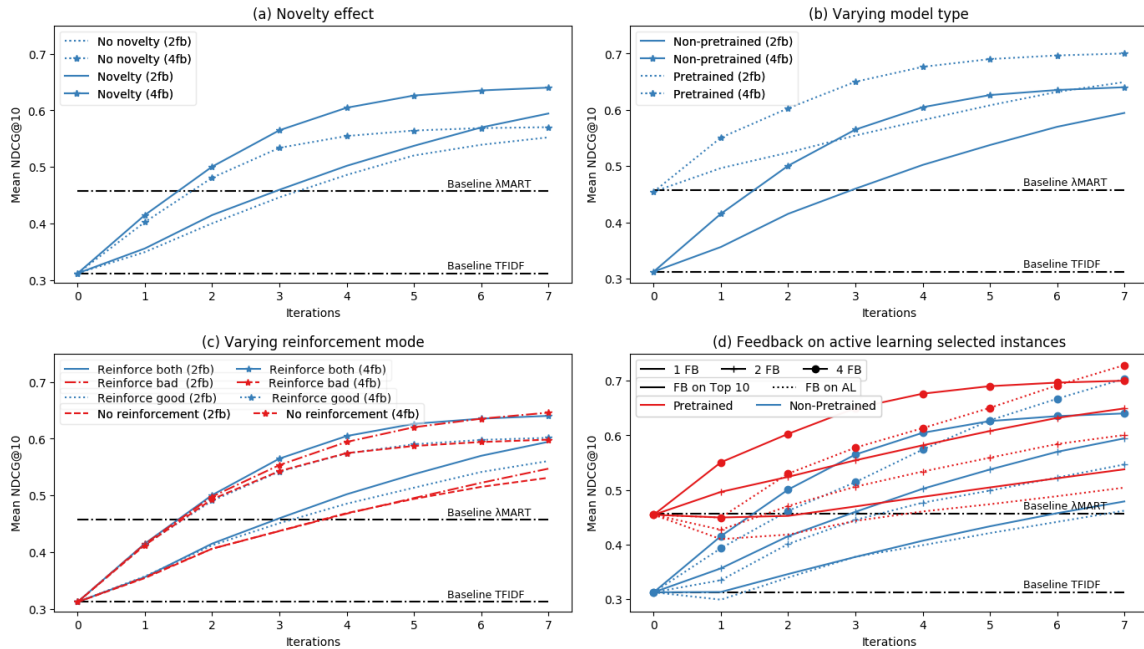


Figure 2.1: The effect of variations of different parameters of the system on the quality of the retrieved results in NDCG@10 over 8 iterations. Iteration 0 is the initial set of results, feedback instances start on iteration number 1. Every graph shows two static baselines, the non-pretrained that uses TF-IDF and the pretrained that considers a LambdaMART model trained on the training set. (a) Effect of usage of novel documents. (b) Varying starting point from a non-pretrained model and a pretrained one. (c) Varying type of reinforcement of previous feedback. (d) Effect of feedback provided considering AL instances from model uncertainty and using the top 10 documents.

We have evaluated our system on four datasets, two standard datasets used for evaluating LtR algorithms, and two in the context of Community Question Answering (CQA) systems. The first two are MQ2007 and MQ2008 from the LETOR 4 collection [48]. LETOR datasets are composed of features for query-document pairs, where each query is related to about 40 documents. There are 5 standard folds, containing training, validation, and test set for each. We utilized the training and validation set to find the best parameters on Fold 1 from MQ2007 and tested on all five folds using the test set. We utilized LETOR standard features and performed the ranking over the instances that are related to each test query. The last two are datasets in the context of CQA. SemEval 2016 [44] contains about 400 questions (users’ queries) along with 7K relevant questions and 57K

¹<https://github.com/jma127/pyltr>

relevant comments in total. SemEval 2017 [43] has 500 questions, 1K relevant questions, and 9K relevant comments. Every test question has 10 related questions, each of which has 10 related comments. We have evaluated our system considering Task 3, subtask C of the SemEval (2016/2017) challenges. The objective of this task is to rank the best answers to a new question. The task assumes that all of these related questions and answers are retrieved and must be ranked, so we only include these documents in the retrieval for evaluation and ranking.

All these datasets are labeled in three levels of relevance from 0 (Not relevant) to 2 (Very relevant). The simulated Oracle utilizes these relevance labels from the dataset to provide feedback (see Section 2.4.1). Since the Oracle uses a random number generator to provide feedback to documents, all graphs were generated using the average of 30 runs.

2.4.1 Simulated Oracle

The user’s behavior is simulated with an Oracle to evaluate the system. This Oracle randomly selects documents to give feedback with descending probability given by the current documents’ ranking. This follows the idea that most users check the top results first, so feedback on those is more likely to happen [18, 19]. Since no click data was available, we defined the probabilities based on the results presented in [27]. The Oracle is constrained to provide feedback only to the top 10 documents (and their respective questions on Community Question Answering test case), or to the 4 instances selected by AL.

Since the Oracle is simulating the user’s behavior, we have considered the following two configurations: Perfect feedback and noisy feedback. In the perfect configuration, the feedback is always correct, while in the noisy configuration, some wrong feedback may be provided by the Oracle.

In the perfect configuration, relevant documents will receive positive feedback ($p(F = \textit{positive} | \textit{related}) = 1$), and non-relevant documents will receive negative feedback ($p(F = \textit{negative} | \textit{non - related}) = 1$) if they are selected. In the case of Community Question Answering, the routine mentioned above applies to questions when they receive feedback. For our experiments, we assumed binary feedback, so documents with a 0 relevancy score are considered as negative, and those with relevancy scores of 1 and 2 are considered as positive. The Oracle does not provide duplicated feedback (two feedback on a document)

in each iteration, which results in having a unique feedback set. However, on the subsequent iterations, an item that already has received feedback may be selected again, which is subjected to feedback reinforcement.

To simulate the noisy feedback configuration, we utilized the informational probability of the user clicking on a relevant document [22] as our probability distribution of providing feedback. The noisy feedback model provides correct feedback with 90% chance ($P(\text{correct} - \text{feedback}) = 0.9$). We consider that a feedback is never repeated for the perfect feedback configuration, and for the noisy configuration, there is a 10% chance of repeating positive feedback and a 90% chance of repeating a negative feedback in case the selected item has already received feedback.

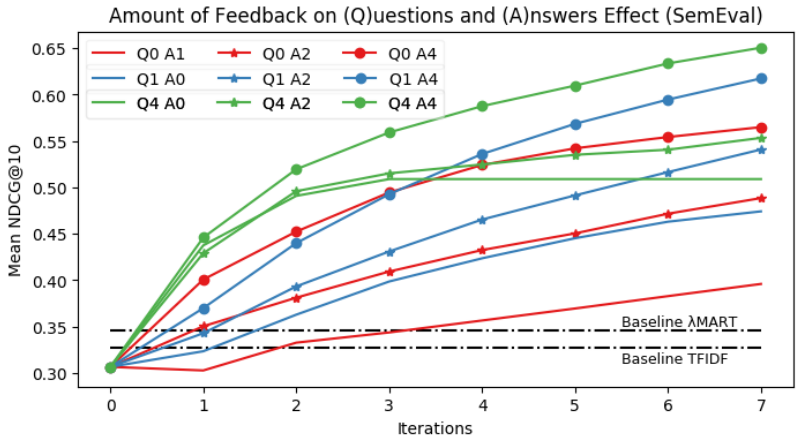


Figure 2.2: Effect of Question and Answer feedback on validation set of SemEval 2016. Questions using $\{0, 1, 4\}$ and Answers using $\{0, 1, 2, 4\}$ feedback per iteration. Question feedback improve results faster than feedback only on Answers.

2.4.2 Ablation Study

There are various parameters in the proposed method whose values may affect the performance. Besides, different settings for those parameters might be found useful for various cases. In this section, we evaluate these parameters on one fold of a LETOR dataset to study their importance on the performance of the model.

To find the best parameters for LambdaMART, we performed a grid search on different combinations of parameters using validation data for MQ2007 Fold1. We consider only a non-pretrained model, learning from scratch for every test question in the validation set. The investigated parameters for the LambdaMART algorithm are the maximum depth of

the trees in the forest, the number of trees added to the forest in the fine-tuning step, and the learning rate. We have tested with $MaxDepth \in [1, 5, 10]$, $TreeGrowth \in [1, 3, 5]$ and $LearningRate \in [0.1, 0.01, 0.001]$. Based on our experimental results, we have chosen the maximum depth of 10, the number of added trees of 5, and the learning rate of 0.01.

Using the simulated Oracle, we have investigated the effect of some of the proposed features of our system regarding the overall quality of the produced results. We have selected the values in Table 2.3.4 for our experiments in this section.

The effect of the amount of feedback per iteration can be seen in all graphs in Figure 2.1. We have compared the improvement on NDCG@10 when the user provides 2 or 4 instances of feedback per iteration. The value of NDCG@10 increases proportionally to the amount of feedback, reaching higher levels faster when more feedback instances per iteration. For instance, considering the corresponding plots for 2 and 4 instances of feedback per iteration in Figure 2.1(a), the user could achieve a score of 0.47 in two iterations with 4 instances of feedback per iteration, while the same value could be achieved with 2 instances of feedback in three iterations.

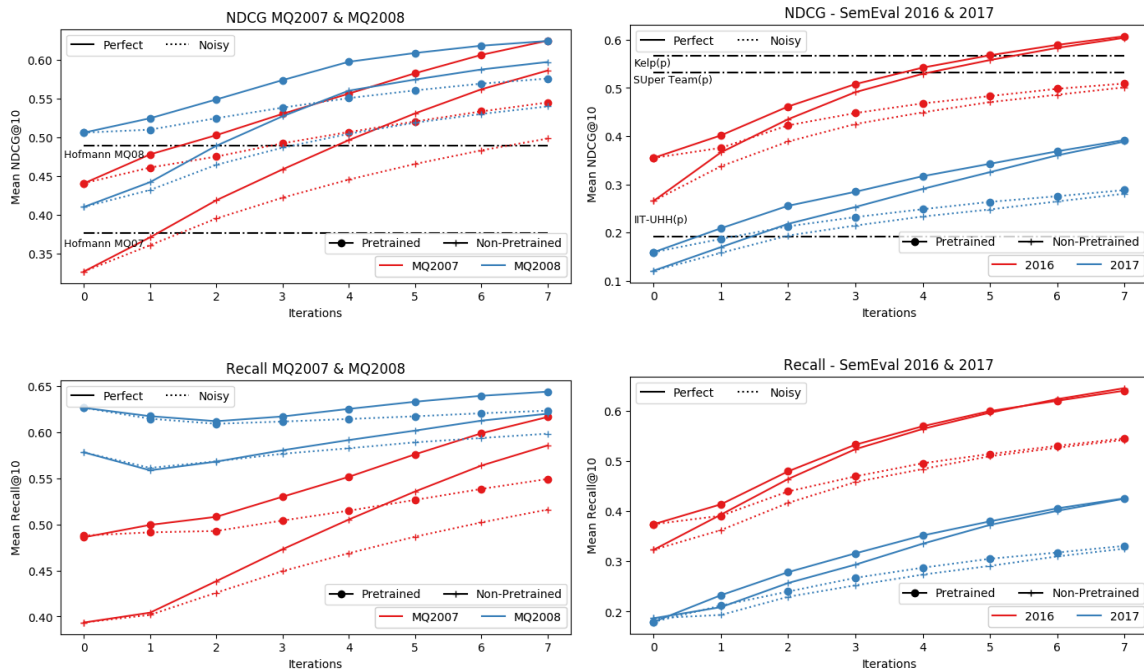


Figure 2.3: Results on SemEval 2016 and 2017, MQ2007 and MQ2008 considering 2 feedbacks per iteration on MQ2007 and MQ2008 and 1 question feedback and 2 answers on SemEval.

The first investigated factor is the effect of adding novelty documents while fine-tuning the model. This technique is similar to the exploratory concept utilized in [22] with the purpose of adding some variability to the training set. However, differently, we do not utilize a separate model for this; instead, we use the most dissimilar documents from the feedback. The effect of using novelty documents is presented in Figure 2.1(a). For many feedbacks per iteration, the improvement on NDCG@10 without adding novelty documents is slower than the improvement when they are included. This is true for different amounts of feedback per iteration.

The diagram in Figure 2.1(b) compares the performance of the model considering the scenario with existing training data (pretrained) and non-existing training data. In our tests, we train the model on iteration 0 using the top 10 and bottom 10 results to reflect the initial rank provided by the retrieval module. As shown in the graphs, there is almost no loss in performance in this approximation. It is clear that the pretrained model is better in delivering the first set of results to the user and can achieve higher NDCG much earlier. However, the non-pretrained model also learns well and can achieve comparable results only with a few more iterations. This shows that a non-pretrained model is a valuable option when no training data is available.

We have tested four settings for reinforcing previous feedback. The reinforcement works by reusing the instances of feedback as part of the training data while fine-tuning the model. We have investigated the following four scenarios: reinforcing only negative, only positive, all, and none of the previous feedback instances. The results in Figure 2.1(c) suggest that reinforcement does not have a significant effect in general. However, reinforcing positive and negative feedback instances achieves the best score with 2 feedback per iteration and is the second-best with 4 feedback per iteration. It is important to reinforce both feedback types to avoid the model from overriding previous knowledge.

The techniques for selecting the documents for feedback are compared in Figure 2.1(d). The active learning technique provides some gain when the total user's feedback instances increases; however, selecting from the top 10 documents shows steady improvement over iterations. Therefore we consider the top 10 documents for the evaluation on test data.

2.4.3 Case Study: Community Question Answering

In the question-answering environment, the question context is not repeated in its associated answers. To address this limitation, the text of the existing question in the database (not the user query) is concatenated to all of its related answers and is considered as a document. This document, along with the user’s query, is one pair that is used for generating features.

We have utilized SemEval datasets as a case study in Community Question Answering. Each training instance in these datasets is represented by 36 features similar to the features in LETOR datasets. Since SemEval datasets provide answers and their related questions Rq , the feature vector for each pair is built considering this relationship. This representation contains the following combinations of the answer’s content: $A + Rq_t$, $A + Rq_b$, $A + Rq_t + Rq_b$, where A represents the answer body and Rq_t and Rq_b the related question title and body, respectively. The following features concerning the new question are calculated for each of these combinations: TF-IDF similarity, BM25 similarity, count of query terms, count of query terms ratio, binary occurrence of query terms, binary occurrence ratio of query terms. There are some features extracted only from the document, which are the Inverse Document Frequency and the Term Frequency.

In the context of information retrieval from question-answering datasets, we have two levels of abstraction. Answers have a detailed description of the solution, while questions produce a higher level of abstraction that contains the problem and various related solutions. To consider this hierarchical structure, we have devised two different mechanisms for imposing user’s feedback on questions and on answers.

Negative feedback on a question q , f_n , is spread towards all of its associated answers, A_q , and marks all of them as non-relevant answers to the user’s query, $Y_{a(i)} = 0 \quad \forall a \in A_q$. On the other hand, positive feedback on a question, f_p , results in increasing the relevance of all its associated answers, $a \in A_q$, to the current query according to Equation 2.2 in which $Y_{a(i)}$ is the relevance score of answer a in relation to the user’s query in iteration i and $Y_{a(0)}$ is the relevance of answer a in relation to the question q .

$$Y_{a(i)} = Y_{a(i-1)} + \max_{a \in A_q} Y_{a(i-1)} \quad \forall a \in A_q \quad (2.2)$$

The feedback f on an answer a , is similar to the feedback on documents in the single-level datasets and is calculated according to equation 2.1 for $d \equiv a$. The only difference

is the initial value of the relevance score. In this case, $Y_a(0)$ is the initial relevance score of answer a toward its parent question q . In case there is a feedback in an answer, and its related question also has a feedback, the answer’s feedback is considered without being affected by the question feedback.

In our system, the user has the option to provide feedback on questions and/or answers. The evaluation presented in Figure 2.2 shows that if the feedback is provided on both questions and answers, a higher performance is achieved compared to scenarios where feedback is provided in just one of them. The feedback only on questions produces higher NDCG@10 compared with feedback on only answers because the feedback on a question provides more information (training instances) than the feedback on an answer. The combination of both also results in better overall performance.

2.4.4 Evaluation

Figure 2.3 shows the NDCG@10 and Recall@10 achieved by the best configurations for our proposed method across 8 iterations. We utilized reinforcement of positive and negative feedback instances and added novelty documents to show the results for a pretrained and a non-pretrained model. The results show an increase in NDCG@10 across all datasets.

We have considered 2 instances of feedback per iteration for MQ2007 and MQ2008, and 1 feedback on question and 2 feedback on answers are considered for SemEval 2016 and 2017. The increase in recall shows that our system is not only reordering documents in the top 10 results but also is finding new relevant documents over time. This is true for all evaluated datasets.

For instance, in SemEval 2017, the system was able to achieve more than three times

Table 2.2: The MAP calculated on SemEval 2016 and 2017 datasets compared with results of two baseline models and the top 3 results from the SemEval competition. Our result represents the configuration with least amount of iterations to achieve a higher result.

SemEval 2016	MAP@10	SemEval 2017	MAP@10
Baseline 2 (Random)	15.01	Baseline 2 (Random)	5.77
Baseline 1 (IR)	40.36	Baseline 1 (IR)	9.18
SemanticZ-primary	51.68	KeLP-primary	14.35
Kelp-primary	52.95	bunji-primary	14.71
SUper team-primary	55.41	IIT-UHH-primary	15.46
Our pre-perf-it7	55.19	Our nonp-perf-it3	16.66

the initial NDCG after only three iterations considering the non-noisy feedback in the non-pretrained model setting. It is expected that the noisy feedback performs worse than the perfect one. By comparing these curves, it is possible to see that the system is sensitive to wrong feedback. However, even then, it can improve the ranking and recover but in a slower pace. Although we have achieved higher performance comparing to the best algorithms in the SemEval 2017 competition, our method does not give the same performance in the first iteration.

As the results for NDCG of SemEval in Figure 2.3 suggest, the proposed method has beaten the winner of the competition by 11% on the 2017 competition when there is 1 Question and 2 Document feedback per iteration for 5 iterations, and by 7% on SemEval 2016 at iteration 7. We compared our results with baselines and winners [44, 43] from the competition, considering the Mean Average Precision, the main metric used in the SemEval competitions, on the test data and reported in Table 2.4.3. In this table, we report the results that achieve the best result for MAP in less amount of iterations. For 2016 we consider the pretrained model with perfect feedback, on iteration 7 and for 2017 the non-pretrained with perfect feedback on iteration 3. In Table 2.4.3, we report the evaluation on the LETOR datasets. For MQ2007 and MQ2008 we compared with the online model presented in [22]. The baseline in the graph represents their purely exploitative pairwise algorithm. We achieved higher results with the non-pretrained model on perfect and noisy feedback at iteration 2 on MQ2007, and our pretrained model achieved better base results, so we include the result for the first iteration of noisy feedback to show that our method maintains a good result. On MQ2008 the results follow the same trend, and we achieve better results on iteration 3 and 4 on perfect and noisy feedback, respectively. Even if compared with the values in the final iteration (1000) in Hofmann’s work, which is reported in a graph

Table 2.3: NDCG for MQ2007/2008 compared with perfect click model from [22]. Our result represents the configuration with least amount of iterations to achieve a higher result.

MQ2007	NDCG@10	MQ2008	NDCG@10
Hofmann-pwise-r0	0.377	Hofmann-pwise-r0	0.490
-	-	-	-
Our nonp-perf-it2	0.418	Our nonp-perf-it3	0.527
Our nonp-noisy-it2	0.395	Our nonp-noisy-it4	0.504
Offline LambdaMART	0.440	Offline LambdaMART	0.506
Our pre-noisy-it1	0.461	Our pre-noisy-it1	0.510

for MQ2007, their model stabilizes at around 0.55 NDCG, while ours can achieve much higher results, reaching 0.62 on the pretrained with perfect feedback model at iteration 7.

Although this project did not consider optimizations for speed, since we are proposing an interactive system, we also measured the execution time of each iteration. In all evaluated datasets, all the features for every query-document pair that will be ranked are available. Thus we mostly only measure the model training time. Overall, every iteration takes around 4.6ms. However, these numbers vary across iterations based on the amount of feedback count and model type. The pre-trained model is expected to have a higher execution time since its model is bigger from the beginning. Also, the model is increasing throughout the iterations, from the addition of trees, so it gets more expensive at each iteration; however, in our tests, the slowest iteration took around 11ms on a *Intel i7-7820X @ 3.60GHz CPU*.

2.5 Conclusion

In this chapter, we have proposed an interactive LtR method, which improves the ranking of retrieved documents through explicit relevance feedback. This system is evaluated on the Community Question Answering task, besides the general information retrieval datasets, and has shown significant improvement over the state-of-the-art. Our proposed method is capable of improving the ranking in a few iterations, making it a valuable option, specially when no training data is available.

As future work, it is possible to utilize an information retrieval module with higher recall, indirectly improving the LtR performance. Besides, generating context-aware features using word embedding is another area for improvement.

Chapter 3

Visualizations for Learning to Rank Interpretation

This section is, in part, based on the paper submitted to the Computer Graphics International 2020. The thesis writer is the main author and contributor to the paper, being responsible for the conception, design and implementation of the RankDiff and Cumulative Feature Gain visualizations proposed. He was also responsible for the implementation of the CRG, for the analysis of all visualizations, most of the writing of the paper.

3.1 Introduction

Over the past years, Machine Learning (ML) has been used to solve a diverse range of problems. Most ML models work by finding patterns and relations in data (indirectly) optimizing an objective function, e.g., the accuracy of classification models. Although ML models have attained a relative success, in many cases, the patterns found by a model are not meaningful in the real world or are not considered correct from a human perspective. Therefore, not only the degree to which a model can optimize an objective function is relevant, but also the understanding of how models make decisions is more and more critical. This gives rise to a currently hot topic for research involving model understanding and interpretation [66].

Learning to Rank (LtR) is a relevant area of research in ML. LtR models are used in many commercial applications from advertisement sorting [30] to general information retrieval [36]. As in most ML areas, much effort has been put into creating better LtR models [63], focusing on metrics, such as Normalized Discounted Cumulative Gain (NDCG) [26] and Mean Average Precision (MAP). But little or no effort has been put into understanding how data items are ranked. The same is true for studies that consider interactive learning to rank settings [7], in which models are trained through user feedback. Consequently, there is an acute need for new techniques to support model interpretation in LtR activities; otherwise, meaningless patterns can be used to drive sensitive decisions.

In general, there are a few different approaches to perform ML model interpretation [13],

which extend from global model interpretation, that tries to understand the entire model, and local interpretation, that provides a more detailed analysis on individual instances. Among the approaches that give support to interpretability, single tree-based models are known for being easily understandable by merely showing the list of decision rules or using a tree diagram. However, in more complex models, such as tree ensembles, the number of rules increases to the point of being infeasible to be analyzed. With this increase, a summarized approach must be defined to keep the model inside understandable to a user. One way of achieving this is by making it possible to analyze many rules at the same time, show their overall influences without cluttering the resulting view.

This work aims to fill this gap by using novel visualization techniques to explain better how LtR models rank elements. Our visualizations provide information about the most important features to a specific ranking result and what differentiates the elements' positions in the ranking. We propose three visualizations. One model-agnostic that considers the final ranked list to compare two different rankings. And two visualizations, applicable to any Regression Tree based ML model, that explains the ranking by showing how the documents are scored considering their features and allowing a detailed comparison of which rules influence the final ranking.

Throughout this chapter we use the terms *document*, *element* and *instance* interchangeably. This work is organized as follows: on Section 3.2 we perform a literature review of related work on the field of model transparency and visualization. Section 3.3 details the LtR model utilized and the proposed visualization methods. On Sections 3.4 and 3.6, we delineate two case studies to show the usefulness of our visualizations. Section 3.5 details the solution considered in Sec 3.4. Finally, we include discussion and limitations in Section 3.7 and final conclusions on Section 3.8.

3.2 Related Work

In this section, our focus is to present papers related with the visualization of LtR models. However, the underlying techniques employed by most LtR algorithms are also used in other tasks. Because of that, we also review studies that are either generic or applied to similar models. We also discuss applications of visualizations in model interpretability and some related works in the area of Interactive Recommendation Systems.

3.2.1 General Models for Interpretation

Given the number of different Machine Learning models, some frameworks for model interpretation are agnostic and provide some level of interpretation over the results of any model. This is usually done by considering only inputs and outputs, without looking in the internal parts of the model.

Manifold [74] is a generic framework capable of analyzing many types of ML algorithms. The goal of the framework is to assist analysis, debugging, and comparison of results. In [32], the authors propose a system to interpret prediction results, allowing inspection of prediction behavior with manual changes to instance features. The study described in [57], proposes a new visualization to show the error “contribution” of different models across different scenarios. LIME [52] is another tool created for model interpretation, highly focused on local interpretation. None of these have focused on LtR and are limited in their instance comparison capabilities. Our visualization technique allows the comparison of a selection of documents at both feature and rule levels.

The study presented in RuleMatrix [41] shows a visualization focused on explaining rule-based models. Their system performs an approximation of decision rules, through model induction, of a black-box model, like Neural Networks, and then uses visualization to help users to explore the data and understand the classification model. Their visualization, however, is focused on interpretation of classification models while ours focus is on interpretation of LtR models.

BaobabView [65] proposes a system to build and visualize tree-based models. However, it does not apply to more complex ensemble models composed of multiple trees. VISE [61] is another work considering trees, which allows the comparison of small ensemble models, but is limited on its exploration and visual capabilities, allowing only the exploration of some branches of a single tree at a time. The visualizations used in these works, however, are not directed to LtR interpretation, while our visualizations allow the view of all rules being applied to a selection of documents, displaying its accumulated effect to the document’s score.

3.2.2 Ranking Interpretation

Recommendation systems are related to Learning to Rank techniques in the sense that both seek to provide a ranked list of results. They mostly differ from the way the algorithms

are designed and the type of training data considered. The survey presented in [20] recaps recent research studies in Interactive Recommender Systems that focus on transparency and explainability of the recommendations. These are based on collaborative filtering, content-based, or hybrid approaches.

Collaborative Filtering recommenders usually explain recommendations based on other users' interests, leveraging similar users or social networks to define the recommendations [17, 45, 15]. Other systems base their recommendations on content, MoviExplain [62] provides a table-based interface that displays a single important feature related with user information to justify the decision, such as how many types of similar movies the user reviewed. Intent Radar [55] utilizes a radial visualization to display relevant terms to the users' initial search and allows the user to guide the search by moving the words in the space. However, visualizations on recommender systems do not consider an LtR model and are usually focused on the exploration of results and use high-level descriptions, while we consider a LtR model, focused on the comparison of a selection of elements, showing how specific features from the element influence its score.

Techniques that employ visualization in item ranking typically have their visual representations and ranking based solely on feature weighting. LineUp [16] and Rank as you Go [11] propose similar visualization approaches for ranking explanation. The user can change the weight of the terms or features, and the resulting rank is updated, displaying the contribution of each feature using a stacked bar chart. In [31] the user can use a node-link visualization to distribute the results in a space delimited by keywords. All these approaches are applied to simple ranking functions that use term weighting and do not consider a Machine Learning model that learns based on user feedback. As suggested in a recent study [42], the usage of wrapped bars is beneficial for explaining ranked lists. Nevertheless, they make difficult the comparison of the same feature for multiple items. We apply a similar idea in one of our visualizations to show rule contribution, but, for better individual feature comparison on elements, we utilized a different approach which compare the feature values and how it influences the element score.

3.3 Method

Typically, LtR are supervised models that learn from query-document pairs associated with a relevance label. Some studies consider learning based on click models [24], where the

ranking model is learned based on the history of clicks for different searches. In contrast, others consider an online scenario where the models are updated on the fly, based on the implicit feedback provided by the click data generated by the users [22]. Relevant to the latter type of model is that systems can reflect changes of interest of the general public over time.

In cases where there is a lack of training data or in information seeking tasks where interests vary, it can be challenging to have a generic model that attends all needs. Thus it is necessary to follow a strategy that allows a dynamic model to reflect the user needs. There are few studies in interactive learning to rank [7, 47] showing that user feedback can effectively retrieve relevant data. However, it can be challenging to interpret the reasoning behind the ranking of the results.

In the system implemented for this work, we utilize the LambdaMART [69] technique as the base ranking algorithm. LambdaMART is a popular LtR model, based on a forest

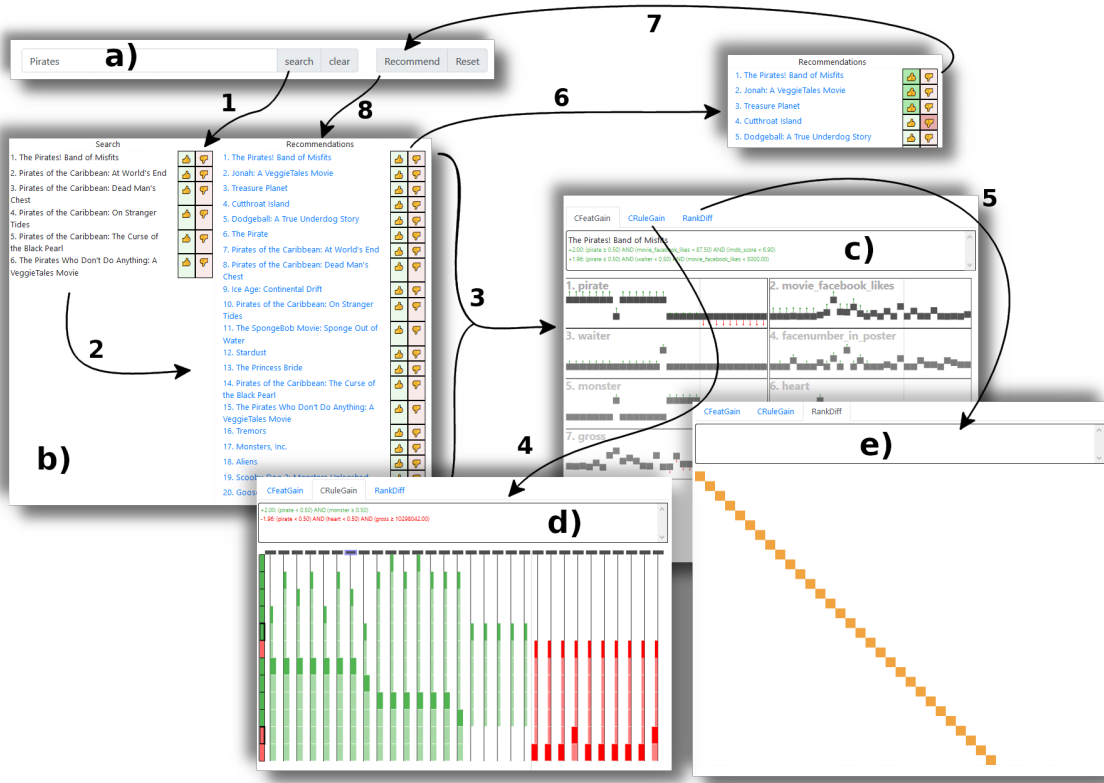


Figure 3.1: Overview of interface and proposed visualizations and navigation flow. a) Search bar. b) Results from search. c) Cumulative Feature Gain. d) Cumulative Rule Gain. e) RankDiff visualization.

of boosted regression trees. The model is built by optimizing the lambdas that minimize the pairwise error regarding the NDCG metric. The lambdas are obtained from the NDCG values resulted from swapping the training pairs in different orders. Given that E is the group of elements, and e represents one instance, the score S_e is calculated from the forest of regression trees. Each tree in the forest is composed of multiple branches that end up in a leaf node with score s_b . Each instance is evaluated for all trees and falls in one leaf node of a tree. Assuming that L represents the group of leaves that the document falls into considering every tree, the score of an instance can be calculated as $S_e = \sum_p^L s_p$. The score of the instances is used to sort the list of entries E to provide the final ranked list.

Our system learns considering the user in a feedback loop, that allows the user to search and interpret the results using our visualizations as shown in Figure 3.1. The figure shows the search bar (a), the list of results found (b), and the three visualizations proposed: Cumulative Feature Gain (c), Cumulative Rule Gain (d) and RankDiff (e).

The numbers in the figure represents the flow of the system. Initially, the user enters a query and receives the initial set of results that match with the user’s query (1). This set is expanded into a recommendation list (2). The recommended items are represented in the 3 proposed visualizations (3), that are displayed in a panel that allows switching between active visualization (4, 5). At any point, the user may provide relevancy feedback on a set of items (6), followed by asking for a new recommendation (7), which returns a new set of recommended items (8). Steps 6, 7 and 8 may be repeated many times. After step 8, the visualizations are updated to represent the new information.

All visualizations were implemented in a web environment, using D3 [5], and backend implementations were made in Python, using pyltr¹ as the lambdaMART implementation.

Next, we describe the three proposed visualizations, the first supporting understanding the difference between two ranked lists, the second providing information about feature usage on document scoring and the last, providing information of which rules affect the score of each document.

3.3.1 Ranking Comparison Visualization

The first visualization, *RankDiff* (Fig. 3.1 e), is a pairwise matrix representation that can be used to compare the difference in the ordering of two different ranked lists. Given that

¹<https://github.com/jma127/pyltr>

i and j represent the rank position of an element e , in the baseline ranker b and the main ranker m , respectively. Then e can be indexed in a matrix at (j, i) , in which the first index represents the row, and the second the column. Considering W and H as the maximum available space in the X-Axis and Y-Axis, and N the selected number of instances from the group E to be displayed, each element can be displayed as a square of size $\frac{\min(W,H)}{N}$.

To see the difference between two rankers, the visualization is first initiated by positioning the elements with coordinates (i, i) , so the items are aligned in a diagonal following the ranking of b . Then, to compare with the ordering of the main ranker, the elements are transitioned to (j, i) . This transition allows the perception of items that are moving up or down in the ranking. The closest the items are to be aligned to a diagonal, from the top left to the bottom right corner, more similar is the ordering of the items for both rankers.

Because of space constraints, when ranking many items, only the top N items ranked by each ranker are shown. In this case, the elements from ranker b that do not appear in the list ranker m , thus were ranked lower, are considered to have a rank position below the items of m . This is considered so that the elements do not pop up or disappear, making it clear that different items are entering or leaving the view. This effect can be seen in Figure 3.14.

Although this visualization can compare two unrelated rankers, it is most interesting when analyzing the evolution of the ranking function over time by using m as the latest ranker and b as the previous state. Then by sequentially swapping the main ranker to be the base ranker while a new ranker takes the main place for the next iteration, it is possible to identify how items move across many iterations. In this case, we draw a trail behind the moving entries, showing how the instances change their ranks over time. Trails that are above the main diagonal indicate elements that moved up in ranking. Those below the diagonal are from instances that moved down. Items of interest can also be color-coded differently to provide a better overview for a specific context, for example, to indicate item's true labels (in case of labeled benchmarks) or highlight items that received user feedback.

3.3.2 Cumulative Feature Gain Visualization

We designed the Cumulative Feature Gain (CFG)(Fig 3.1 c) visualization to provide an overview of the feature utilization by a model and how they influence the ranking score of a selection of items. In this visualization, each feature is represented by a cell C_f , where f represents a feature index. An overview of a single feature cell is shown in Figure 3.2. The

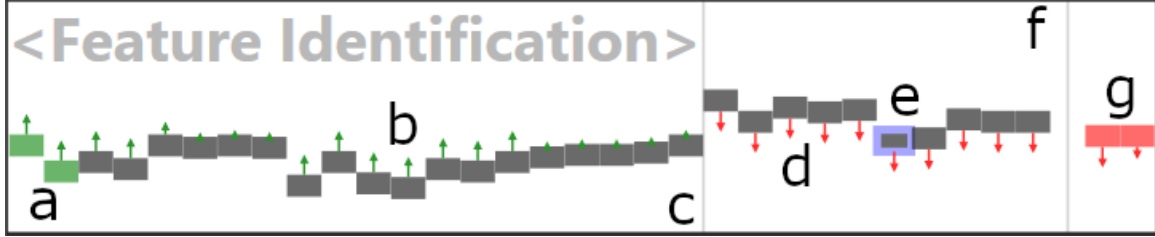


Figure 3.2: Example feature cell utilized in the Cumulative Feature Gain visualization. a) Elements marked as relevant. b,d) Arrows represent score influence of this feature over the instance. c) Division line, on the left are the top 20 relevant items, and on the right the bottom 10. e) An element highlighted by selection. f) The right most division line is used to separate instances of interest that did not appear in the previous lists. g) Elements that were marked as irrelevant. Elements are positioned sequentially based on their ranking order in the X-Axis. The position on the Y-Axis encodes the feature value for each element.

cells are displayed in a grid, of C columns by R rows, where we consider C as a constant with value 2 and R calculated by $R = \frac{W}{C}$. Then the horizontal and vertical size of a cell can be defined as $w = \frac{W}{C}$ and $h = \frac{W}{R}$, respectively. Given that P_r is an array that contains the position of every instance in the ranker r and that μ represents the size of the element, given by $\mu = \frac{w}{N}$. The position of elements in the X-Axis (e_x) encode their rank position for a ranker r , following the equation:

$$e_x = \mu \cdot P_r[x]. \quad (3.1)$$

The Y-Axis is used to encode the feature value for each item displayed in the cell. In both axes, the visual representation is scaled to fit the available space in the cell, considering the number of instances and the respective feature range F_{min} and F_{max} of the elements being displayed. To be more intuitive, each cell displays in the top the feature name or other relevant identification, so we use a constant gap λ at the top and a proportion of the space for the arrows θ , explained later in this section. Given that F_e is a list of feature values for the element e , the value of a feature f can be accessed by $F_e[f]$. Assuming that the Y-Axis has the origin at the top of the cell, the position e_y for an element can then be calculated as

$$e_y = h - \frac{F_e[f] - F_{min}}{F_{max} - F_{min}} \cdot (h - (2\mu + 2\theta h + \lambda)) - (\mu + \theta \cdot h) - \frac{\mu}{2} \quad (3.2)$$

We divided the horizontal space of the cell to represent different groups of elements. In our case, we show first the top 20 instances, then the last 10 instances, and then instances

that may be of interest that did not appear in the previous lists (Fig. 3.2 *c* & *f*). The groups are split using a line.

Element colors can be used for various purposes, but here the elements that received feedback were colored with distinct colors to indicate positive (green) and negative (red) feedback (Fig. 3.2 *a* & *g*). The contribution of the features for each element is calculated considering the decisions of the forest model. These contributions are shown as arrows that point up or down (Fig. 3.2 *b* & *d*) based on the average contribution of rules that use the feature for the element. To better identify the overall contribution of features, these arrows are scaled to indicate the contributions for the features of the N displayed elements.

The feature contribution of each instance D_{fe} is calculated by the sum of the leaf score for all relevant rules. Relevant rules are extracted from the branching paths B_{fe} for all trees in the model that uses the feature f in which the element e falls into. This is done using

$$D_{fe} = \sum_b^{B_{fe}} L[b] \quad \text{where } L[b] \in \mathfrak{R} \quad (3.3)$$

where L is an array of leaf values for the branches and $L[b]$ is the leaf value for the branch b . Worth noting that this is different from the final instance score S_e , since now we only consider the leaves that were in a branch that considered the current cells' feature. The sum of the absolute contribution value of a feature for every instance provides the feature importance, and is calculated as

$$I_f = \sum_e^N |D_{fe}| \quad (3.4)$$

The intuition is that we consider of high importance the features that apply more difference to the documents scores.

To show the feature contribution for every document, an arrow is displayed pointing either up or down, based on the signal of D_{fe} and with length l normalized using

$$l = \frac{|D_{fe}| - I_{f_{min}}}{I_{f_{max}} - I_{f_{min}}} \cdot h \cdot \theta \quad (3.5)$$

where the limits $I_{f_{max}}$ and $I_{f_{min}}$ are the maximum and minimum importance for the feature. Such arrows are then positioned using

$$x = e_x + \frac{\mu}{2} \quad (3.6)$$

$$y = e_y + \frac{\mu}{2} \cdot \text{sign}(D_{fe}) \quad (3.7)$$

The feature importance is utilized to decide the order and which features to show. Features with importance 0 are omitted since any tree uses them. By hovering one instance of one of the cells, the system displays in a space on the top of the visualization, all rules that utilize the cell's feature (Figure 3.10).

3.3.3 Cumulative Rule Gain Visualization

The third proposed visualization, Cumulative Rule Gain (CRG)(Fig 3.1 d), displays all rules, from the ranking model, that are used to score each instance. The visualization was built considering documents represented on the X-Axis as columns, and the rules taking the Y-Axis as rows. Documents that are displayed are distributed in the space and have a base guideline going down through all rules. Rules that do not affect visible documents are omitted to save space, and we will refer to the visible set as U .

This guideline represents the origin, with 0 score, and is used as a base to draw a bar chart that represents the accumulated score for every rule, from top to bottom, for each document. This accumulated score, S_{acc} , is calculated similarly to Equation 3.3, but considering the U set of rules instead of B_{fe} . All rules in the set U are applied to at least one document in the view, and when it is not, thus not changing the accumulated score, the rule is represented using a more transparent color. All of these can be seen in Figure 3.3, that displays a diagram that explains this visualization.

Given that V_{min} and V_{max} represent the upper and lower limits of the accumulated score for the visible set of documents E . The bar-chart for every document assumes these values as baselines to normalize the size of the bars, similarly to Equation 3.4. Where $|D_{fe}|$ is replaced by $|S_{acc}|$, I_f are replaced by the V limits, h is the horizontal space for representing a document, and θ is 0.5, since the bar takes either the left or the right side of the origin line. Note that negative accumulated scores are displayed to the left of the origin line. The way the cumulative scores are displayed as bars is represented in Figure 3.3, for instance at E2 getting some score from rule 2 (R2) and later from rule 6 (R6).

The documents are sorted on the X-Axis following the ranking order, and the features can be ranked in different ways, e.g., all positives first, then all negatives to show the contrasting rules. Here we keep them in order of creation. Similarly to the CFG, we also

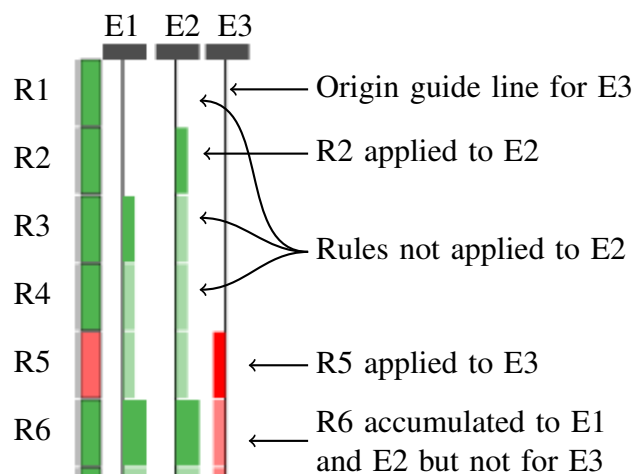


Figure 3.3: Cumulative Rule Gain, looking only at 3 ranked elements (E1, E2, E3) and 6 ranking rules (R1 to R6). Color red and green of the rules indicate a negative or positive contribution, respectively.

have a dividing line to split specific groups of documents.

Both rules and the cumulative bar chart are color encoded using a red to green color scale, which represents negative and positive values, respectively. A small rectangle is also displayed on the top of every column to represent the element, which can be color encoded to represent relevant information to the context, such as true labels or marked documents. In Figure 3.13, on the top, there are 3 elements with green color and one red by the end, indicating that those documents received positive and negative user feedback, respectively.

3.4 Case Study: Ranking Model Evolution

In this case study, we assume a LtR system that performs re-ranking of the results based on user feedback. The user can select documents as relevant or irrelevant and then ask the system to refresh the results. Using the RankDiff visualization, we show that it is possible

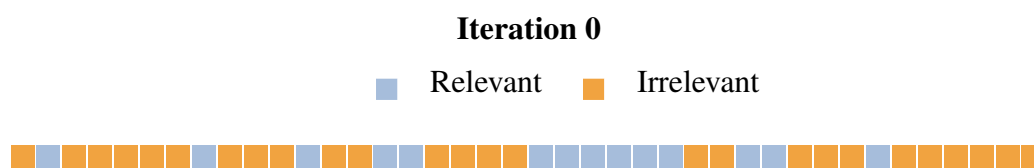


Figure 3.4: First iteration result on the RankDiff case study. The squares represent items sorted by ranking score, from left to right in descending order.

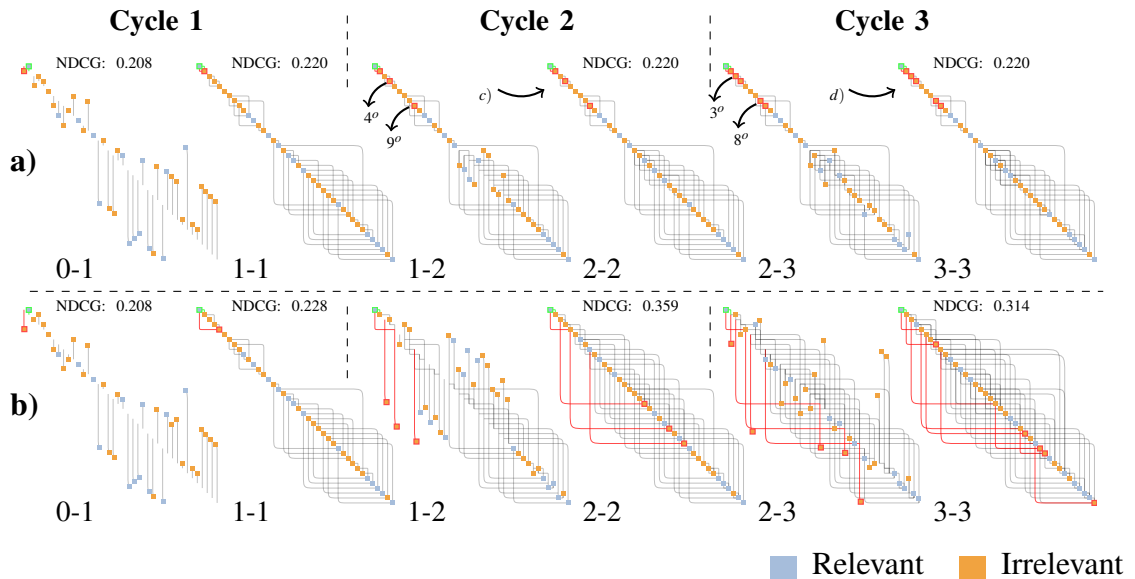


Figure 3.5: Behavior of the Learning to Rank algorithm over 4 iterations. a) the behavior of the original algorithm. b) the behavior after changes to the algorithm from the intuition taken from the previous result. c,d) Feedback that did not change position.

to better understand the behavior of the model and potential flaws in the learning model.

3.4.1 Dataset

For this case study, we selected one query in the validation set of the **MQ2007-Fold1** dataset as an example of how our RankDiff visualization can be used to understand at a high level how the ranking changes over iterations. **MQ2007** is one of the datasets provided in the LETOR 4 [48] collection, that contains many groups of query-document pairs with relevance labels. This is a standard dataset used on the evaluation of LtR algorithms.

3.4.2 Usage Scenario

The first results are retrieved from the information retrieval system and ranked by a model that approximates the order of a TFIDF score. From the initial results, we can then analyze how the ranking order changes after receiving feedback. Considering that, the first results retrieved for the query to be analyzed are represented in Figure 3.4, where relevant documents are represented by the blue color and irrelevant documents by orange. The leftmost documents are at the top and the rightmost are at the bottom of the ranking.

Given this initial ranking, now we start our feedback iterations. In the row **a)** of Figure 3.5, are displayed the cycle of 3 iterations of feedback. For every cycle, a transition is shown, where items are moving in the vertical axis, showing where they are going to after the feedback. After the transition, the items are aligned in a diagonal, showing the final ranking order.

On the first iteration, one relevant and one irrelevant instance receive positive and negative feedback. A user can select these instances by looking at their content and judging it either relevant or not. Here the user is simulated by an oracle, that knows the true labels of the instances. Any number of instances can be marked as feedback, but in this study we always consider 2 instances per cycle. The instances that received feedback in the first cycle are the first two instances, that are highlighted in red and green in the transition **(0-1)**. This transition then is followed by the final ranking at **(1-1)**, which represents the result at iteration 1.

Then 2 new instances of feedback are selected at positions 4 and 9, as seen in the transition **(1-2)**. The newly marked documents did not change positions (Fig. 3.5 c), but there were a few changes in the middle of the result set. Such changes are not visible to a user since we consider, in this example, that only the top 10 instances are shown to the user. The final result for this iteration is shown in **(2-2)**.

At the last iteration, the transition **(2-3)** shows 2 new irrelevant documents selected at positions 3 and 8, again resulting in no meaningful changes in the top-ranked results, as seen in the final state **(3-3)** with final NDCG@10 of 0.19.

As displayed by this example, at iteration 1, where there are some feedback on wrongly ranked instances, e.g. a relevant one after an irrelevant one, the model perform a meaningful change in the ranking. However, after this iteration the model is stagnated and does not change further since the following instances of feedback are, by the model's perspective, already ranked properly, because they are below the relevant instance.

A more usual behavior to most users would be that the irrelevant result should probably not be visible in the top N results after receiving feedback. Considering that, it is possible to change the algorithm by adding another objective to the training of the model. The idea is to make changes so that the feedback is considered more thoroughly, leading to a better ranking function.

The change to the interactive algorithm will be discussed in more detail in Section 3.5,

but as an overview, we made the algorithm reconsider some items that were not in the feedback list to be in between them. This change forces the algorithm to look for more features that split the feedback instances, thus avoiding getting stuck in a solution that only properly ranks the feedback instances.

After updating the algorithm, we can verify the behavior of the model considering the same starting state. Looking in the row **b**) of images in Figure 3.5, it is possible to see how the behavior has improved. The first iteration receives the same instances of feedback. In transition **(0-1)** is visible that the selected irrelevant instance is being forced down more positions than the previous method, being ranked at position 5 versus 2 from before as seen in **a**) vs **b**) at step **(1-1)**.

The transition **(1-2)** shows two new instances that were marked as irrelevant at positions 3 and 9 to be pushed down along with the previously selected instance. All of the marked negative instances have now been pushed below the top 10, as the state in **(2-2)** clearly shows. At the next iteration, two other irrelevant instances are marked, and the transition **(2-3)** takes us to the final state **(3-3)**, achieving final NDCG@10 of 0.31, which indicates a better ranking order than the previous method.

Although, on average, the results considering the NDCG, MAP and Recall metrics seemed to be satisfactory as in the *before* setting represented in Figure 3.6, where they seem to improve over time. The RankDiff visualization exposed one flaw that was not visible through the metrics average. The visualization showed that, in some cases, the items that received negative feedback were still being ranked in the top positions, as indicated by the arrows (Fig. 3.5 c & d), even after being explicitly marked as irrelevant documents, showing that the model was stuck and not having any significant changes.

One could argue that by simply removing these items from the list, the problem would be solved. However, the fact that those items are still at the top indicates that the model might have failed to properly learn from that feedback.

Knowing the problem, we can then define a new metric that will capture whether the model is getting stuck, thus allowing us to properly numerically evaluate our solution to the issue. We define the $NDCG_{learn}$ in Equation 3.8, where D_r represents the list of labels for the current top K documents where relevant items have value S_{max} and all others S_{min} with feedback are considered, similarly D_i represents a list of labels where irrelevant documents have value S_{max} , and S_{min} for all others. S_{max} and S_{min} represent the range of the true labels

of the dataset. As displayed in the last graph in Figure 3.6, we can see that the $NDCG_{learn}$ is mostly stable for many iterations in the *before* configuration, indicating the stagnation.

$$NDCG_{learn} = \frac{NDCG(D_r) + (1 - NDCG(D_i))}{2} \quad (3.8)$$

Lastly, after tuning some parameters of our new method in the validation set of **MQ2007**, we then re-evaluate using the test set. The average of 30 runs over the test set is reported in Figure 3.6. The new method achieved comparable or even higher levels of NDCG and MAP. Considering the $NDCG_{learn}$, we see that the new method is superior, indicating that our solution should have minimized the stagnation problem, presented in this case study, for more queries. Note that the $NDCG_{learn}$ is undefined at iteration 0, since it depends on existing feedback, thus we just use the value from Iteration 1 when plotting the graph, to keep the axis consistent.

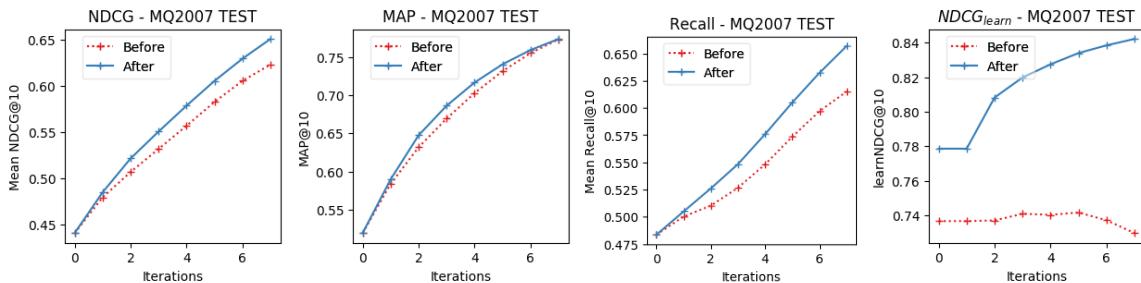


Figure 3.6: Evaluation on MQ2007 average for all folds on test set. *Before* represents the results before the changes to the method, and *After* changes to the method, developed after insights taken from the RankDiff visualization.

3.5 Improving the Learning to Rank Method

Earlier in this chapter, the RankDiff visualization was demonstrated in a Case Study. The case study described an issue where the iterative learning method stagnated and could not improve the lists of results any further even after receiving multiple instances of feedback. In this section I go over more details on the issue and the proposed solution.

3.5.1 Problem Analysis

A closer analysis and considerations regarding the LtR model utilized provided an insight of why it was happening and how the problem could be minimized. As explained

in Section 2.3.2, LambdaMART learns based on the lambda obtained from the difference in NDCG from pairwise changes in the list of training instances.

Considering that X is the current list of training instances and that the current model M already ranks X in a perfect order. By extending X with new instances t , labeled in such a way that if ranked by the model, M , results in the same order of the original elements in X . And that the extended elements being ranked anywhere, as long as they respect the ordering from their labels, e.g. if labeled good it is in a better rank position than bad labeled instances, LambdaMART will not enforce any changes to the current model. This happens because all possible lambdas from pairwise changes will have no effect and result in no changes to the model.

A new metric to evaluate if the model was stagnated during learning was defined in Equation 3.8. The metric measures how well the model is representing the feedback in the final list, and is considered better for higher values.

3.5.2 Solution Evaluation

After understanding why LambdaMART was getting stuck, an intuitive solution to the problem is to include more training data so that we stimulate the model to consider more information that differentiate the negative feedback elements from the good feedback. The proposed solution was to then include a set t that would make LambdaMART find lambdas that would change the current ranking function. The group of elements t is defined by selecting the top j instances that did not receive feedback, and temporarily marking them with a static label l that is in between the instances with positive and negative feedback. The value l influences the lambda value seen by LambdaMART and causes different decisions inside the model, since lower l will make the model see the set t as more close to the negative instances, and vice-versa for higher values and positive instances.

A parameter fine-tuning was executed with a few different combinations of parameters on **MQ2007-Fold1**, displayed in Figure 3.7. The graph on the left shows that the $NDCG_{learn}$ achieved higher results, indicating a better behavior, regarding the stagnation problem. Also, on the right it is possible to see the influence of the change on the NDCG metric, that has a small improvement, specially after the second iteration. The results reported earlier, in Section 3.4, considered $l = 1$ and $j = 100$. In Figure 3.8 the same evaluations are performed for the MQ2008, Semeval 2016 and SemEval 2017 datasets on the

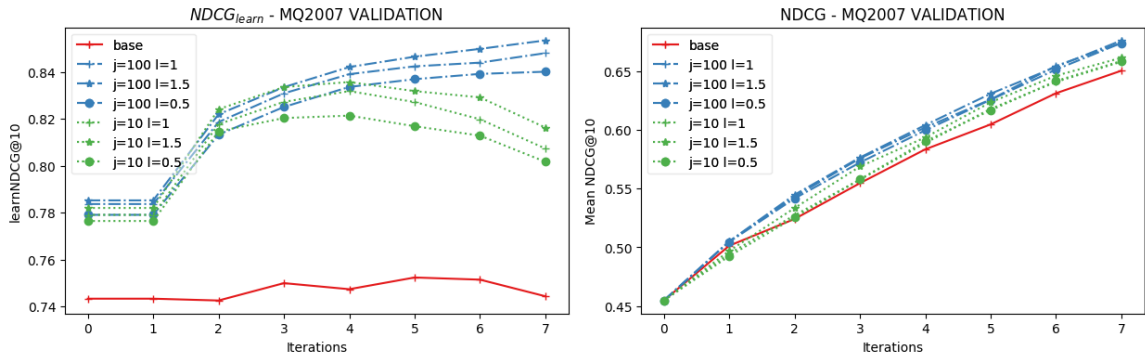


Figure 3.7: Evaluation of different combinations of parameters j and l in the proposed solution to avoid the stagnation problem.

test set. MQ2008 uses 2 feedback per iteration and the SemEval datasets use 1 question and 2 answers feedback per iteration, same to the settings used in Section 2.4. It is possible to see that, overall, the implemented solution is beneficial across all evaluated datasets.

3.6 Case Study: Understanding Ranking Order

In this section, we analyze how the LtR algorithm generated a ranked list of recommendations based on an initial set of results. We demonstrate how our CFG and CRG visualizations can be applied to explain the reasoning behind the LtR model results.

3.6.1 Dataset

For this case study, we utilized the **IMDB 5000** dataset [73] available on Kaggle. The original dataset contains around 5,000 movie entries with 28 attributes. The data contains general movie information, plot keywords, directors, revenue, review scores, actor information, and social media related scores.

The dataset was cleaned by dropping all rows with missing values, removing attributes with people’s names and their related social media scores, and excluding keywords related to adult content. We also transformed categorical columns to a one-hot encoding since our LtR model is based on regression trees and expect only numerical attributes. After preprocessing, the final dataset remained with 3,656 movies and 1,055 attributes, most of which are plot keywords that appear for more than 3 unique entries.

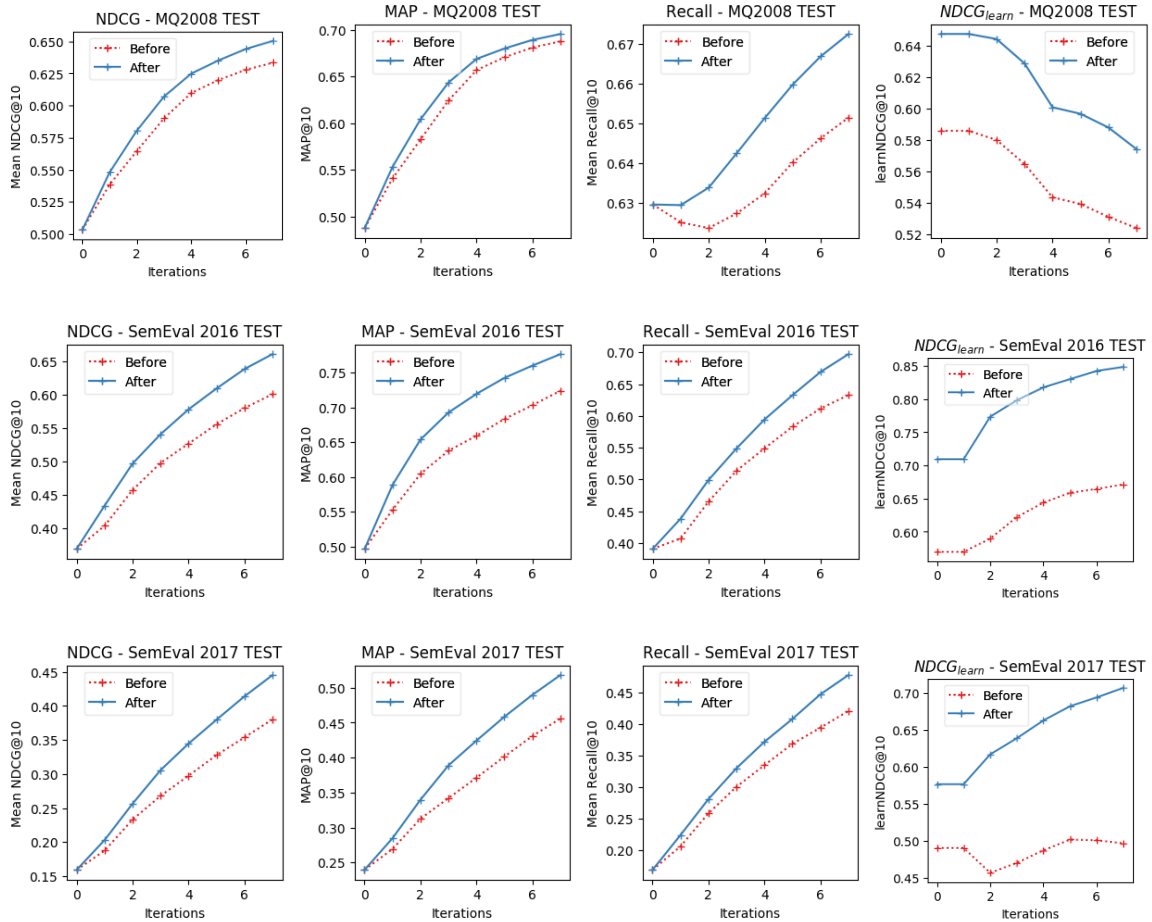


Figure 3.8: Evaluation on the SemEval 2016 and 2017 and on MQ2008 averaged for all folds on test set, average of 30 executions. *Before* represents the results before the changes to the method, and *After* changes to the method, developed after insights taken from the RankDiff visualization.

3.6.2 Usage Scenario

Consider a system that recommends similar movies based on a single search. In our usage scenario, the user enters the word “pirate” and the system initially finds 6 movies that relate with the search by their name. Then a LtR model is trained to find recommendations considering that those results may be relevant.

A list of recommendations is then created, seen in Figure 3.9. Many entries related to pirates are found, even some that do not match the exact term used in the search. Now, looking at the CFG visualization, Figure 3.10, the user can quickly understand what type of features are being used to generate the ranked list. As expected, the term “pirate” is a relevant feature, which is a plot key-term present in the movies found on the initial search.

Note that these key-terms are binary features and are divided by the regression trees at the mean value of 0.5, where lower values indicate no occurrence of the term, and the opposite its presence.

On the recommendation list, starting at position 16, we have many movies that are not related with **pirate**, such as “*Monsters, Inc.*”, “*Aliens*” and “*Scooby-Doo 2: Monsters Unleashed*”. By inspecting the CRG (Figure 3.12) we can have a better idea of what type of rules brought these results up. In the figure, we see that the last documents (**a**) are receiving +2.0 score from just one rule ($R1 : pirate < 0.5 \text{ and } monster \geq 0.5$), which can be seen in (**b**). Then, the elements are penalized in -1.96 by the rule ($R3 : pirate < 0.5 \text{ and } heart < 0.5 \text{ and } gross \geq 10298042$), which is perceived by the reduction of the accumulated score represented in the bars at (**c**). We select this positive rule ($R1$) and all the red ones ($R2, R3, R4$) to see what the model is considering as bad entries. Looking at the rule description on the top, we then see that the last documents received points for being “non-pirate” and “monster” types of movies from rule $R2$. Besides, they lost some points in rule $R3$ for not being pirate or heart related and having a gross revenue of over 10.29\$ million. We can also see that the bottom documents are scored by the three negative rules, that relate with “non-**pirate**”, “non-**monster**”, “non-**italian**” and “non-**heart**”.

The movie at position 8 is seen in the CFG, Figure 3.10, as not being related with “**pirate**” since it is represented on the lower part of the cell, however, by the movie name, this is wrong and indicates a possible data issue. Even though this item has value 0 for this feature, it receives a positive score, while the results at the end of the list, after the first division, receive a negative score on this feature. Note that the movie was on the initial search results. Thus the LtR model considered it as relevant when preparing the recommendations. Also, because the “non-**pirate**” feature entered in conflict with the others, the model had to look for other possible features to score the document, such as Heart and Monster, which are very related with the movie in question. This caused the model to rank other items related to those features higher as well.

Now consider that the user selected the first 3 movies from the recommendation list as relevant instances and the fourth as not relevant. The CFG for the second iteration is shown in Figure 3.11, which shows the top 10 important features. In this figure, we can see that new features started to be considered for ranking the documents. If we consider the documents that received feedback, it is possible to understand why the model included





















































Search		Recommendations	
1. The Pirates! Band of Misfits	 	1. The Pirates! Band of Misfits	 
2. Pirates of the Caribbean: At World's End	 	2. Jonah: A VeggieTales Movie	 
3. Pirates of the Caribbean: Dead Man's Chest	 	3. Treasure Planet	 
4. Pirates of the Caribbean: On Stranger Tides	 	4. Cutthroat Island	 
5. Pirates of the Caribbean: The Curse of the Black Pearl	 	5. Dodgeball: A True Underdog Story	 
6. The Pirates Who Don't Do Anything: A VeggieTales Movie	 	6. The Pirate	 
		7. Pirates of the Caribbean: At World's End	 
		8. Pirates of the Caribbean: Dead Man's Chest	 
		9. Ice Age: Continental Drift	 
		10. Pirates of the Caribbean: On Stranger Tides	 
		11. The SpongeBob Movie: Sponge Out of Water	 
		12. Stardust	 
		13. The Princess Bride	 
		14. Pirates of the Caribbean: The Curse of the Black Pearl	 
		15. The Pirates Who Don't Do Anything: A VeggieTales Movie	 
		16. Tremors	 
		17. Monsters, Inc.	 
		18. Aliens	 
		19. Scooby-Doo 2: Monsters Unleashed	 
		20. Goosebumps	 

Figure 3.9: Search results for “pirates” keyword. The left is a list of entries matching the query term, and on the right the system recommendations based on the left list.

these features. All the documents that received positive feedbacks were animations, “*The Pirates! Band of Misfits*” has a relation to the keyword “**scientist**” from one of its characters, “*Jonah: A VeggieTales Movie*” is related with a “**whale**” that is part of the movie and “*Treasure Planet*” is an adventure in space which is linked to the keyword “**planet**”.

The feature “**man with glasses**” is not related in any of the documents that received feedback. However, it is still being considered in the ranking. This is an artifact from the change to the training model performed on Case Study 1, which implicitly includes more training instances along with the feedback instances. By looking at the rule related to this feature, on the top of the Figure 3.11, we can see that it still relates to relevant features regarding the documents that received positive feedback. The negative feedback was in an “**action**” movie, feature which applies a negative score to some movies, besides other rules considering the absence of the keywords mentioned previously (Figure 3.13). This iteration is also interesting to be analyzed in the RankDiff visualization shown in Figure 3.14, which

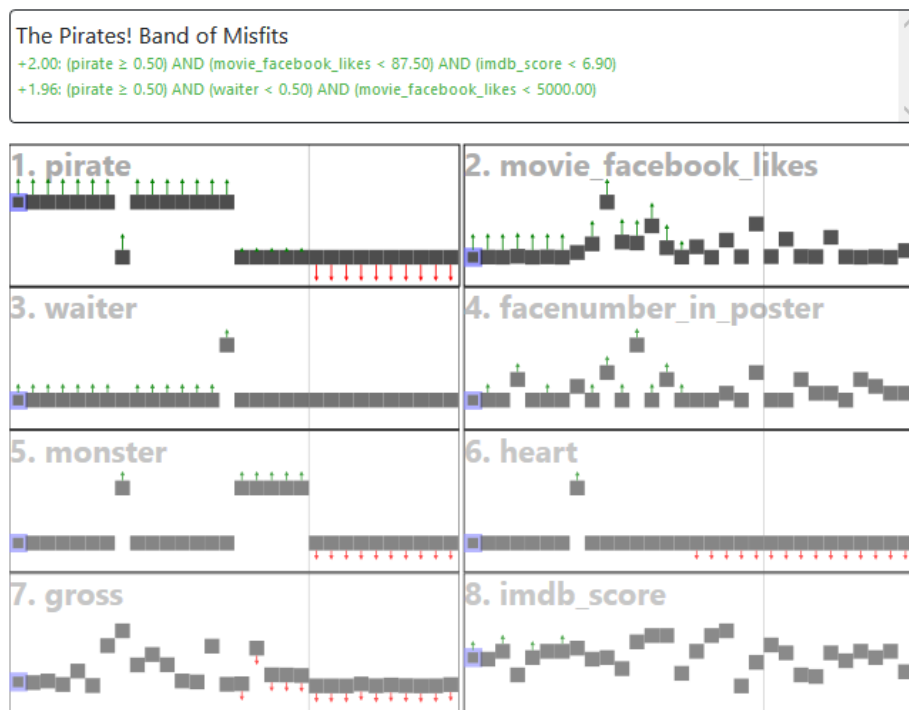


Figure 3.10: Cumulative Feature Gain view for the recommendations after the initial search. The rules on the top were shown after hovering the first element in cell #1.

indicates that some documents remained in the view, and new ones entered the top 20.

3.7 Discussion and Limitations

This work proposes three visualizations to be applied in LtR. One of them revealed a stagnation issue in the iterative LtR method that was not clearly visible in the standard metrics. After the issue was identified, a metric to capture the unwanted behavior was defined, so it was possible to evaluate potential solutions. Besides that, other visualizations are proposed to explain how elements are ranked by the models, making it clear what the model is prioritizing in the ranking.

Our work is not perfect and has some limitations in the proposed visualizations. For instance, our visualizations can not provide much assistance in analyzing the feature dependencies considered in the model. The CFG visualization tries to express the overall contribution of the features in an isolated form, thus not making obvious its relations with other features. We only try to provide this rule dependency information with basic textual

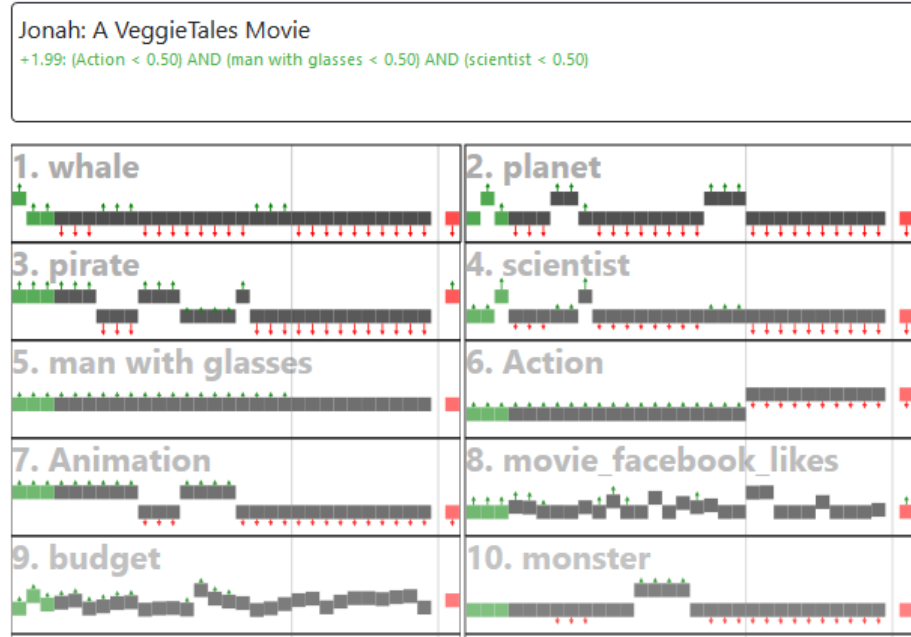


Figure 3.11: Cumulative Feature Gain view for the recommendations in the **second** iteration. The rule on the top relate to the first element in cell #5.

rule descriptions. The CRG shows the contribution of rules for the documents, but for models with many rules, the visualization can become extensive, and may require other ways to filter or select visible rules.

All visualizations have some limitations in the number of elements they can display, so we try to focus on showing interesting instances. Other approaches could be considered to select displayed instances, such as allowing the user to select specific instances to be analyzed. It is also worth noting that interpretation of the CFG and CRG is subject to user knowledge on the domain and understanding of the features.

3.8 Conclusion

Three visualization techniques are proposed in this work, viz., RankDiff, Cumulative Feature Gain (CFG), and Cumulative Rule Gain (CRG), each focused on explaining one different aspect of the Learning to Rank algorithm. With the RankDiff visualization, it is possible to quickly recognize how the ordering of two ranked lists differs, and to identify potential flaws that are not obvious when looking at standard LtR metrics. The visualization provided enough information to improve the iterative LtR method, and, as a result, we propose

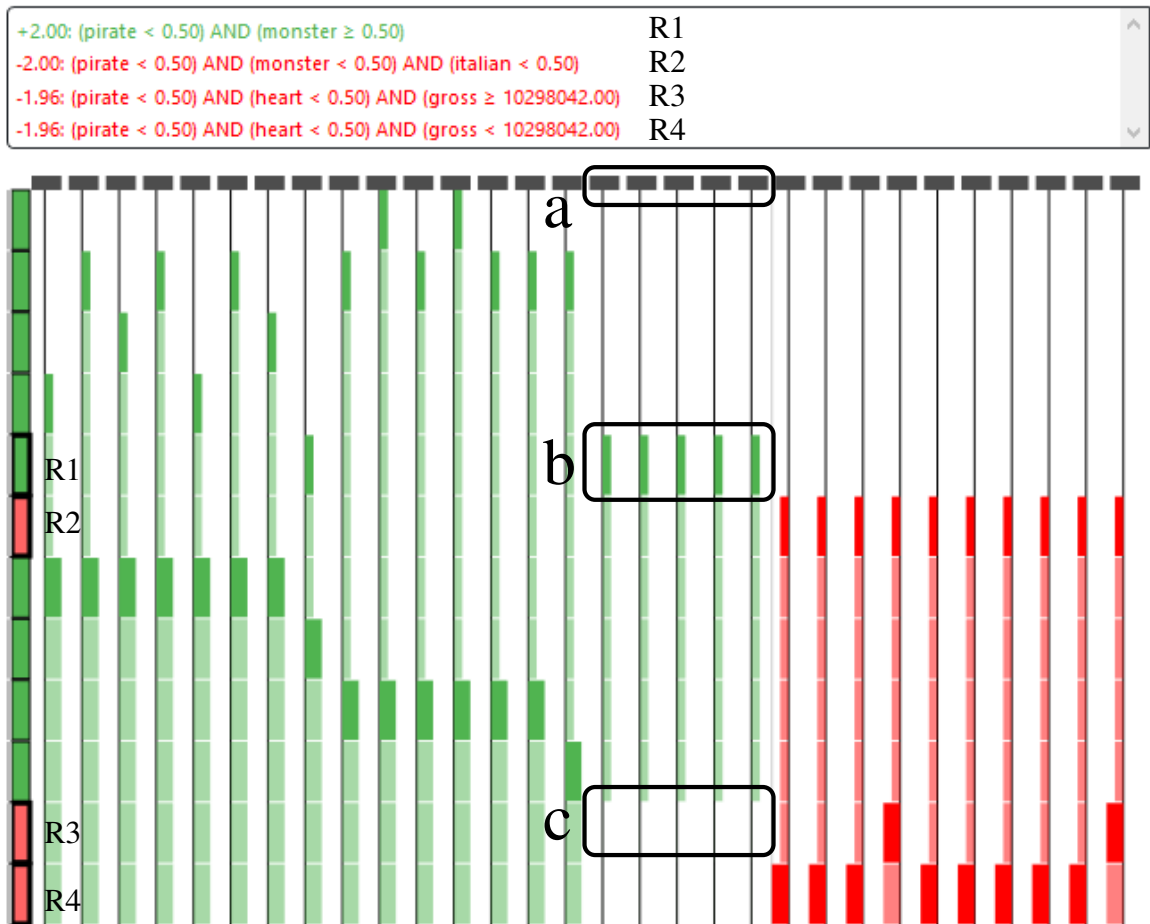


Figure 3.12: Cumulative Rule Gain view for the recommendations, displaying all rules that are applied to score the documents on the **first** iteration. On the top, the selected rules (R1, R2, R3 and R4) are displayed in detail.

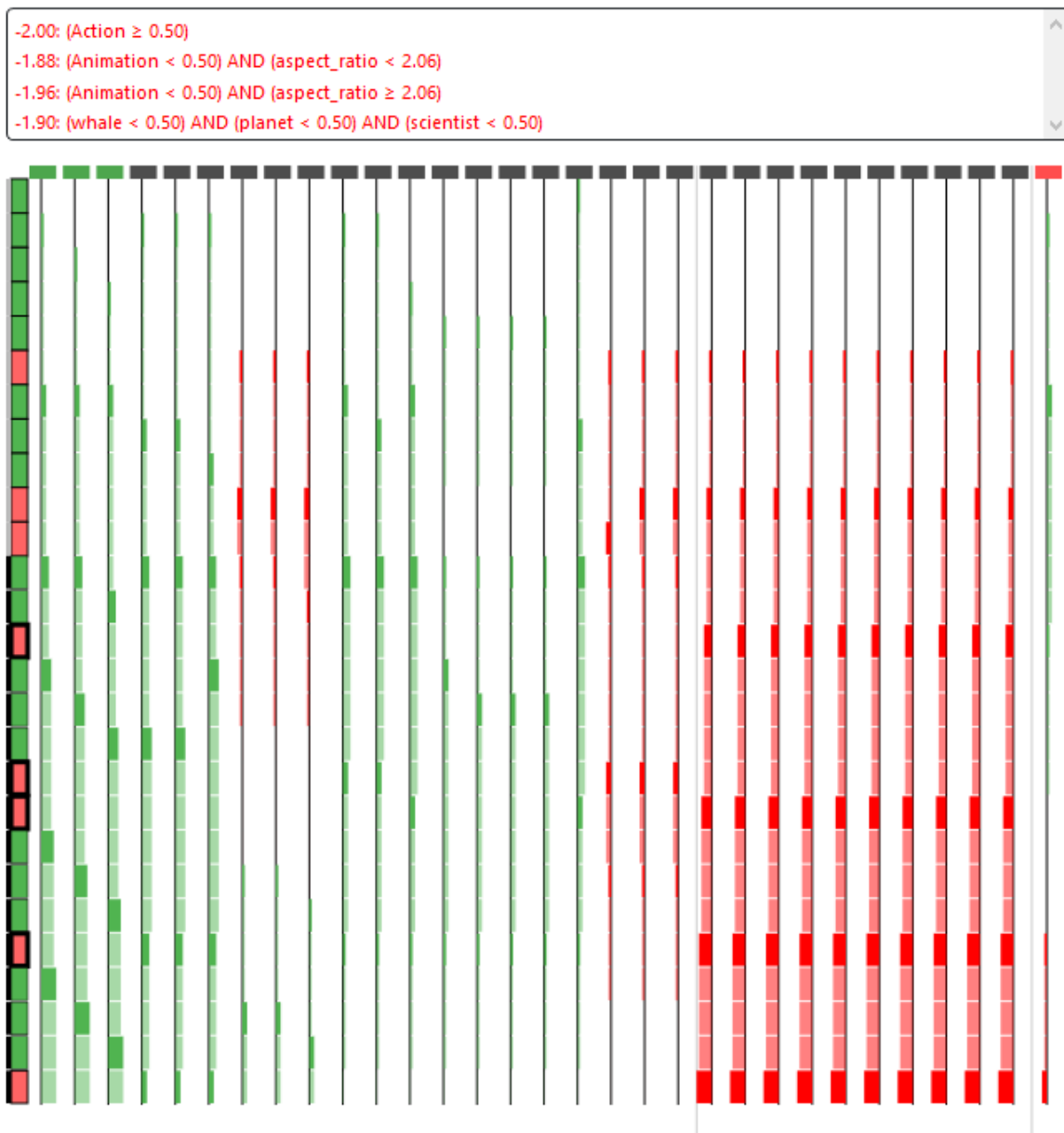


Figure 3.13: Cumulative Rule Gain view for the recommendations, displaying the **second** iteration. The rules related to the current iteration have a black indication on the left side. Rules with black borders are selected and shown in detail on the top.

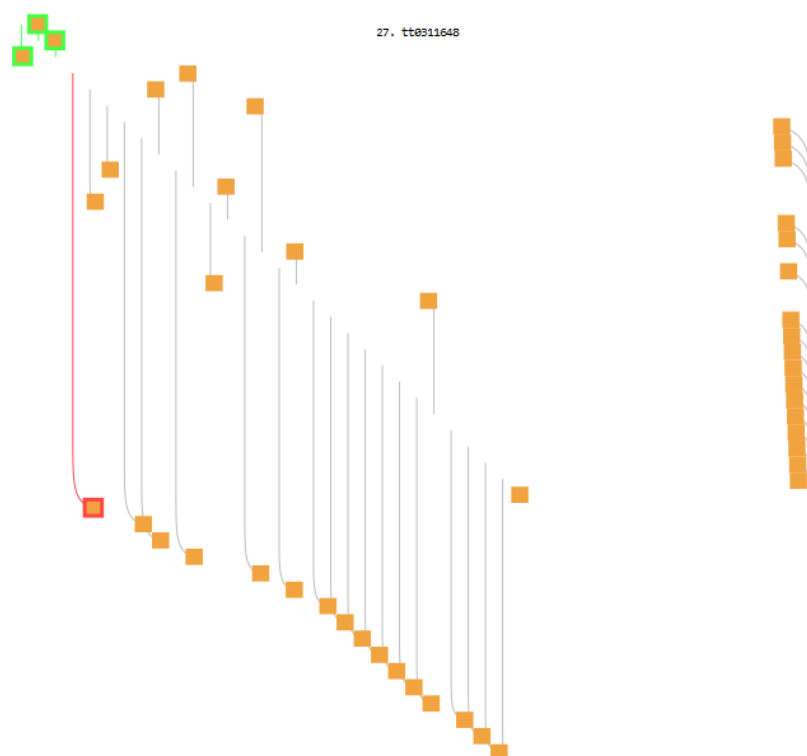


Figure 3.14: RankDiff in the middle of the transition to results of the **second** iteration of the "Understanding Ranking Order" Case Study, showing how new documents enter the view.

a new metric to evaluate the learning aspect of the algorithm. The CFG visualization allows easy identification of important features and how they influence the ranking of the results. And the CRG summarizes the rule contribution over documents, allowing analysis of the score change of the documents as rules are applied.

By combining the CFG and the CRG visualizations, it is possible to verify the most critical rules necessary to rank the results. Although the presented visualizations are utilized in an active learning setting, the same concept can be applied to offline models to explain the results of individual queries. As future work, it would be interesting to perform a user test, that could suggest possible improvements to the proposed methods. Also, we believe these visualizations could also be applied to other tasks, such as classification, with small adaptations.

Chapter 4

Discussion

In this chapter, I complement this work with a discussion of earlier experiments performed. Including some justifications on decisions, which led to the selected models, features and design process of the visualizations considered in the previous chapters. I also discuss some limitations of this work and the final conclusions.

4.1 Learning to Rank Models & Features

Many LtR algorithms could have been used in this work, but it is impossible to evaluate them all. Neural Network models were not considered because they usually require many training instances to perform well. Thus they did not look like the right approach for our problem. Support Vector Machines (SVM) had performed well in previous studies and have been adapted to ranking tasks before [21]. Therefore, it was considered and will be referred to as Rank SVM. LambdaMART is another algorithm designed for LtR tasks and used in recent studies [24, 10, 63], so it was also considered in this work. Besides the models, there are numerous ways to generate query-document representations, and some of them were also explored in this work.

Although the LETOR 4.0 dataset already contains features to be used for learning, the SemEval 2016 and 2017 datasets are based on text. Thus I was required to generate features to be able to use this dataset for evaluation. The SemEval dataset was also selected for evaluation because it is a recent dataset and of manageable size in which I could learn more about feature generation for text documents while also applying it to evaluate the interactive learning to rank from explicit feedback method.

To generate features, it is necessary to preprocess the original content and use some method to generate the query-document vector representations. The preprocessing by itself is a very complex task. The data contained misspellings, emoticons (faces and expressions based on special characters), *urls* and some garbage text with many repeated characters. After getting some cleaned data I started testing with simple approaches of feature vectors,

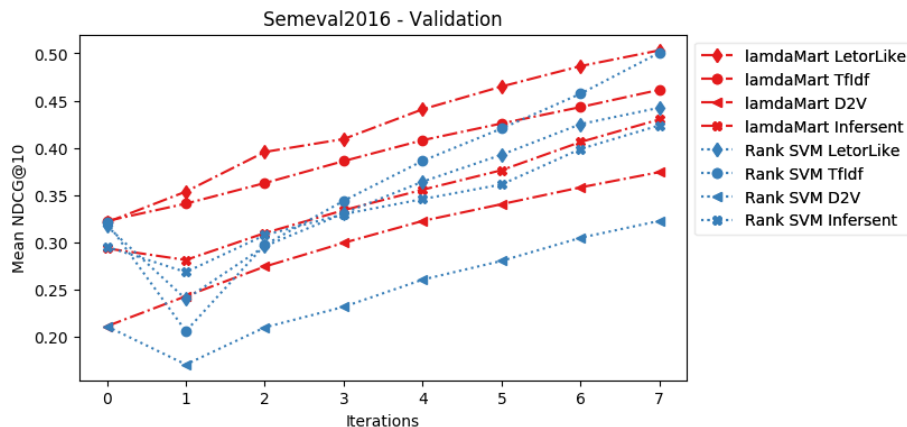


Figure 4.1: Comparison of LambdaMART and Rank SVM performance on NDCG@10 for different feature sets.

such as BOWs and TFIDF vectors. Later I also tried more complex features based on word embeddings such as Doc2Vec [33] and Infsent [9].

The feature set that showed to perform best and more consistently was used as the final feature set for evaluations. Such feature set was composed of similar features considered in Letor 4.0, which is referred to as *LetorLike* in this section. The size of the feature sets tested are displayed in Table 4.1 along with execution time comparison. Figure 4.1 compares the ranking performance using the NDCG@10 metric of both models in different feature sets. All these results are based on the validation set for SemEval2016. Any training required by the feature models was performed using the training set.

All types of features worked to some extent for both LambdaMART and Rank SVM models. However, they also came with a cost. Since the generation of query-document features depends on a dynamic element, the query, the generation of features must be considered on-demand on a real system. Nevertheless, my main concern was in studying the iterative learning behavior and improvements to the ranking. So for faster iteration, the features were generated once before execution. Although little effort was put into time optimizations, the times in Table 4.1 shows that both models achieved acceptable execution times, disregarding feature generation, overall to be used for real-time purposes.

Regarding ranking performance, Rank SVM managed to improve over time. However, in the first few iterations, which are most important, the model performed very poorly and worse than LambdaMART. The usage of word embedding features seems to make it harder

Table 4.1: Size of feature sets and average execution time for one iteration for different feature sets for LambdaMART and Rank SVM, averaged after 7 iterations of feedback.

	FeatSize	LambdaMART	Rank SVM
		Avg. exec. time (ms)	Avg. exec. time (ms)
LetorLike	36	6.05	4.44
TFIDF	1184	9.57	7.83
D2V	100	6.83	5.06
Infersent	4096	20.76	11.45

for both models to learn because they were too noisy to be learned well with just a few feedback instances. The results were better when using sparser features or using small feature sets, as can be seen in Figure 4.1. Another attempt was generating a smaller embedding by fine-tuning the Infersent features. However, it also did not achieve performance as good as using more straightforward features or the original Infersent embedding. Considering these results and that LambdaMART provided a direct way of being used incrementally, due to its boosted nature, and that it is considered an effective LTR model [63], LambdaMART was preferred.

4.2 Visualizations Design Process

Four other visualizations were designed and analyzed before the ones proposed in Chapter 3. Initially, they were considered as being part of a big dashboard, using the RankDiff visualization as the primary guide, as displayed in Figure 4.2. The idea was to have different views, showing various aspects of the main ranker. For that, the views were aligned to either the rows or the columns of the RankDiff visualization, so that surrounding views would be displaying diverse information regarding those documents.

Two of these four visualizations are model agnostic and consider only input features and model output (ranked order). One of the visualizations is a correlation matrix (Fig. 4.2 **a**) that displays the similarity of pairs of elements considering the cosine-similarity between their feature vectors. Although this visualization allows seeing some related groups of documents, it did not provide information regarding which features were related and affected their ranking.

Considering this gap on that first strategy, a new visualization was developed, to display individual features (Figure 4.2 **b**). This visualization is simple and displays the feature

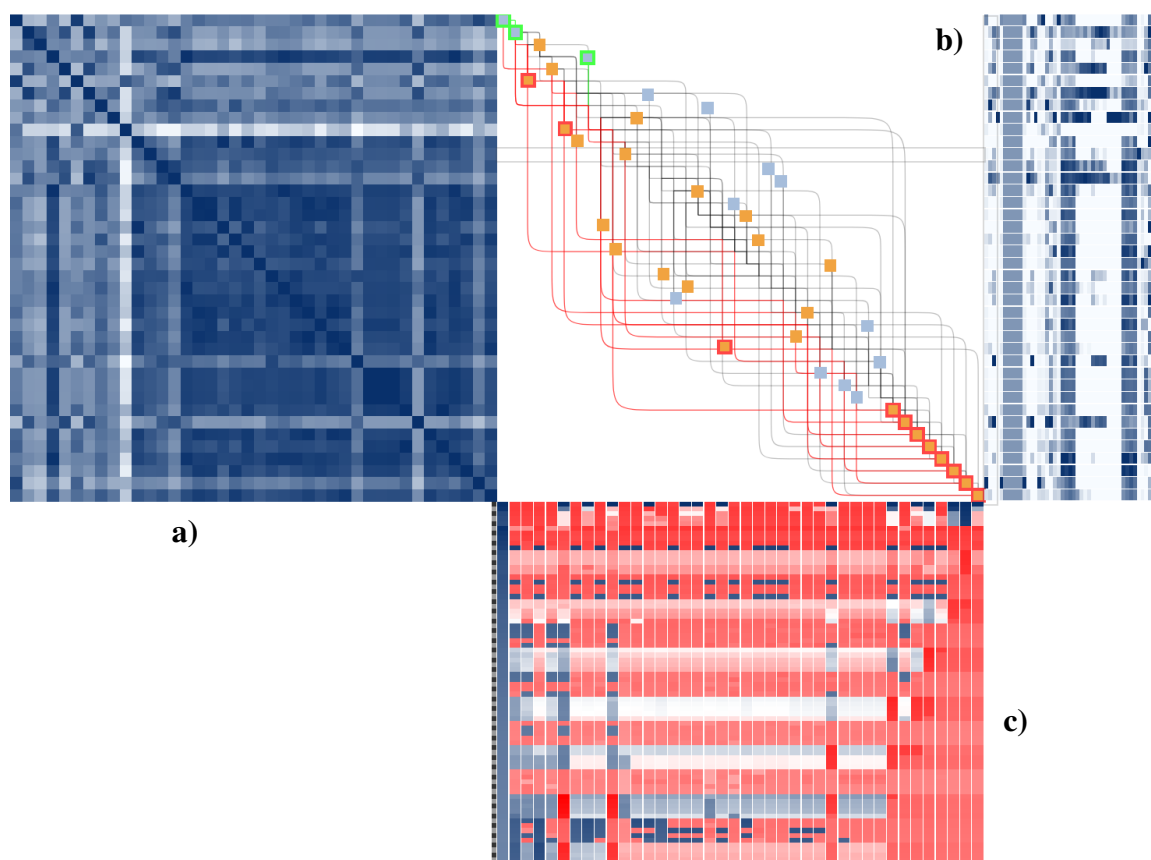


Figure 4.2: Initial visualizations developed for the dashboard with RankDiff as the central visualization, along with other surrounding visualizations. a) Correlation Matrix displaying pairwise cosine similarity between elements. b) Feature Heat-Map displaying element features. c) Leaf Scores visualization, displaying active leaf scores for every element.

vectors as small heat-maps, in which similar documents were evident when close together, same as what happened in the correlation matrix. Although it was possible to see similar feature patterns in the heat-map, the actual feature description still depended on special interaction, such as hovering. Also, if big feature vectors are used, the view gets too complex for a general overview, and there is no connection between the features and their actual influence in the ranking.

The other two visualizations tried to grasp more specific aspects of the tree-based model used in the approach presented in Chapter 2. Considering that the overall ranking score of the documents is based on the leaf values, a visualization that displays those values was developed (Figure 4.2 c). This visualization generated compelling patterns and allowed the overall view of how many positive and negative rules each document was falling into.

However, it was not related to any feature, so it was difficult to link those rules with relevant or irrelevant features.

Trying to create a view to show features used, a hierarchical visualization that displays all the levels for every tree in the forest was developed (Figure 4.3). The initial implementation used straight lines to represent the feature connections, but it was very noisy, with many lines overlapping. Then an edge-bundling[23] technique was applied to group edges and emphasize groups of similar connections. Also, the visualization considered connecting the branching paths over feature nodes at a relative position to the threshold value for the feature space. For example, a connection precisely in the middle of a feature node represents that the threshold is the middle value considering the value range for that feature for all elements. This visualization allowed an overview of used features and inter-feature dependency. By highlighting active nodes for a document, it was possible to see which features were considered when ranking the document. However, it was difficult to compare multiple documents at once because of the overlap.

All of these four visualizations had some positive sides and negative sides, but they did not cover well the information of interest required to understand the model considered in Chapter 2. Eventually, new ideas led to the concepts presented in Chapter 3, which condenses the information, allowing easy understanding of rank differences, feature importance, and feature influence on the ranking of individual documents.

4.3 Limitations and Future Work

Many algorithms could be used in Learning to Rank, but this work focused mostly on LambdaMART. Other models and/or features could achieve better results, but more study is required to determine that. Three visualizations for interpreting the LtR algorithm were analyzed through Case Studies. However, a formal user study could reveal possible improvements to the proposed visualizations that could lead to better interpretability of the ranking and the model. While the visualizations were proposed for use on ranking interpretability, they may be adaptable for other purposes, especially the CFG and CRG visualizations. For instance, CFG could be adapted to show the most probable class that the instance tends to be in regards to the individual features. While CRG could be used to display contrasting rules over different classes by using the bar color to represent the probably predicted class over the rules.

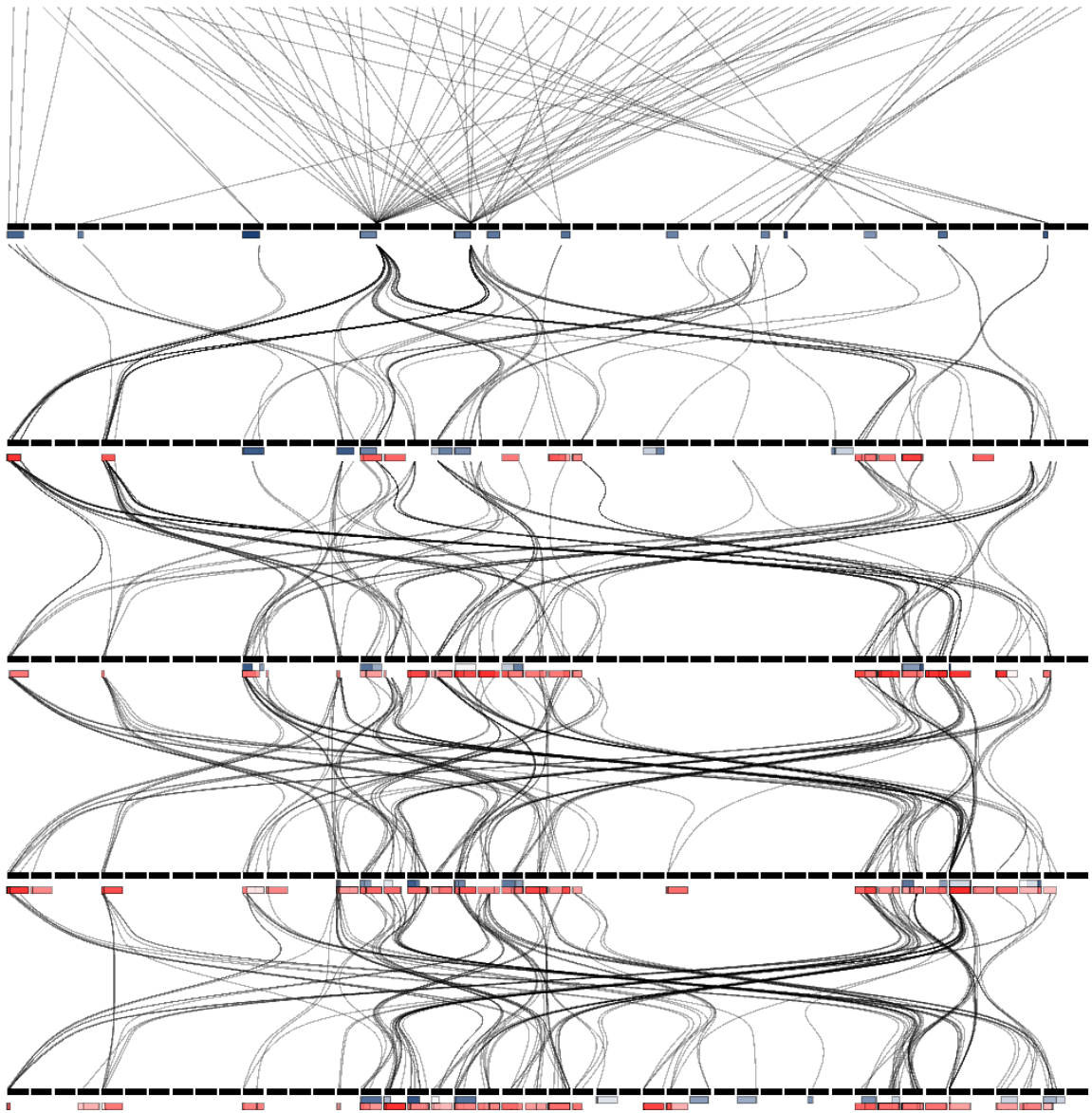


Figure 4.3: Hierarchical Forest visualization, which was initially considered for the dashboard, here showing five levels of the regression forest. Each black rectangle represents a feature. Topmost connections represent a tree, connecting to its first feature for decision. Connections to the feature nodes are positioned relative to their decision threshold. Each line represents a branch of a tree. The red and blue rectangles represent the leaves, which are a shade from red to blue representing a negative to positive contribution.

4.4 Conclusion

This work shows that LambdaMART can be used in iterative learning to rank systems that learn from explicit user feedback. I also explore diverse formats of the training methodology to find settings that would better utilize explicit user feedback to achieve higher results in fewer iterations. The evaluations show that the system can be a valuable option when no previous training data is available, achieving comparable or higher performance than offline methods in just a few iterations.

The proposed visualizations summarize diverse aspects of the ranking model and results, allowing a better understanding of how the ranking list was generated. RankDiff allows easy comparison between two ranked lists, and it was already demonstrated useful, by exposing a stagnation issue in the iterative learning to rank system. This issue was not obvious by only looking at standard metrics. After more analysis of this issue, a solution was considered and tested, resulting in improvements to the performance of the method. The Cumulative Feature Gain visualization provides a condensed view of how features influence the ranking of the results. In contrast, the Cumulative Rule Gain complements the previous visualization by providing a high-level overview of the rule's influence on elements. These visualizations allow comparison of multiple elements and reveal the rule's contribution and contrasting rules applied to the documents. I show, through a detailed Case Study, that these visualizations are effective in understanding why elements differ in ranking and what matters in their ranking.

Appendix A

Metrics used in Evaluations

There are many standard metrics from Information Retrieval that can be applied on Learning to Rank, such as Precision@K, Recall@K, MAP@K, NDCG@K and others. Here are described the main metrics that were used for evaluations performed in this work.

A.1 Recall

Recall is a metric to evaluate amount of relevant information found considering the amount of relevant information that could be found. For LtR, it is common to consider a Recall@K, which consider elements retrieved until position K at the ranking. As defined in Equation A.1, where r represent the number of relevant documents retrieved up to position K and R the total amount of relevant documents that exist, up to K . The final recall value range is $[0, 1]$, where 1 would indicate a total recall, where all relevant documents were retrieved up to K .

$$Recall@K = \frac{r}{R} \quad (A.1)$$

A.2 Mean Average Precision

To calculate The Mean Average Precision (MAP) it is necessary to calculate many precision values at different levels. The Precision@K is calculated by the fraction of found relevant items r divided by the amount of documents found K in the list, as in Equation A.2.

$$Precision@K = \frac{r}{K} \quad (A.2)$$

Then, Equation A.3 shows how the Average Precision can be calculated by averaging all Precision@x, with X going from 1 to K.

$$AveragePrecision@K = \sum_{x=1}^K Precision@x \quad (A.3)$$

The average precision is calculated for every test query and then averaged by all queries, which then gives the final MAP value. The MAP metric ranges from 0 to 1, where 1 represents the best score possible.

A.3 Normalized Discounted Cumulative Gain

The Normalized Discounted Cumulative Gain (NDCG) [26], is a ranking evaluating metric with the purpose of providing a good metric to access the quality of a sorted list of results with different levels of importance. For instance, relevant documents should be considered more valuable than irrelevant documents.

Note that the NDCG differs from the Recall@K and MAP@K in the sense that the last two only considers a binary relevance level, since the precision and recall are calculated based on an amount of relevant documents, which must be defined as a threshold in case a dataset with more relevance levels is used. The NDCG calculates a score based on how far the relevant documents are from the top of the list and is usually considered up to the top K documents. This comes from the intuition the documents farther from the top are less likely to be seen.

To define NDCG it is necessary to first define the Cumulated Gain (CG) and the Discounted CG. Considering a list G of discrete relevance scores that relate to the list of documents to be evaluated, CG is a the vector of accumulated relevance scores until position i of all entries i in G , represented by G_i . CG can be recursively defined as in Equation A.4.

$$\begin{cases} CG_i = G_i & \text{If } i = 0 \\ CG_i = G_i + G_{i-1} & \text{If } i > 0 \end{cases} \quad (A.4)$$

The DCG_i is then defined considering the relevance scores weighted by a discount factor, based on the item's position in the list, as in Equation A.5. The discounting factor is calculated using $\log_b i$, where b is usually set to 2, and higher values would increase the importance of documents farther away from the beginning of the list.

$$\begin{cases} DCG_i = G_i & \text{If } i < b \\ DCG_i = DCG_i + \frac{G_i}{\log_b i} & \text{If } i \geq 0 \end{cases} \quad (\text{A.5})$$

Then, to calculate the NDCG, it is necessary to get an ideal list I , that contains the relevance of the same documents from G , but in descending order: from most relevant to least relevant. The NDCG@K can be then calculated by the fraction of the DCG for the list relevance scores G and I , as seen in Equation A.6:

$$NDCG@K = \frac{DCG_k}{IDCG_k} \quad (\text{A.6})$$

The NDCG should yield a value in the $[0, 1]$, range, where 1 indicates the best ranking order.

Appendix B

ACM Copyright Letter

The following pages include a copy of the ACM Copyright letter related to the paper “Iterative Learning to Rank from Explicit Relevance Feedback”.

ACM Copyright and Audio/Video Release

Title of the Work: Iterative Learning to Rank from Explicit Relevance Feedback

Author/Presenter(s): Mateus Malvessi Pereira:Dalhousie University;Elham Etemad:Dalhousie University;Fernando Paulovich:Dalhousie University

Type of material: Full Paper

Publication and/or Conference Name: The 35th ACM/SIGAPP Symposium on Applied Computing Proceedings

I. Copyright Transfer, Reserved Rights and Permitted Uses

* Your Copyright Transfer is conditional upon you agreeing to the terms set out below.

Copyright to the Work and to any supplemental files integral to the Work which are submitted with it for review and publication such as an extended proof, a PowerPoint outline, or appendices that may exceed a printed page limit, (including without limitation, the right to publish the Work in whole or in part in any and all forms of media, now or hereafter known) is hereby transferred to the ACM (for Government work, to the extent transferable) effective as of the date of this agreement, on the understanding that the Work has been accepted for publication by ACM.

Reserved Rights and Permitted Uses

(a) All rights and permissions the author has not granted to ACM are reserved to the Owner, including all other proprietary rights such as patent or trademark rights.

(b) Furthermore, notwithstanding the exclusive rights the Owner has granted to ACM, Owner shall have the right to do the following:

(i) Reuse any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.

(ii) Create a "[Major Revision](#)" which is wholly owned by the author

(iii) Post the Accepted Version of the Work on (1) the Author's home page, (2) the Owner's institutional repository, (3) any repository legally mandated by an agency funding the research on which the Work is based, and (4) any non-commercial repository or aggregation that does not duplicate ACM tables of contents, i.e., whose patterns of links do not substantially duplicate an ACM-copyrighted volume or issue. Non-commercial repositories are here understood as repositories owned by non-profit organizations that do not charge a fee for accessing deposited articles and that do not sell advertising or otherwise profit from serving articles.

(iv) Post an "[Author-Izer](#)" link enabling free downloads of the Version of Record in the ACM Digital Library on (1) the Author's home page or (2) the Owner's institutional repository;

(v) Prior to commencement of the ACM peer review process, post the version of the Work as submitted to ACM ("[Submitted Version](#)" or any earlier versions) to non-peer reviewed servers;

(vi) Make free distributions of the final published Version of Record internally to the Owner's employees, if applicable;

(vii) Make free distributions of the published Version of Record for Classroom and Personal Use;

(viii) Bundle the Work in any of Owner's software distributions; and

(ix) Use any Auxiliary Material independent from the Work. (x) If your paper is withdrawn before it is published in the ACM Digital Library, the rights revert back to the author(s).

When preparing your paper for submission using the ACM TeX templates, the rights and permissions information and the bibliographic strip must appear on the lower left hand portion of the first page.

The new [ACM Consolidated TeX template Version 1.3 and above](#) automatically creates and positions these text blocks for you based on the code snippet which is system-generated based on your rights management choice and this particular conference.

NOTE: For authors using the ACM Microsoft Word Master Article Template and Publication Workflow, The ACM Publishing System (TAPS) will add the rights statement to your papers for you. Please check with your conference contact for information regarding submitting your source file(s) for processing.

Please put the following LaTeX commands in the preamble of your document - i.e., before `\begin{document}`:

```
\copyrightyear{2020}
\acmYear{2020}
\setcopyright{acmcopyright}
\acmConference[SAC '20]{The 35th ACM/SIGAPP Symposium on Applied Computing}{March
30-April 3, 2020}{Brno, Czech Republic}
\acmBooktitle{The 35th ACM/SIGAPP Symposium on Applied Computing (SAC '20), March
30-April 3, 2020, Brno, Czech Republic}
\acmPrice{15.00}
\acmDOI{10.1145/3341105.3374002}
\acmISBN{978-1-4503-6866-7/20/03}
```

NOTE: For authors using the ACM Microsoft Word Master Article Template and Publication Workflow, The ACM Publishing System (TAPS) will add the rights statement to your papers for you. Please check with your conference contact for information regarding submitting your source file(s) for processing.

If you are using the ACM Interim Microsoft Word template, or still using or older versions of the ACM SIGCHI template, you must copy and paste the following text block into your document as per the instructions provided with the templates you are using:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

SAC '20, March 30-April 3, 2020, Brno, Czech Republic
 © 2020 Association for Computing Machinery.
 ACM ISBN 978-1-4503-6866-7/20/03...\$15.00
<https://doi.org/10.1145/3341105.3374002>

NOTE: Make sure to include your article's DOI as part of the bibstrip data; DOIs will be registered and become active shortly after publication in the ACM Digital Library. Once you have your camera ready copy ready, please send your source files and PDF to your event contact for processing.

A. Assent to Assignment. I hereby represent and warrant that I am the sole owner (or authorized agent of the copyright owner(s)), with the exception of third party materials detailed in section III below. I have obtained permission for any third-party material included in the Work.

B. Declaration for Government Work. I am an employee of the National Government of my country and my Government claims rights to this work, or it is not copyrightable (Government work is classified as Public Domain in U.S. only)

Are any of the co-authors, employees or contractors of a National Government? Yes No

II. Permission For Conference Recording and Distribution

* Your Audio/Video Release is conditional upon you agreeing to the terms set out below.

I hereby grant permission for ACM to include my name, likeness, presentation and comments in any and all forms, for the Conference and/or Publication.

I further grant permission for ACM to record and/or transcribe and reproduce my presentation as part of the ACM Digital Library, and to distribute the same for sale in complete or partial form as part of an ACM product on CD-ROM, DVD, webcast, USB device, streaming video or any other media format now or hereafter known.

I understand that my presentation will not be sold separately as a stand-alone product without my direct consent. Accordingly, I give ACM the right to use my image, voice, pronouncements, likeness, and my name, and any biographical material submitted by me, in connection with the Conference and/or Publication, whether used in excerpts or in full, for distribution described above and for any associated advertising or exhibition.

Do you agree to the above Audio/Video Release? Yes No

III. Auxiliary Material

Do you have any Auxiliary Materials? Yes No

IV. Third Party Materials

In the event that any materials used in my presentation or Auxiliary Materials contain the work of third-party individuals or organizations (including copyrighted music or movie excerpts or anything not owned by me), I understand that it is my responsibility to secure any necessary permissions and/or licenses for print and/or digital publication, and cite or attach them below.

- We/I have not used third-party material.
 We/I have used third-party materials and have necessary permissions.

V. Artistic Images

If your paper includes images that were created for any purpose other than this paper and to which you or your employer claim copyright, you must complete Part V and be sure to include a notice of copyright with each such image in the paper.

- We/I do not have any artistic images.
 We/I have any artistic images.

VI. Representations, Warranties and Covenants

The undersigned hereby represents, warrants and covenants as follows:

- (a) Owner is the sole owner or authorized agent of Owner(s) of the Work;
- (b) The undersigned is authorized to enter into this Agreement and grant the rights included in this license to ACM;
- (c) The Work is original and does not infringe the rights of any third party; all permissions for use of third-party materials consistent in scope and duration with the rights granted to ACM have been obtained, copies of such permissions have been provided to ACM, and the Work as submitted to ACM clearly and accurately indicates the credit to the proprietors of any such third-party materials (including any applicable copyright notice), or will be revised to indicate such credit;
- (d) The Work has not been published except for informal postings on non-peer reviewed servers, and Owner covenants to use best efforts to place ACM DOI pointers on any such prior postings;
- (e) The Auxiliary Materials, if any, contain no malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software; and
- (f) The Artistic Images, if any, are clearly and accurately noted as such (including any applicable copyright notice) in the Submitted Version.

I agree to the Representations, Warranties and Covenants

Funding Agents

1. Mitacs award number(s): IT12193, IT12997

DATE: **12/06/2019** sent to mpereira@dal.ca at **16:12:34**

Bibliography

- [1] Leila Arras, Franziska Horn, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. "What is Relevant in a Text Document?": An Interpretable Machine Learning Approach. *PLoS ONE*, 12(8), dec 2016.
- [2] Kumaripaba Athukorala, Alan Medlar, Antti Oulasvirta, Giulio Jacucci, and Dorota Glowacka. Beyond Relevance: Adapting Exploration/Exploitation in Information Retrieval. In *Proceedings of the 21st International Conference on Intelligent User Interfaces*, IUI '16, pages 359–369, New York, NY, USA, 2016. ACM.
- [3] Sebastian Bach, Alexander Binder, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. Analyzing Classifiers: Fisher Vectors and Deep Neural Networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-Decem:2912–2920, dec 2015.
- [4] Alberto Barrón-Cedeno, Simone Filice, Giovanni Da San Martino, Shafiq Joty, Lluís Màrquez, Preslav Nakov, and Alessandro Moschitti. Thread-level information for comment classification in community question answering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 687–693, 2015.
- [5] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, dec 2011.
- [6] Christopher J.C. Burges, Robert Ragno, Quoc Viet Le, and Quoc V Le Christopher J. Burges Robert Ragno. Learning to rank with nonsmooth cost functions. In B Schölkopf, J C Platt, and T Hoffman, editors, *Advances in Neural Information Processing Systems*, pages 193–200. MIT Press, 2006.
- [7] Rodrigo Tripodi Calumby, Marcos André Gonçalves, and Ricardo da Silva Torres. On interactive learning-to-rank for IR: Overview, recent advances, challenges, and directions. *Neurocomputing*, 208:3–24, oct 2016.
- [8] Zhe Cao, Tao Qin, Tie-Yan Yan Liu, Ming-Feng Feng Tsai, and Hang Li. Learning to rank: From pairwise approach to listwise approach. *ACM International Conference Proceeding Series*, 227:129–136, 2007.
- [9] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Loïc Loic Barrault, and Antoine Bordes. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. pages 670–680, may 2017.

- [10] Faïza Dammak, Imen Gabsi, Hager Kammoun, and Abdelmajid Ben Hamadou. Active Learning to Rank Method for Documents Retrieval. *The 10th international conference on Internet and Web Applications and Services*, pages 16–21, 2015.
- [11] Cecilia Di Sciascio, Vedran Sabol, and Eduardo E. Veas. Rank as you go: User-driven exploration of search results. In *International Conference on Intelligent User Interfaces, Proceedings IUI*, volume 07-10-Marc, pages 118–129, New York, New York, USA, mar 2016. Association for Computing Machinery.
- [12] Pinar Donmez and Jaime G Carbonell. Active sampling for rank learning via optimizing the area under the ROC curve. In *European Conference on Information Retrieval*, pages 78–89, 2009.
- [13] Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning. *Communications of the ACM*, 63(1):68–77, jul 2020.
- [14] Leilani H. Gilpin, David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *Proceedings - 2018 IEEE 5th International Conference on Data Science and Advanced Analytics, DSAA 2018*, pages 80–89. Institute of Electrical and Electronics Engineers Inc., jan 2019.
- [15] Liang Gou, Fang You, Jun Guo, Luqi Wu, and Xiaolong Zhang. SFViz: Interest-based friends exploration and recommendation in social networks. In *ACM International Conference Proceeding Series*, pages 1–10, New York, New York, USA, 2011. ACM Press.
- [16] Samuel Gratzl, Alexander Lex, Nils Gehlenborg, Hanspeter Pfister, and Marc Streit. LineUp : Visual Analysis of Multi-Attribute Rankings. *IEEE Transactions on Visualization and Computer Graphics*, 19(March):2277–2286, 2013.
- [17] Brynjar Gretarsson, John O’Donovan, Svetlin Bostandjiev, Christopher Hall, and Tobias Höllerer. SmallWorlds: Visualizing social recommendations. *Computer Graphics Forum*, 29(3):833–842, 2010.
- [18] Fan Guo, Lei Li, and Christos Faloutsos. Tailoring click models to user goals. pages 88–92, feb 2009.
- [19] Fan Guo, Chao Liu, and Yi Min Wang. Efficient Multiple-click Models in Web Search. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining, WSDM ’09*, pages 124–131, New York, NY, USA, 2009. ACM.
- [20] Chen He, Denis Parra, and Katrien Verbert. Interactive recommender systems: A survey of the state of the art and future research challenges and opportunities. *Expert Systems with Applications*, 56:9–27, sep 2016.

- [21] Ralf Herbrich, Thore Graepel, and Klaus Obermayer. Large Margin Rank Boundaries for Ordinal Regression. *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, 2000.
- [22] Katja Hofmann, Shimon Whiteson, and Maarten de Rijke. Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval. *Information Retrieval*, 16(1):63–90, feb 2013.
- [23] Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. In *IEEE Transactions on Visualization and Computer Graphics*, volume 12, pages 741–748, sep 2006.
- [24] Ziniu Hu, Qu Peng, Yang Wang, and Hang Li. Unbiased LambdaMart: An unbiased pairwise learning-to-rank algorithm. In *The Web Conference 2019 - Proceedings of the World Wide Web Conference, WWW 2019*, pages 2830–2836, New York, New York, USA, may 2019. Association for Computing Machinery, Inc.
- [25] Muhammad Ibrahim and Mark Carman. Comparing pointwise and listwise objective functions for random-forest-based learning-to-rank. *ACM Transactions on Information Systems (TOIS)*, 34(4):20, 2016.
- [26] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, oct 2002.
- [27] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately Interpreting Clickthrough Data As Implicit Feedback. *SIGIR Forum*, 51(1):4–11, aug 2017.
- [28] Michael Jugovac and Dietmar Jannach. Interacting with recommenders-overview and research directions, sep 2017.
- [29] Antti Kangasrääsiö, Yi Chen, Dorota Głowacka, and Samuel Kaski. Interactive modeling of concept drift and errors in relevance feedback. In *Proceedings of the 2016 conference on user modeling adaptation and personalization*, pages 185–193, 2016.
- [30] Maryam Karimzadehgan, Wei Li, Ruofei Zhang, and Jianchang Mao. A stochastic learning-to-rank algorithm and its application to contextual advertising. In *Proceedings of the 20th International Conference on World Wide Web, WWW 2011*, pages 377–386, New York, New York, USA, 2011. ACM Press.
- [31] Khalil Klouche, Tuukka Ruotsalo, Luana Micallef, Salvatore Andolina, and Giulio Jacucci. Visual re-ranking for multi-aspect information retrieval. In *CHIIR 2017 - Proceedings of the 2017 Conference Human Information Interaction and Retrieval*, pages 57–66, New York, New York, USA, mar 2017. Association for Computing Machinery, Inc.
- [32] Josua Krause, Adam Perer, and Enrico Bertini. Using Visual Analytics to Interpret Predictive Machine Learning Models. jun 2016.

- [33] Quoc V. Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. *31st International Conference on Machine Learning, ICML 2014*, 4:2931–2939, may 2014.
- [34] Ping Li, Qiang Wu, and Christopher J Burges. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in neural information processing systems*, pages 897–904, 2008.
- [35] Zachary C. Lipton. The Mythos of Model Interpretability. *Communications of the ACM*, 61(10):35–43, jun 2016.
- [36] Tie Yan Liu. Learning to rank for Information Retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–231, 2009.
- [37] Bo Long, Jiang Bian, Olivier Chapelle, Ya Zhang, Yoshiyuki Inagaki, and Yi Chang. Active learning for ranking through expected loss optimization. *IEEE Transactions on Knowledge and Data Engineering*, 27(5):1180–1191, 2015.
- [38] Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, and Salvatore Trani. 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*, pages 1077–1080, 2017.
- [39] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. Tf-idf weighting. In *An Introduction to Information Retrieval*, chapter 6, pages 118–120. Cambridge University Press.
- [40] Tomas Mikolov, Wen-Tau Yih, and Geoffrey Zweig. Linguistic Regularities in Continuous Space Word Representations. Technical report, may 2013.
- [41] Yao Ming, Huamin Qu, and Enrico Bertini. RuleMatrix: Visualizing and Understanding Classifiers with Rules. jul 2018.
- [42] Pranathi Mylavarapu, Adil Yalçın, Xan Gregg, and Niklas Elmqvist. Ranked-list visualization: A graphical perception study. *Conference on Human Factors in Computing Systems - Proceedings*, pages 1–12, 2019.
- [43] Preslav Nakov, Doris Hoogeveen, Lluís Màrquez, Alessandro Moschitti, Hamdy Mubarak, Timothy Baldwin, and Karin Verspoor. SemEval-2017 task 3: Community question answering. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 27–48, 2017.
- [44] Preslav Nakov, Lluís Màrquez, Moschitti Alessandro, Walid Magdy, Hamdy Mubarak, Abed Alhakim Freihat, James Glass, and Bilal Randeree. Semeval-2016 task 3: Community question answering. *Proceedings of SemEval*, pages 525–545, 2016.

- [45] John O’Donovan, Barry Smyth, Brynjar Gretarsson, Svetlin Bostandjiev, and Tobias Höllerer. PeerChooser: Visual interactive recommendation. In *Conference on Human Factors in Computing Systems - Proceedings*, pages 1085–1088, New York, New York, USA, 2008. ACM Press.
- [46] Jaakko Peltonen, Kseniia Belorustceva, and Tuukka Ruotsalo. Topic-relevance map: Visualization for improving search result comprehension. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces*, pages 611–622, 2017.
- [47] Mateus Pereira, Elham Etemad, and Fernando Paulovich. Iterative Learning to Rank from Explicit Relevance Feedback. In *Proceedings of the 35th ACM/SIGAPP Symposium on Applied Computing, SAC 20*, 2020.
- [48] Tao Qin and Tie-Yan Liu. Introducing LETOR 4.0 Datasets. *CoRR*, abs/1306.2, jun 2013.
- [49] Xipeng Qiu and Xuanjing Huang. Convolutional neural tensor network architecture for community-based question answering. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.
- [50] Filip Radlinski and Thorsten Joachims. Query chains: learning to rank from implicit feedback. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 239–248, 2005.
- [51] Filip Radlinski and Thorsten Joachims. Active exploration for learning rankings from clickthrough data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 570–579, 2007.
- [52] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ”Why should i trust you?” Explaining the predictions of any classifier. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 13-17-Aug, pages 1135–1144. Association for Computing Machinery, aug 2016.
- [53] S E Robertson, S Walker, S Jones, M M Hancock-Beaulieu, and M Gatford. Okapi at TRECC2. Technical report, jan 1994.
- [54] Tuukka Ruotsalo, Giulio Jacucci, Petri Myllymäki, and Samuel Kaski. Interactive intent modeling: information discovery beyond search. *Commun. ACM*, 58(1):86–92, 2015.
- [55] Tuukka Ruotsalo, Jaakko Peltonen, Manuel Eugster, Dorota Głowacka, Ksenia Konyushkova, Kumaripaba Athukorala, Ilkka Kosunen, Aki Reijonen, Petri Myllymäki, Giulio Jacucci, and Samuel Kaski. Directing exploratory search with interactive intent modeling. In *International Conference on Information and Knowledge Management, Proceedings*, pages 1759–1764, New York, New York, USA, 2013. ACM Press.

- [56] Gerard Salton and Michael J McGill. Introduction to modern information retrieval. 1986.
- [57] Catarina Silva and Bernardete Ribeiro. Visualization of individual ensemble classifier contributions. In *Communications in Computer and Information Science*, volume 611, pages 633–642. Springer Verlag, 2016.
- [58] Rodrigo Silva, Marcos A. Gonçalves, and Adriano Veloso. Rule-based active sampling for learning to rank. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6913 LNAI, pages 240–255. Springer, Berlin, Heidelberg, 2011.
- [59] Ivan Srba and Maria Bielikova. A comprehensive survey and classification of approaches for community question answering. *ACM Transactions on the Web (TWEB)*, 10(3):18, 2016.
- [60] Ivan Srba and Maria Bielikova. Why is stack overflow failing? preserving sustainability in community question answering. *IEEE Software*, 33(4):80–89, 2016.
- [61] Gregor Stiglic, Matej Mertik, Vili Podgorelec, and Peter Kokol. Using visual interpretation of small ensembles in microarray analysis. *Proceedings - IEEE Symposium on Computer-Based Medical Systems*, 2006:691–695, 2006.
- [62] Panagiotis Symeonidis, Alexandros Nanopoulos, and Yannis Manolopoulos. Movi-Explain: A recommender system with explanations. In *RecSys'09 - Proceedings of the 3rd ACM Conference on Recommender Systems*, pages 317–320, New York, New York, USA, 2009. ACM Press.
- [63] Niek Tax, Sander Bockting, and Djoerd Hiemstra. A cross-benchmark comparison of 87 learning to rank methods. *Information Processing and Management*, 51(6):757–772, aug 2015.
- [64] Quan Hung Tran, Vu Tran, Tu Vu, Minh Nguyen, and Son Bao Pham. JAIST: Combining multiple features for answer selection in community question answering. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 215–219, 2015.
- [65] Stef Van Den Elzen and Jarke J. Van Wijk. BaobabView: Interactive construction and analysis of decision trees. *VAST 2011 - IEEE Conference on Visual Analytics Science and Technology 2011, Proceedings*, pages 151–160, 2011.
- [66] Alfredo Vellido, José D Martín-Guerrero, and Paulo J G Lisboa. *Making machine learning models interpretable*. 2012.
- [67] Eric S. Vorm. Assessing Demand for Transparency in Intelligent Systems Using Machine Learning. In *2018 IEEE (SMC) International Conference on Innovations in Intelligent Systems and Applications, INISTA 2018*. Institute of Electrical and Electronics Engineers Inc., sep 2018.

- [68] Mintao Wu, Jihua Zhu, Jun Wang, Shanmin Pang, and Yaochen Li. Listwise approach based on the cross-correntropy for learning to rank. *Electronics Letters*, 54(14):878–880, 2018.
- [69] Qiang Wu, Christopher J.C. C Burges, Krysta M. Svore, and Jianfeng Gao. Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3):254–270, 2010.
- [70] Dong Xishuang, Chen Xiaodong, Guan Yi, Xu Zhiming, and Li Sheng. An overview of learning to rank for information retrieval. In *2009 WRI World Congress on Computer Science and Information Engineering, CSIE 2009*, volume 3, pages 600–606, 2009.
- [71] Hema Yoganarasimhan. Search personalization using machine learning. *Management Science*, 2019.
- [72] Hwanjo Yu, Taehoon Kim, Jino Oh, Ilhwan Ko, Sungchul Kim, and Wook-Shin Han. Enabling multi-level relevance feedback on PubMed by integrating rank learning into DBMS. In *BMC bioinformatics*, volume 11, page S6, 2010.
- [73] Yueming. IMDB 5000 Movie Dataset — Kaggle.
- [74] Jiawei Zhang, Yang Wang, Piero Molino, Lezhi Li, and David S. Ebert. Manifold: A Model-Agnostic Framework for Interpretation and Diagnosis of Machine Learning Models. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):364–373, aug 2019.
- [75] Guangyou Zhou, Tingting He, Jun Zhao, and Po Hu. Learning continuous word embedding with metadata for question retrieval in community question answering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 250–259, 2015.
- [76] Masrour Zoghi, Shimon Whiteson, and Maarten Rijke. MergeRUCB: A Method for Large-Scale Online Ranker Evaluation. *WSDM 2015 - Proceedings of the 8th ACM International Conference on Web Search and Data Mining*, pages 17–26, feb 2015.