DELFT UNIVERSITY OF TECHNOLOGY

Master Thesis

USER/QUERY-WISE METRIC BOUNDING IN LEARNING TO RANK

by

T.D. WESTERBORG

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science
in the
Multimedia Computing Group

Student number:	4386817
Supervisor:	dr. J. Urbano

May 11, 2022

Contents

1 Introduction **1**
- 1.1 Research goals . . . 3
- 1.2 Main findings . . . 3
- 1.3 Thesis outline . . . 3

2 Background **4**
- 2.1 Learning To Rank Framework . . . 4
- 2.2 Models . . . 5
- 2.3 Evaluation . . . 6
- 2.4 Listwise loss functions . . . 8

3 Metric Bounding **11**
- 3.1 Min-Max Normalization . . . 11
  - 3.1.1 Bounds of $nDCG$ . . . 12
  - 3.1.2 Bounds of $nRBP$ . . . 12
  - 3.1.3 Bounds of AP . . . 13
  - 3.1.4 Impact of bounding on loss scores . . . 13
- 3.2 Expectation-Based Metric Bounding . . . 14
  - 3.2.1 Expectation of $nDCG$ . . . 14
  - 3.2.2 Expectation of $nRBP$ . . . 16
  - 3.2.3 Expectation of AP . . . 17
  - 3.2.4 Impact of bounding on loss scores . . . 18
- 3.3 Expectation-Max Bounding . . . 19
  - 3.3.1 Impact of bounding on loss scores . . . 19
- 3.4 Distribution-Based Metric Bounding . . . 20
  - 3.4.1 Impact of bounding on loss values . . . 23

4 Experimental Setup **24**
- 4.1 Models . . . 24
  - 4.1.1 Hyperparameters . . . 24
- 4.2 Datasets . . . 25
- 4.3 Protocol . . . 25

5 Results **27**
- 5.1 Effect of metric bounding on overall performance . . . 27
- 5.2 Effect of metric bounding on individual user/query performance . . . 29
- 5.3 Where does metric bounding have the largest effect? . . . 30
- 5.4 Who benefits from metric bounding? . . . 34
- 5.5 Summary . . . 36

6 Conclusion **37**
6.1 Recommendations 38
6.2 Future Work. 38

References **39**

1 INTRODUCTION

Learning to Rank (LTR) is at the core of many retrieval problems such as document retrieval and recommendation and aims at learning how to properly rank a set of items based on some criterion like relevance to a query or user. As often times users tend to only browse the top of the predicted ranking, a good ranking is one where relevant items are placed near the top and irrelevant items near the bottom. A wide variety of machine learning (ML) techniques have been applied to create ranking models capable of ranking items according to their relevance. The main difference between traditional ML and LTR is that while traditional ML solves a prediction problem on an instance consisting of a single item, LTR defines instances as a set of items. Subsequently, LTR methods care less about individual predicted relevance, but aim at optimizing the relative ordering of the items.

In order to train such models, a training set consisting of users or queries, sets of items and ground truth relevance judgments is often used. A loss function (or objective function) is utilized to measure the accordance between the predicted and ground-truth ranking. Logically, the choice of loss function plays a big role in the optimization process, as it defines what is considered to be correct and what is not. While a wide variety of loss functions exist, a straightforward choice of loss would be one that closely resembles the measure used to evaluate the ranking function. We will call this sub-category of LTR techniques the direct optimization method and will mainly focus on this subcategory throughout the rest of this thesis. Information retrieval (IR) metrics such as $nDCG$, $nRBP$ and AP, which evaluate systems on a per-query basis, are often used to evaluate the performance of ranking functions. Therefore, direct optimization methods deploy loss functions which closely resemble these metrics.

During training, an optimization algorithm such as gradient descent or Adam is then used to adjust the models parameters in order to minimize the loss on the training set. The optimization is an iterative process which utilizes the loss function and its derivatives with respect to the models parameters to determine how to update the model. To increase the computational efficiency of the training process, models are often trained on batches of data and the optimization process thus aims at minimizing the average

Table 1.1: *nDCG* scores for two instances I_A and I_B. Green cells indicate relevant items, while red cells indicate non-relevant items. Items are ranked from left to right.

Instance	Ranked Items	*nDCG* Score
I_A	🟥🟩🟥🟥🟥🟥	0.63
I_B	🟥🟩🟩🟩🟩🟩	0.78

loss on a set of instances. As a result, each instance is assumed to be equally informative, while in reality, this might not be the case.

As will become apparent in chapter 2, the bounds of the aforementioned metrics and their listwise losses are either not upper- or lower-bounded. Instead, their bounds heavily depend on 1. the number of relevant and non-relevant items in a given instance and 2. the ratio of relevant to non-relevant items. This means that the worst possible ordering of items yields a score greater than zero.

Figure 1.1 visualizes the lower bound of *nDCG* and *AP* as a function of the fraction of relevant items in an instance with instance size $N \in \{25, 50, 100, 200\}$. While the lower bound of *nDCG* is more dependent on the instance size, the fraction of relevant items has a big impact on the lower bound of both metrics. In a typical training dataset, the number of (relevant and non-relevant) items is not constant and as a consequence, the bounds of the listwise losses are not constant across users or queries. In addition, instances with a high relevant-to-non-relevant item ratio have a greater lower bound than instances with a lower relevant-to-non-relevant item ratio. One could therefore argue that it is easier to produce high scores for instances with a higher relevant-to-non-relevant item ratio and that the metric score does not solely reflect the performance of the model, but also depends on the instance properties. To illustrate this, Table 1.1 lists two ranked instances I_A and I_B. Instance I_A ranks its only relevant item at the second rank and achieves an *nDCG* score of 0.63. But while instance I_B ranks all its relevant items at the bottom, its *nDCG* score is still higher than that of instance I_A. As the optimization process aims at minimizing the average loss on the training set, the lack of and inconsistency in upper- and lower-bounds might pose a problem.

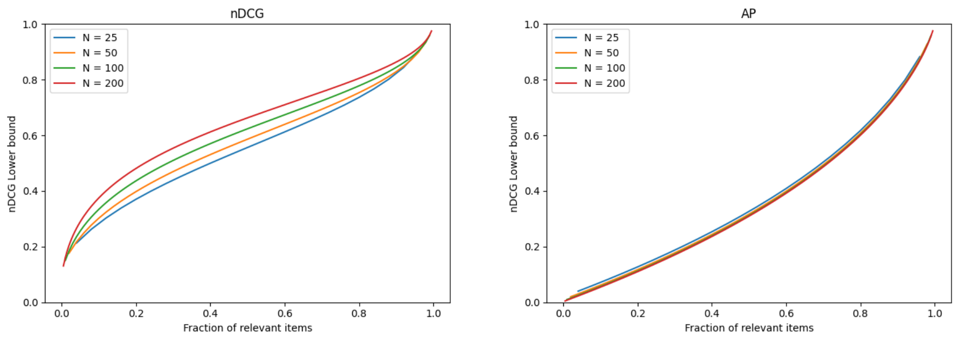


Figure 1.1: Lower bounds of *nDCG* and *AP* as a function of the fraction of relevant items in an instance of size $N \in \{25, 50, 100, 200\}$.

Considering the above, we argue that different instances have different ranking difficulty. We call this ranking difficulty 'instance-difficulty' and note that popular listwise loss functions do not account for this difficulty, which might negatively impact the performance of ranking functions.

1.1. RESEARCH GOALS

This thesis sets out to define bounding methods to create difficulty-aware losses. In addition, we analyze the impact of user/query wise metric bounding on the average performance of learning to rank models. Finally, we investigate the effect of user/query-wise metric bounding on individual user/query performance. To summarize, this thesis aims to answer the following questions:

- How can instance-difficulty be quantified and incorporated in listwise loss functions?
- Does user/query-wise metric bounding affect the average performance of learning to rank models?
- What effect does user/query-wise metric bounding have on individual user/query performance?

1.2. MAIN FINDINGS

In this thesis, we define multiple notions of instance-difficulty and utilize these notions to define four bounding methods. The proposed bounding methods are applied to three popular listwise losses and empirical results on two real-world datasets are obtained. Experimental results indicate that optimizing bounded variants of popular listwise loss functions may increase the average performance of the ranking models, albeit often marginally. In addition, we find that user/query-wise metric bounding offers benefits when considering $nDCG@k$ for evaluation, further increasing the recommendation utility and making metric bounding interesting for applications where users are presented with a small selection of recommended items. To further analyze the impact of the proposed bounding methods, we analyzed the change in performance on instance level. We found that in addition to increasing the overall performance, optimizing a bounded variant of $nRBP$ may increase the recommendation performance for instances with a higher number of relevant items.

Overall, our results show promising results for user/query-wise metric bounding in LTR, especially when applied to $nRBP$.

1.3. THESIS OUTLINE

The remainder of this thesis is organized as follows. In chapter 2, we describe a general framework for LTR, give a brief overview of popular models and loss functions, and describe popular evaluation metrics. In chapter 3, this work is further motivated and four bounding methods are introduced. chapter 4 describes the experimental setup of our experiment, the results of which are presented in chapter 5. Finally, in chapter 6 this thesis is concluded.

2 BACKGROUND

In this chapter, the required knowledge to understand the problem and its proposed solution is described. First, we describe a general framework for LTR and continue by introducing IR- and Recommendation-specific models within this framework. Second, the evaluation methodology often used to evaluate ranking models is introduced. Next, three widely used IR metrics are described. We conclude this chapter by introducing three listwise loss functions used throughout the rest of this thesis.

2.1. LEARNING TO RANK FRAMEWORK

Learning to Rank is the application of supervised machine learning techniques for training models which aim to solve the ranking problem introduced in chapter 1. In supervised machine learning, models are fit on training data comprised of example inputs and target vectors, also called instances. While LTR represents a family of models, we can distinguish between three main types, namely pointwise, pairwise and listwise approaches [1].

The pointwise approach aims to predict the relevance of a single item and methods for classification and regression can be applied. Instances are defined as a single Query-Item pair and hence these approaches also define loss functions based on individual items. Popular examples of pointwise approaches include Subset Ranking [2], McRank [3] and Prank [4].

In pairwise approaches, instances are defined as a pair of items and the problem is formalized as that of pairwise classification or regression. More specifically, pairwise models take two items as input and aim to predict which one should be ranked higher than the other in the final order. By making a prediction for all item pairs, a total order can be created. In contrast to the pointwise approach, pairwise approaches define loss functions as a function of two items. Examples of this approach include RankBoost [5], RankNet [6], LambdaRank [7] and LambdaMART [8].

Finally, listwise approaches take item lists as instances and are capable of utilizing listwise loss functions [9]. A great advantage of this approach is that the group structure of ranking is maintained and can be incorporated into the loss function. Instances

Table 2.1: Notation Rules

Meaning	Notation
Number of Queries or Users (Topics)	T
Query or User (Topic)	t_i or t
Number of items in an instance t	N or N_t
Number of relevant items in an instance t	P or P_t
A ranked instance for topic t of size N of which P items are relevant	$X^t_{(N,P)}$ or $X_{(N,P)}$
Ground-truth relevance of the i^{th} ranked item in instance x for topic t	x_i or x^t_i
Ground-truth relevance of item i for Topic t	y_i or y^t_i
Predicted relevance of item i for Topic t	$\hat{y}_i$ or $\hat{y}^t_i$
Loss function	$L(\cdot)$
Metric function	$M(\cdot)$
Sigmoid function with gain a	$\sigma_a(\cdot)$
Ranking Position (See Equation 2.1)	$R_{ti}(\cdot)$
Smooth Ranking Position (See Equation 2.9)	$\tilde{R}_{ti}(\cdot)$

are defined as list of items for a given topic $v^x = \{v^x_1, \dots, v^x_{N_x}\}$ and its corresponding relevance $y^x = \{y^x_1, \dots, y^x_{N_x}\}$. A ranking function f will output a score $f(v^x_j)$ for each item v^x_j, resulting in a list with predicted relevance scores $\hat{y}^x = (f(v^x_1), \dots, f(v^x_{N_x}))$. The training objective then becomes minimizing

$$\sum_{i=1}^{T} L\left(y^i, \hat{y}^i\right),$$

where L is a listwise loss function.

2.2. MODELS

Using this framework, LTR can be introduced from two perspectives, namely Recommendation and IR. The goal of a ranking function in a typical recommendation setting is to provide relevant items for a given user. Users and items are often represented by a non-informative identifier. For a system with T users and N items, interactions between users and items can be represented by a $T \times N$ matrix Y, where each element y_{ti} describes an interaction between user t and item i. These interactions can represent, among other things, clicks or ratings, but are assumed to be binary relevance throughout the rest of this thesis.

Matrix factorization is one of the most effective methods for learning a ranking function from this data and can be combined with direct optimization of IR metrics [10, 11, 12]. In this approach, users and items are represented by latent features which aim to capture the characteristics of users and items. For a given user t and item i and their associated latent vectors p_t and q_i, the predicted relevance is given by the dot product $p_u \cdot q_i$. Using the latent vectors of users and items, the recommendation model can predict a $T \times N$ matrix $F^{T \times N}$ with element f_{ti} representing the relevance of item i to user t. The ranking position R_{ti} follows from a pairwise comparison between the predicted relevance score for item i and all other items, as can be seen in Equation 2.1.

$$R_{ti} = 1 + \sum_{j=1\setminus i}^{N} \mathbb{I}(f_{tj} > f_{ti}), \tag{2.1}$$

where $\mathbb{I}(\cdot)$ denotes the indicator function.

More complex models which aim to capture nonlinear and more complicated representations between users and items exist [13], but are outside of the scope of this thesis.

When considering IR, data is often represented in a different manner and hence other machine learning methods can be applied [9]. Instead of having identifiers for queries and documents, a feature vector is created from every query-document pair. These features can be categorized into three main classes: features based on the query, features based on the document and features based on both the query and document. The predicted ranking position R_{ti} of item i for query t again follows from a pairwise comparison between the predicted relevance score for item i and all other items using Equation 2.1.

Cao et al. [9] propose ListNet, a listwise approach which employs a neural network as model and gradient decent as algorithm. The method is heavily inspired by RankNet [6], but while RankNet deploys a pairwise loss function, ListNet uses a listwise loss to optimize the model parameters. Other well-known listwise LTR algorithms include AdaRank[14], ListMLE [15], ListRank-MF [16] and SoftRank [17].

2.3. EVALUATION

The performance of ranking models is often evaluated on a query-bases, using the method described below.

- Collect a (large) set of topics $S = \{t_1, t_2, \ldots, t_T\}$
- For every $t_i \in S$:
  - Collect a set of associated items for t_i
  - Collect the relevance of each item for t_i by human assessment
  - For each item, use the ranking model to predict the relevance for t_i
  - Use an evaluation measure to evaluate the performance of the ranking function for t_i
- Report the average measure on all topics in S

A wide variety of evaluation measures have been proposed. We will consider three widely-used measures and describe them below. Normalized discounted cumulative gain ($nDCG$) [18] is a measure of ranking quality which discounts the gain of a document based on its position in the ranked list. The idea is that documents with a lower rank are less valuable for the user, and hence a smaller share of its gain is added to the cumulated gain. For a topic t and a ranked set of items X, $nDCG$ is calculated as follows.

$$\begin{aligned} nDCG(t) &= \frac{DCG(t)}{IDCG(t)}, \text{ where} \\ DCG(t) &= \sum_{i=1}^{N} \frac{2^{x_i^t}-1}{\log_2(i+1)} \end{aligned} \tag{2.2}$$

and $IDCG$ represents the ideal DCG. Also, as often times only a small fraction of the results are displayed to the user, $nDCG$ may be cut off at a certain rank. $nDCG@k$, where k is the cut-off position is calculated using Equation 2.3.

$$\begin{aligned} nDCG@k(t) &= \frac{DCG@k(t)}{IDCG@k(t)}, \text{ where} \\ DCG@k(t) &= \sum_{i=1}^{k} \frac{2^{x_i^t}-1}{\log_2(i+1)} \end{aligned} \tag{2.3}$$

and $IDCG@k$ represents the ideal $DCG@k$ through position k.

Table 2.2: Computation of DCG

Rank (i)	Item	x_i^t	$\log_2(i+1)$	Gain
1	A	1	1.00	1.00
2	B	1	1.58	0.63
3	C	1	2.00	0.50
4	D	0	2.32	0.00
5	E	0	2.58	0.00
6	F	0	2.81	0.00
7	G	0	3.00	0.00
8	H	0	3.17	0.00
9	I	0	3.32	0.00
			Sum =	2.13

(a) Ideal DCG

Rank (i)	Item	x_i^t	$\log_2(i+1)$	Gain
1	I	0	1.00	0.00
2	H	0	1.58	0.00
3	G	0	2.00	0.00
4	F	0	2.32	0.00
5	E	0	2.58	0.00
6	D	0	2.81	0.00
7	C	1	3.00	0.33
8	B	1	3.17	0.32
9	A	1	3.32	0.30
			Sum =	0.95

(b) Worst DCG

Table 2.2 shows toy examples of the calculation of DCG for $N = 9$ documents of which $P = 3$ are relevant. The order of the result list is given by the order in which the documents are listed in the two tables.

Second, we consider Rank-Biased Precision (RBP) [19], a metric for scoring rankings which includes a simple user model by utilizing a persistence parameter $p \in [0,1)$. To model user behavior, it is assumed that users traverse the ranked list from top to bottom. In addition, it is assumed that users move on to the next document with probability p, and stop with probability $1-p$. A gain per document can then be calculated by multiplying the probability a user will look at the document by the relevance of that document. RBP is defined as follows.

$$RBP(t;p) = (1-p)\cdot\sum_{i=1}^{N} x_i^t \cdot p^{i-1}, \tag{2.4}$$

Table 2.3: Computation of $RBP(t)$, $p = 0.7$

Doc	R_{ti}	y_i^t	$p^{R_{ti}-1}$	Gain
A	1	1	1.00	1.00
B	2	1	0.70	0.70
C	3	1	0.49	0.49
D	4	0	0.34	0.00
E	5	0	0.24	0.00
F	6	0	0.17	0.00
G	7	0	0.12	0.00
H	8	0	0.08	0.00
I	9	0	0.06	0.00
			Sum	2.19
			$\times(1-p)$	0.66

(a) Ideal RBP

Doc	R_{ti}	y_i^t	$p^{R_{ti}-1}$	Gain
I	1	0	1.00	0.00
H	2	0	0.70	0.00
G	3	0	0.49	0.00
F	4	0	0.34	0.00
E	5	0	0.24	0.00
D	6	0	0.17	0.00
C	7	1	0.12	0.12
B	8	1	0.08	0.08
A	9	1	0.06	0.06
			Sum	0.26
			$\times(1-p)$	0.08

(b) Worst RBP

where p is the persistence of the user. Table 2.3 shows examples of the computation of RBP.

To align the bounds of RBP with the other metrics used in this theis, we consider the normalized Rank-Biased Precision, or $nRBP$, which normalizes the bare RBP by the maximum obtainable RBP [20]. Formally, $nRBP$ is defined as follows:

$$nRBP(t;p) = Z(t,p) \cdot RBP, \tag{2.5}$$

where $Z(t,p) = 1/(1-p^{P^t})$, the normalization factor. Note that p denotes the persistence parameter and P^t the number of relevant items for topic t.

Finally, Average Precision (AP) [21] is the average of the $Precision@k$ values at each relevant item in the ranking. Given a topic t and its ranking permutation x^t, $AP(t)$ is given by Equation 2.6.

$$AP(t) = \frac{\sum_{i=1}^{N_t} Precision@i \times x_i^t}{P_t}, \tag{2.6}$$

where $Precision@k$ is the precision at cut-off k and x_i^t is the relevance of the document at rank i. Two toy examples of the calculation of AP are given in Table 2.4.

2.4. LISTWISE LOSS FUNCTIONS

Qin et al. [22] describe three properties a good listwise loss function should have. Loss functions should be insensitive to the number of document pairs, to avoid a high inconsistency between document-level losses and query-level losses. In addition, a good loss function should penalize ranking errors at the top more than ranking errors at the bottom. And finally, query level losses should be upper bounded, to avoid queries with large losses to dominate the training process. More general, there should exist a constant C such that for any topic t

$$L(y^t, \hat{y}^t) \leq C$$

Table 2.4: Computation of AP

Doc	Rel	Precision
A	1	1/1
B	1	2/2
C	1	3/3
D	0	-
E	0	-
F	0	-
G	0	-
H	0	-
I	0	-
	Sum	3.00
	$/R$	1.00

(a) Ideal *AP*

Doc	Rel	Precision
I	0	-
H	0	-
G	0	-
F	0	-
E	0	-
D	0	-
C	1	1/7
B	1	2/8
A	1	3/9
	Sum	0.73
	$/R$	0.24

(b) Worst *AP*

IR metrics such as $nDCG$ and AP meet these requirements, but cannot be deployed as loss due to the non-differentiable nature of the sorting operation which is used in these metrics. Following Li et al. [20] and using the notation as given in Table 2.1, $nDCG$ and AP can be written as a function of the predicted relevance using Equation 2.7 and Equation 2.8.

$$nDCG(t) = \frac{\sum_{i=1}^{N_t} (2^{y_i^t} - 1)/\log_2(R_{ti}+1)}{\sum_{i=1}^{P_t} 1/\log_2(i+1)} \tag{2.7}$$

$$AP(t) = \frac{1}{P_t} \sum_{i=1}^{N_t} \frac{y_i^t}{R_{ti}} \sum_{i=1}^{N_t} y_j^t \mathbb{I}(R_{tj} \leq R_{ti}) \tag{2.8}$$

To address the problem of non-differentiability, a smooth function such as a sigmoid or ReLU can be used to approximate the indicator function [20, 11, 23]. Using a sigmoid function, the approximated ranking position $\tilde{R}_{ti}$ can be calculated using Equation 2.9.

$$\tilde{R}_{ti} = 1 + \sum_{j=1 \setminus i}^{N_t} \sigma(f_{tj} - f_{ti}), \tag{2.9}$$

where $\sigma(x) = 1/(1+e^{-x})$.

Using $\tilde{R}_{ti}$, the smooth variants of AP and $nDCG$ for topic t can be formulated as follows:

$$\widetilde{nDCG}(t) = \frac{\sum_{i=1}^{N_t} (2^{y_i^t} - 1)/\log_2(\tilde{R}_{ti}+1)}{\sum_{i=1}^{P_t} 1/\log_2(i+1)} \tag{2.10}$$

$$\widetilde{AP}(t) = \frac{1}{P_t} \sum_{i=1}^{N_t} \frac{y_i^t}{\tilde{R}_{ti}} (1 + \sum_{j=1 \setminus i}^{N_u} y_j^t \sigma(f_{tj} - f_{ti})) \tag{2.11}$$

As the target of the optimization process is to maximize $nDCG$ and AP, we consider the additive inverse of Equation 2.10 and Equation 2.11 as loss. In addition, we will consider the RBP-based loss function proposed by Li et al. [20], which is given by

$$\widetilde{nRBP}(t) = \sum_{i=1}^{N_t} y_i^t (\tilde{R}_{ti} - 1) - \sum_{j=1}^{P_t} (j-1) \tag{2.12}$$

3 Metric Bounding

During training, a loss function and its derivatives are utilized to adjust the model parameters. An iterative optimization algorithm such as Gradient Descent or Adam utilizes a loss function and batches of instances to update the model in order to decrease the average loss on a given batch. The direction and size of change are determined by the loss function and its derivatives with respect to the model parameters. Losses such as $nDCG$, AP and $nRBP$ do not account for instance-difficulty. Instead, each instance is assumed to be equally informative, while in reality, this might not be the case. This could negatively impact the performance of the model. It might therefore be beneficial to take this instance-difficulty into consideration during training, by incorporating it in the loss function. In this chapter, we describe three notions of instance-difficulty and provide four bounding methods to create difficulty-aware losses. In addition, the required upper bounds, lower bounds and expectations of instances for the losses introduced in the previous chapter are given. Bounds, expectations, loss scores and metric scores are calculated on instances. For simplicity, and following the notation rules as introduced in Table 2.1, we let $X_{(N,P)}$ denote an instance with N documents, of which P are relevant and x_i denote the relevance of the i^{th} ranked item in an instance. In addition, we let $M(\cdot)$ be a metric and $\tilde{M}(\cdot)$ be its smooth approximate. Finally, we note that in case of $nDCG$ and AP, we need to use their additive inverse as loss function, but ignore that in this chapter for simplicity.

3.1. Min-Max Normalization

The first aspect which motivates the use of user/query-wise metrics bounding, is the inconsistency of bounds of the listwise loss functions introduced the previous section. As previously shown, $nDCG$ and AP and their listwise losses do not have a constant lower bound of zero across instances. Instead, their lower bounds depend on the number of relevant and non-relevant items in a given instance.

In a typical IR and RecSys setting, it is safe to assume the number of relevant items is dwarfed by the number of non-relevant items. In addition, the number of rated (and thus relevant) items is often not constant across topics. To address this issue of unbalanced data, the majority class is often under-sampled in the training (and testing) data. This results in instances to have a fixed ratio of relevant to non-relevant items but does not address the inconsistency in the number of relevant items across topics. Due to this inconsistency in the number of relevant items, the lower bounds are not constant across instances. As a consequence, it is easier to produce high scores for instances with a high number of relevant items. We can therefore say that the metric/loss score does not solely reflect the performance of the model, but also depends on the number of relevant items in an instance.

A simple, yet effective method to create constant upper and lower bounds across topics is to apply Min-Max Normalization. Min-Max Normalization addresses the issue of inconsistent bounds by bringing all values into the range $[0,1]$. Given an instance x and a metric function $M(\cdot)$, the bounded and smooth metric $\tilde{M}_{MM}(\cdot)$ is given by Equation 3.1.

$$\tilde{M}_{MM}(x) = \frac{\tilde{M}(x) - M_{min}(x)}{M_{max}(x) - M_{min}(x)}, \tag{3.1}$$

where $M_{min}(x)$ and $M_{max}(x)$ denote the minimal and maximal loss achievable on instance x, respectively. We continue by providing formulas for the upper and lower bounds of $nDCG$, $nRBP$ and AP and show their dependence on the number of relevant and non relevant items in an instance.

3.1.1. Bounds of $nDCG$

The largest obtainable $nDCG$ score is obtained when ranking all relevant items at the top, producing an $nDCG$ of 1. In contrast, the smallest obtainable $nDCG$ score is obtained when ranking all relevant items at the bottom. For an instance with N items of which P are relevant, this means ranking the relevant items at ranks $N-P+1, N-P+2, \ldots, N-P+P$, producing discounted gains of $\frac{1}{\log_2(N-P+2)}, \frac{1}{\log_2(N-P+3)}, \ldots, \frac{1}{\log_2(N-P+P+1)}$. It follows that the minimal obtainable $nDCG$ is given by Equation 3.2.

$$nDCG_{min}(X_{(N,P)}) = \sum_{i=N-P+1}^{N} \frac{1}{\log_2(i+1)} \tag{3.2}$$

3.1.2. Bounds of $nRBP$

While $nRBP$ has a lower bound of 0, its upper bound is not constant across topics [20]. The largest $nRBP$ loss is obtained when ranking all relevant items at the bottom. For an instance $X_{(N,P)}$, this means ranking the relevant items at ranks $N-P+1, N-P+2, \ldots, N-P+P$. This results in the following $nRBP$ loss.

$$nRBP_{max}(X_{(N,P)}) = \sum_{i=N-P+1}^{N} (i-1) - \sum_{j=1}^{P} (j-1)$$

We can rewrite the summations to obtain Equation 3.3.

$$nRBP_{max}(X_{(N,P)}) = \frac{1}{2}P(2N-P-1) - \frac{1}{2}P(P-1) \tag{3.3}$$

3.1.3. BOUNDS OF AP

Recall that AP is the average of the $Precision@k$ values at each relevant document in the ranking. The optimal AP is obtained when ranking all P relevant documents at the top, producing an AP score of 1. To obtain the smallest AP score, relevant documents are ranked at ranks $N-P+1, N-P+2, \dots, N-P+P$, producing $Precision@k$ values of $\frac{1}{N-P+1}, \frac{2}{N-P+2}, \dots, \frac{P}{N-P+P}$. The AP score is then obtained by summing these values and dividing by P. It naturally follows that the smallest obtainable AP score for an instance $X_{(N,P)}$ is given by Equation 3.4.

$$AP_{min}(X_{(N,P)}) = \frac{1}{P}\sum_{i=1}^{P}\frac{i}{N-P+i} \tag{3.4}$$

3.1.4. IMPACT OF BOUNDING ON LOSS SCORES

Recall that, as mentioned in chapter 2, the goal of optimization is to maximize $nDCG$ and AP. To align the analysis of the impact of metric bounding on $nDCG$ and AP with that of $nRBP$, we consider their additive inverse as loss. Figure 3.1 shows the bounded loss scores as a function of the unbounded loss scores for three instances with $NSR = 2$ (top) and three instances with $NSR \in \{1,2,3\}$ (bottom), by loss function (columns).

As expected, `min-max bounding` guarantees consistent bounds across instances. The resulting loss function is thus insensitive to the number of (relevant) items in an instance. We analyze the consequences of this in chapter 5.

We further observe minimal changes for the bounded AP loss, hence we expect `min-max bounding` to result in minimal changes in datasets with a constant NSR.

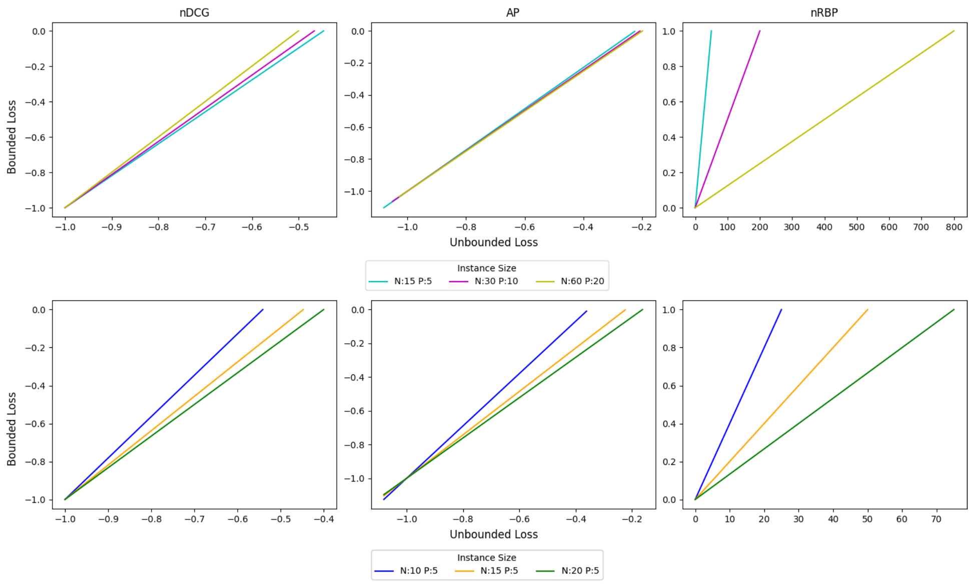


Figure 3.1: Analyzing the impact of `min-max bounding` on the loss score for three instances with $NSR = 2$ (top) and three instances with $NSR \in \{1,2,3\}$ (bottom), by loss function (columns).

3.2. EXPECTATION-BASED METRIC BOUNDING

A second method to create difficulty-aware losses, is to consider the expected score of a random guess. This method is heavily inspired by Gienapp et al. [24], who propose the expected $nDCG$ score of a hypothetical random ranker as a method to identify difficult and non-difficult instances. The expected $nDCG$ score of a random ranker or $RnDCG$ is calculated as follows:

$$RnDCG(t) = \frac{\sum_{r=1}^{N_t} \frac{\mu}{\log_2(r+1)}}{IDCG(t)}, \tag{3.5}$$

where μ denotes the average gain over the set of documents.

We adopt this method and use the expectation of a metric as a method to quantify instance-difficulty. Take as example two instances A and B, A then has a higher instance-difficulty if

$$\mathbb{E}[M(A))] < \mathbb{E}[M(B))],$$

where $\mathbb{E}[M(I)]$ denotes the expectation of metric M given instance I.

The expectation of a metric for an instance is given by the average of the scores produced by all possible permutations of that instance and can be utilized to quantify instance-difficulty [24]. We propose `expectation based bounding` (EB), which aims at utilizing this expectation in order to account for instance-difficulty. For a metric $M(\cdot)$, the bounded metric $\tilde{M}_{EB}(\cdot)$ is given by Equation 3.6.

$$\tilde{M}_{EB}(x) = \frac{\tilde{M}(x)}{\mathbb{E}[M(x)]} \tag{3.6}$$

We continue by providing formulas for $\mathbb{E}[nDCG]$, $\mathbb{E}[nRBP]$ and $\mathbb{E}[AP]$ and again show its dependence on the number of relevant and non relevant items in an instance.

3.2.1. EXPECTATION OF $nDCG$

The expectation of $nDCG$ for an instance $X_{(N,P)}$, or $\mathbb{E}[nDCG(X_{(N,P)})]$, is given by the average of the $nDCG$ scores produced by all possible orderings of that instance. As the ideal DCG ($IDCG$) score is independent of the ranking, we can write the expectation as follows:

$$\mathbb{E}[nDCG(X_{(N,P)})] = \frac{\mathbb{E}[DCG(X_{(N,P)})]}{IDCG(X_{(N,P)})} \tag{3.7}$$

We further note that DCG is the sum of multiple terms g_i and that the value of term g_i depends on the relevance of the item at rank i. Given the relevance x_i of the item at rank i, the value of g_i is given by Equation 3.2.1.

$$g_i = \frac{2^{x_i} - 1}{\log_2(i+1)}$$

Table 3.1 shows examples of the computation of the expectation of two instances $X_{(2,1)}$ and $X_{(3,1)}$. Note that in an instance with N items of which P are relevant, we have $N!$ permutations of which $\frac{N!}{P!*(N-P)!}$ are distinct. To ease calculations, we consider all $N!$

permutations and thus include duplicates. Below, we describe these examples in more detail and finally provide a general function for computing $\mathbb{E}[nDCG(X_{(N,P)})]$.

Table 3.1: Calculation of $\mathbb{E}[DCG(X_{(N,P)})]$

x_1	x_2	DCG
		$g_1 + g_2$
1	0	$\frac{2^1-1}{\log_2(2)} + \frac{2^0-1}{\log_2(3)}$
0	1	$\frac{2^0-1}{\log_2(2)} + \frac{2^1-1}{\log_2(3)}$

(a) $\mathbb{E}[DCG(X_{(2,1)})]$

x_1	x_2	x_3	DCG
			$g_1 + g_2 + g_3$
1	1	0	$\frac{2^1-1}{\log_2(2)} + \frac{2^1-1}{\log_2(3)} + \frac{2^0-1}{\log_2(4)}$
1	0	1	$\frac{2^1-1}{\log_2(2)} + \frac{2^0-1}{\log_2(3)} + \frac{2^1-1}{\log_2(4)}$
1	1	0	$\frac{2^1-1}{\log_2(2)} + \frac{2^1-1}{\log_2(3)} + \frac{2^0-1}{\log_2(4)}$
1	0	1	$\frac{2^1-1}{\log_2(2)} + \frac{2^0-1}{\log_2(3)} + \frac{2^1-1}{\log_2(4)}$
0	1	1	$\frac{2^0-1}{\log_2(2)} + \frac{2^1-1}{\log_2(3)} + \frac{2^1-1}{\log_2(4)}$
0	1	1	$\frac{2^0-1}{\log_2(2)} + \frac{2^1-1}{\log_2(3)} + \frac{2^1-1}{\log_2(4)}$

(b) $\mathbb{E}[DCG(X_{(3,2)})]$

To calculate $\mathbb{E}[DCG(X_{(2,1)})]$, we need to sum all DCG scores and divide by the number of possible orderings, which equals $N!$, 2 in this case.

$$\mathbb{E}[DCG(X_{(2,1)})] = \frac{\frac{2^1-1}{\log_2(2)} + \frac{2^0-1}{\log_2(3)} + \frac{2^0-1}{\log_2(2)} + \frac{2^1-1}{\log_2(3)}}{2} = \frac{\frac{2^1-1}{\log_2(2)} + \frac{2^1-1}{\log_2(3)}}{2}$$

For $\mathbb{E}[DCG(X_{(3,1)})]$, we again need to sum all DCG scores and divide by the number of possible orderings. Note that the DCG score is always the sum of multiple terms g_i, divided by the $\log_2(\cdot)$. We can thus first add up all terms which are divided by the same $\log_2(\cdot)$, and finally divide by that log.

$$\mathbb{E}(DCG(X_{(3,1)})) = \frac{\frac{4\cdot(2^1-1)}{\log_2(2)} + \frac{4\cdot(2^1-1)}{\log_2(3)} + \frac{4\cdot(2^1-1)}{\log_2(4)}}{(3)!}$$

$$= \frac{4/6\cdot(2^1-1)}{\log_2(2)} + \frac{4/6\cdot(2^1-1)}{\log_2(3)} + \frac{4/6\cdot(2^1-1)}{\log_2(4)}$$

Note that each term g_i is non-zero exactly $\frac{P}{N}\cdot N!$ times. More general, we can write the expectation of DCG for a instance $X_{(N,P)}$ as follows:

$$\mathbb{E}[DCG(X_{(N,P)})] = \frac{\frac{\frac{P}{N}\cdot N!\cdot(2^1-1)}{\log_2(2)} + \frac{\frac{P}{N}\cdot N!\cdot(2^1-1)}{\log_2(3)} + \cdots + \frac{\frac{P}{N}\cdot N!\cdot(2^1-1)}{\log_2(N+1)}}{N!}$$

$$= \frac{\frac{P}{N}\cdot(2^1-1)}{\log_2(2)} + \frac{\frac{P}{N}\cdot(2^1-1)}{\log_2(3)} + \cdots + \frac{\frac{P}{N}\cdot(2^1-1)}{\log_2(N+1)} = \sum_{r=1}^{N} \frac{\frac{P}{N}\cdot(2^1-1)}{\log_2(r+1)}$$

It follows that the expectation of $nDCG(\cdot)$ is given by:

$$\mathbb{E}[nDCG(X_{(N,P)})] = \frac{\sum_{r=1}^{N} \frac{\frac{P}{N}}{\log_2(r+1)}}{IDCG(X_{(N,P)})} \tag{3.8}$$

3.2.2. EXPECTATION OF $nRBP$

To compute $\mathbb{E}[nRBP(X_{(N,P)})]$, we take a similar approach and ignore the second term of $nRBP$ (Equation 2.12), which ensures a lower bound of 0 for all instances and is independent of the ranking.

$$\mathbb{E}[nRBP(X_{(N,P)})] = \mathbb{E}[L^1_{nRBP}(X_{(N,P)})] - \sum_{j=1}^{M}(j-1), \text{ where}$$

$$L^1_{nRBP}(X_{(N,P)}) = \sum_{i=1}^{N} y_i^t(\tilde{R}_{ti}-1) = \sum_{i=1}^{N} x_i \cdot (i-1)$$

We continue to compute the expectation of $L^1_{nRPB}(\cdot)$ and note that it is equal to the sum of multiple terms $g_i = x_i \cdot (i-1)$. Table 3.2 shows examples of the computation of $L^1_{nRPB}(\cdot)$.

Table 3.2: Calculation of $\mathbb{E}[L^1_{nRBP}(X_{(N,P)})]$

		L^1_{nRBP}
x_1	x_2	g_1+g_2
1	0	$1\cdot(1-1)+0\cdot(2-1)=0$
0	1	$0\cdot(1-1)+1\cdot(2-1)=1$

(a) $\mathbb{E}[nRBP(X_{(2,1)})]$

			L^1_{nRBP}
x_1	x_2	x_3	$g_1+g_2+g_3$
1	1	0	$1\cdot(1-1)+1\cdot(2-1)+0\cdot(3-1)=1$
1	0	1	$1\cdot(1-1)+0\cdot(2-1)+1\cdot(3-1)=2$
1	1	0	$1\cdot(1-1)+1\cdot(2-1)+0\cdot(3-1)=1$
1	0	1	$1\cdot(1-1)+0\cdot(2-1)+1\cdot(3-1)=2$
0	1	1	$0\cdot(1-1)+1\cdot(2-1)+1\cdot(3-1)=3$
0	1	1	$0\cdot(1-1)+1\cdot(2-1)+1\cdot(3-1)=3$

(b) $\mathbb{E}(L^1_{nRBP}(X_{(3,2)})$

Each term g_i is non-zero exactly $\frac{P}{N}\cdot N!$ times, producing a score of $(i-1)$. To compute the expectation of g_i, we sum these values and divide by the total number of permutations to obtain

$$\mathbb{E}[g_i] = \frac{\frac{P}{N}\cdot N!}{N!}\cdot(i-1) = \frac{P}{N}\cdot(i-1).$$

It follows that the expectation of L^1_{nRBP} can be computed as follows.

$$\mathbb{E}[L^1_{nRBP}(X_{(N,P)})] = \sum_{i=1}^{N}\mathbb{E}[g_i] = \frac{P}{N}\cdot\sum_{i=1}^{N}(i-1).$$

Now, when again introducing the normalizing term of Equation 2.12, the expectation becomes:

$$\mathbb{E}[nRBP(X_{(N,P)})] = \frac{P}{N} \cdot \sum_{i=1}^{N} (i-1) - \sum_{j=1}^{P} (j-1)$$
$$= \frac{P}{N} \cdot \frac{1}{2}(N-1)N - \frac{1}{2}(P-1)P = \frac{1}{2} \cdot P \cdot (N-P) \tag{3.9}$$

3.2.3. EXPECTATION OF *AP*

The expectation of Average Precision for an instance $X_{(N,P)}$ is equal to the average of the *AP* scores produced by all possible orderings of that instance. Table 3.3 shows two examples.

Table 3.3: Calculation of $\mathbb{E}[AP(X_{(N,P)})]$

x_1	x_2	AP
1	0	1/1
0	1	1/2
	sum:	3/2
	$\mathbb{E}$:	3/4

(a) $\mathbb{E}[AP(X_{(2,1)})]$

x_1	x_2	x_2	AP
1	1	0	(1/1 + 2/2)/2
1	1	0	(1/1 + 2/2)/2
1	0	1	(1/1 + 2/3)/2
1	0	1	(1/1 + 2/3)/2
0	1	1	(1/2 + 2/3)/2
0	1	1	(1/2 + 2/3)/2
		sum	29/6
		$\mathbb{E}$	29/36

(b) $\mathbb{E}[AP(X_{(3,2)})]$

Following Y. Bestgen [25], ignoring the two divisors P and the total number of (different) permutations, $\mathbb{E}[AP(\cdot)]$ is the sum of precision scores at each rank. The idea is to calculate, for each rank n, the probability of having the i^{th} relevant document at rank n. This would produce a score of i/n. This probability is equal to the probability of having i successes in n draws from a population of size N with P successes. In addition, we require the last draw to be a success. The probability of having i successes in n draws from a population of size N with P successes is given by the following formula.

$$Pr(X=i) = \frac{\binom{P}{i}\binom{N-P}{n-i}}{\binom{N}{n}}$$

The probability of the last draw to be a success when having n draws and i successes is $\frac{i}{n}$. Combining this probability with $Pr(X=i)$ and the produced scores, results in the following expectation for AP.

$$\mathbb{E}[AP(X_{N,P})] = \frac{\sum_{i=1}^{P} \sum_{n=i}^{N-P+i} \frac{i}{n} \cdot \frac{i}{n} \cdot Pr(X=i)}{P} \tag{3.10}$$

3.2.4. IMPACT OF BOUNDING ON LOSS SCORES

To align the analysis of the impact of metric bounding on $nDCG$ and AP with that of $nRBP$, we consider their additive inverse as loss. Figure 3.2 visualizes the expectation of the loss scores of $nDCG$, AP and $nRBP$ as a function of the number of relevant items in an instance for instances with a constant ratio of non-relevant to relevant items (negative sampling ratio). We can make several interesting observations. First, for all losses we observe a positive correlation between the NSR and the expected loss score. This observation indicates that having a higher fraction of non-relevant items in an instance makes the ranking task harder.

Second, the expectation of $nDCG$ loss is a decreasing function from $P \geq 5$ relevant items. This means that for a constant NSR, instances with a higher number of relevant items are expected to perform better. In contrast, the expectation of AP and $nRBP$ is, for a constant NSR, a increasing function with respect to the number of relevant items in an instance. This means that for a given NSR, larger instances are more difficult.

To analyze the effect of the bounding methods on the produced loss scores, Figure 3.3 shows the bounded loss value, as a function of the unbounded loss value for three instances with $NSR = 2$ (top) and three instances with $NSR \in \{1,2,3\}$ (bottom), by loss function (columns). We can make the following observations. First, while bare $nDCG$ and AP have consistent upper bounds across topics, the bounds of the bounded losses are depended on the instance properties. In contrast, we observe that the bounded $nRBP$ losses have consistent upper and lower bounds across instances. As a result, we expect that that `expectation based bounding` and `expectation-max bounding` will achieve similar performance as
`min-max bounding` . We investigate whether this is indeed the case in chapter 5.

When considering instances with different NSR, we observe that instances with a higher NSR are able to achieve higher losses, making the training process more sensitive to these difficult instances.

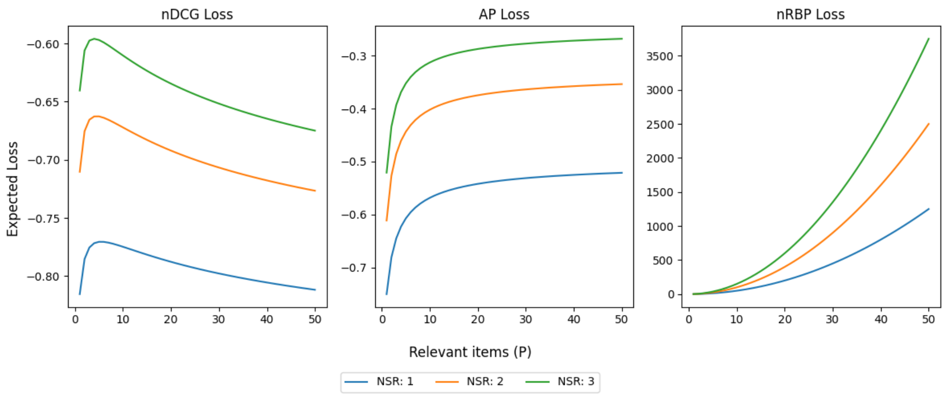


Figure 3.2: Expectation of $nDCG$, AP and $nRBP$ for different negative sampling ratios, as function of the number of relevant items in an instance.

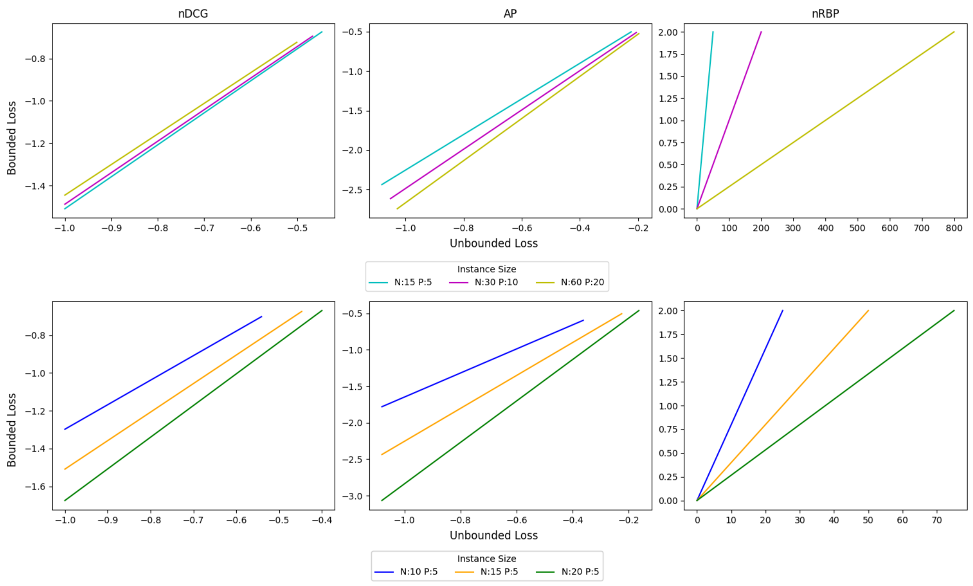


Figure 3.3: Analyzing the impact of `expectation based bounding` on the loss score for three instances with $NSR = 2$ (top) and three instances with $NSR \in \{1,2,3\}$ (bottom), by loss function (columns).

3.3. EXPECTATION-MAX BOUNDING

We propose a third method bounding method which combines Min-Max Normalization with the expectation of a metric. `Expectation-max bounding` applies Min-Max normalization, but takes the expectation as the minimal value. More formally, `expectation-max bounding` is given by Equation 3.11.

$$\tilde{M}_{EM}(x) = \frac{\tilde{M}(x) - \mathbb{E}[M(x)]}{M_{max}(x) - \mathbb{E}[M(x)]} \tag{3.11}$$

We use the formulas for the expectation of $nDCG$, AP and $nRBP$ as given in the previous section and continue with an analysis on the impact on the produced loss scores.

3.3.1. IMPACT OF BOUNDING ON LOSS SCORES

To align the analysis of the impact of metric bounding on $nDCG$ and AP with that of $nRBP$, we consider their additive inverse as loss. Figure 3.4 shows the bounded loss value, as a function of the unbounded loss value for three instances with $NSR = 2$ (top) and three instances with $NSR \in \{1,2,3\}$ (bottom), by loss function (columns).

First, for instances with a constant NSR, we observe that `expectation-max bounding` has a similar impact as `expectation based bounding` on the produced loss scores and thus expect to achieve similar performance.

Second, when considering instances with different NSR, we observe that instances have a consistent lower bound of -1. However, when considering $nDCG$ and AP we observe the upper bound depends on the instance properties. We analyze the impact of this observation in chapter 5.

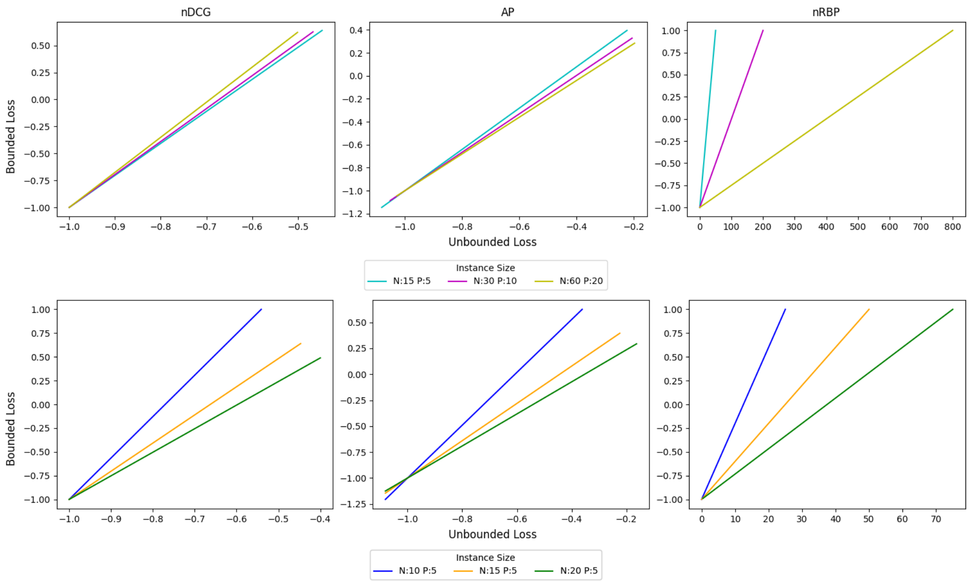


Figure 3.4: Analyzing the impact of `expectation-max bounding` on the loss score for three instances with $NSR = 2$ (top) and three instances with $NSR \in \{1,2,3\}$ (bottom), by loss function (columns).

3.4. DISTRIBUTION-BASED METRIC BOUNDING

While `expectation based bounding` and `expectation-max bounding` aim to create difficulty-aware losses by utilizing the performance of a random ranker, the resulting loss function is not guaranteed to have a constant lower and upper bound across instances. A second method of utilizing the performance of a random ranker is to use the distribution of scores produced by this random ranker. For a given instance and metric, a distribution of scores can be calculated: each of the possible orderings of the instance results in a particular value. The distribution is then defined by the probability of those orderings. A discrete cumulative distribution naturally follows. `distribution based bounding` aims at incorporating the instance difficulty by utilizing this distribution and simultaneously ensures a constant lower and upper bound of zero and one, respectively.

For a metric $M(\cdot)$ and instance x, the bounded metric $M_{DB}(x)$ is given by Equation 3.12.

$$\tilde{M}_{DB}(x) = F_{(x;M)}(\tilde{M}(x)), \tag{3.12}$$

where $F_{(x;M)}(\cdot)$ denotes the cumulative distribution for the given metric and instance, which maps a metric score to the fraction of instances which produce the same score or lower. This fraction is equivalent to the probability we outperform a random system.

While the distribution of measures like recall and $Precision@k$ have nice closed forms, for more complex measures like $nDCG$ and $nRBP$ the only resource is simulation [26]. We therefore propose a method to calculate $F_{(x;M)}(\cdot)$ for any metric, which makes no assumptions about the underlying distribution of the metric, but relies on precalculated probability density functions.

Given the discrete probability density function for a given metric M and instance X, which is given by the metric scores D_x and the probability of those scores D_y, $F_{(X;M)}(\cdot)$ can be calculated as shown in algorithm 1. The Heaviside function $H(\cdot)$ is defined as follows.

$$H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

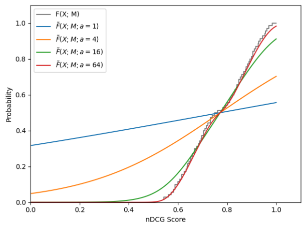
As a result of using the Heavyside function H in algorithm 1, the resulting CDF is not smooth. However, to allow $M_{DB}(x)$ to be used as loss, the function should be differentiable and thus smooth. Hence, $F(\cdot)$ should be a smooth function. To smoothen $F_{(x;M)}(S)$, we approximate the Heaviside function and deploy a sigmoid function σ_a with gain a. The resulting function $\tilde{F}_{(x;M;a)}(S)$ is given by algorithm 2. Figure 3.5 shows the approximated distributions for $nDCG$, AP and $nRBP$ for two example instances using different gains. When using the sigmoid function σ_a with default gain $a = 1$, the resulting function is smooth, but does not approximate the distribution well. In addition, the gain which results in a smooth distribution which closely resembles the underlying distribution, is not constant. We therefore propose a value for the gain which depends on both the metric and distribution size. The gain a is calculated using Equation 3.13.

$$a(D) = \frac{1}{\frac{max(D_x) - min(D_x)}{len(D_x)}}, \tag{3.13}$$

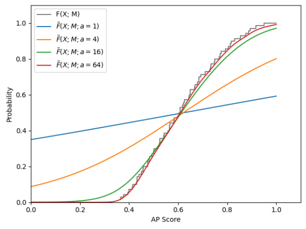
where $max(D_x)$ and $min(D_x)$ denote the maximal and minimal metric score in the distribution and $len(D_x)$ denote the number of scores in the distribution. Figure 3.6 shows the approximated distributions for $nDCG$, AP and $nRBP$ for two example instances.

Algorithm 1: Discrete $F_{(X;M)}(S)$
```
Input score $S$
Initialize distribution:
$D = (D_x, D_y)$

diff = $S$ - $D_x$
ans = $H$(diff) $\cdot D_y$
return $sum(ans)$
```

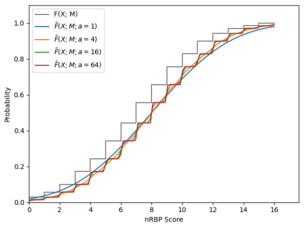
Algorithm 2: Smooth $\tilde{F}_{(X;M;a)}(S)$
```
Input score $S$
Initialize distribution:
$D = (D_x, D_y)$

diff = $S$ - $D_x$
ans = $\sigma_a$(diff) $\cdot D_y$
return $sum(ans)$
```



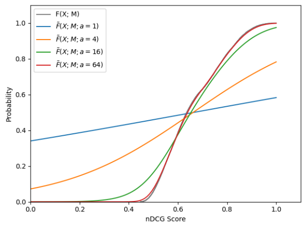
(a) *nDCG*



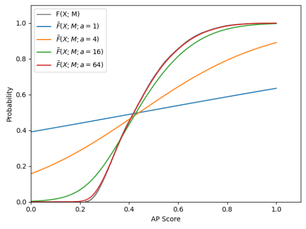
(b) *AP*



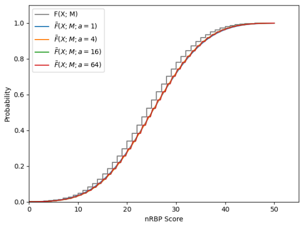
(c) *nRBP*



(d) *nDCG*

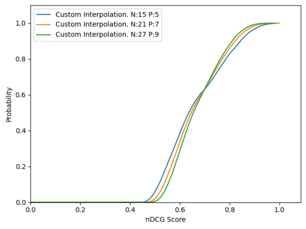


(e) *AP*

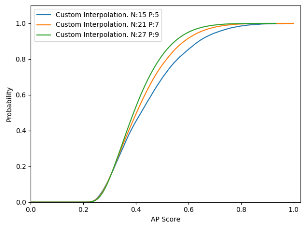


(f) *nRBP*

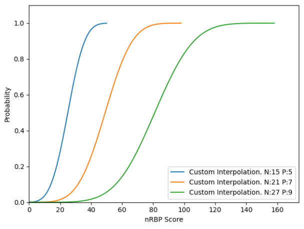
Figure 3.5: Cumulative distribution and approximated distributions (See algorithm 2) of *nDCG*, *AP*, and *nRBP* for two instances $X_{(8,4)}$ (top) and $X_{(15,5)}$ (bottom) for gain $a \in \{1,4,16,64\}$.



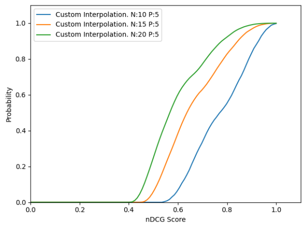
(a) *nDCG*



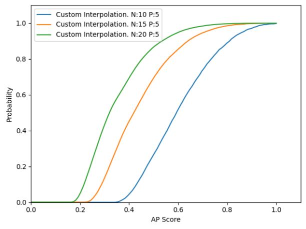
(b) *AP*



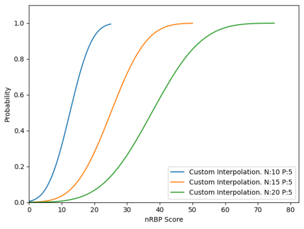
(c) *nRBP*



(d) *nDCG*



(e) *AP*



(f) *nRBP*

Figure 3.6: Approximated distributions of *nDCG*, *AP*, and *nRBP* for three instances with $NSR = 2$ (top) and three instances with different $NSR \in \{1,2,3\}$ (bottom). Using algorithm 2 and Equation 3.13.

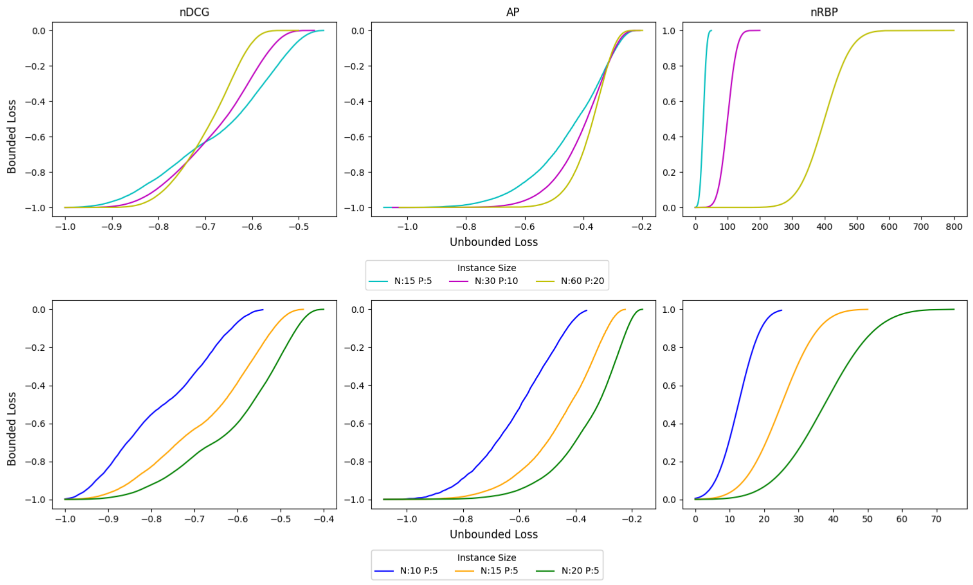


Figure 3.7: Analyzing the impact of `distribution based bounding` on the loss score for three instances with $NSR = 2$ (top) and three instances with $NSR \in \{1,2,3\}$ (bottom), by loss function (columns).

3.4.1. IMPACT OF BOUNDING ON LOSS VALUES

To analyze the effect of the bounding methods on the produced loss scores, Figure 3.7 shows the bounded loss value, as a function of the unbounded loss values for three instances with $NSR = 2$ (top) and three instances with $NSR \in \{1,2,3\}$ (bottom), by loss function (columns). To align the analysis of the impact of metric bounding on $nDCG$ and AP with that of $nRBP$, in this subsection we consider the additive inverse of $nDCG$ and AP as loss function. As a result, the goal of optimization is to minimize the $nDCG$ loss, AP loss and $nRBP$ loss.

Similar to `min-max bounding`, `distribution based bounding` ensures constant upper and lower bounds across instances, but is in contrast to the previous bounding methods not a linear mapping. As a result of the consistent bounds across topics, the resulting loss should be insensitive to the number of (relevant) items in an instance. We analyze the impact of `distribution based bounding` in chapter 5.

4
Experimental Setup

The goal of this thesis is to assess the impact of query/user-wise metric bounding on the relative performance of learning to rank models. First, we explain the selection of models and datasets. Then, we describe the experimental protocol for our experiment.

4.1. Models

As mentioned in chapter 1, LTR can be introduced from two perspectives, namely Recommendation and IR. We therefore choose to analyze the relative performance of ranking models in both a recommendation and IR setting. Considering recommendation, we follow the practice from other ranking-based recommendation that target direct metric optimization and choose Matrix Factorization as the model [20, 10]. For IR, we follow the approach of Liu et al. [15] and deploy a Neural Network as recommendation model. While more advanced methods for LTR exits, our goal is to assess the relative performance of the models when trained using the bounding methods introduced in chapter 3. A more comprehensive study on the performance of more advanced models is left for future work.

4.1.1. Hyperparameters

We explored the impact of the dimensionality of the latent vectors of Matrix Factorization within the range {16, 32, 64}. In accordance with [20], we found that while a higher dimensionality often results in better ranking accuracy, the embedding size has no impact on the relative performance of the models. We therefore select 32 as the number of latent factors and initialize them randomly in the range $[-0.01, 0.01]$. Finally, the batch size, which defines the number of instances that will be propagated through the network before updating it, is set to 32.

While complex Neural Networks with a high number of hidden layers are capable of achieving high metric scores, the goal of this experiment is to analyze the impact of using different bounding methods and realizing the optimal performance is outside the scope of this thesis. We therefore simply initialize a Neural Network with one fully connected hidden layer. For selecting the number of neurons in the hidden layer, we performed a search in the range {6, 12, 23, 46, 69, 92}. The results showed minimal change in overall performance, we therefore simply selected 46 as the number of neurons in the hidden layer. PyTorch [27] will be used to implement and train the models.

4.2. DATASETS

To assess the performance of the proposed bounding methods, two widely-used datasets are selected. For training the Neural Network, we will use the LETOR 4.0 [28] package. LETOR 4.0 is a package of benchmark datasets for research in learning to rank and includes the `MQ2008` dataset. The dataset contains about 800 queries with labeled documents. Each data point represents a query-document pair and consists of a 3-level relevance judgment and a 46-dimensional feature vector.

The `MovieLens` [29] dataset includes 100,000 5-star ratings to 9,000 movies by 600 users and will be used to train the Matrix Factorization model.

Both datasets contain multi-level ratings which need to be binarized before they can be used as training and testing data. In the `MQ2008` dataset, items with the highest relevance rating are considered as relevant. For the `MovieLens` dataset, we consider items with ratings of 4 and higher as relevant. The remaining items, including non-rated items, are taken as non relevant. In addition, we filter out users with less than 25 relevant items in the `MovieLens` dataset, because the lack of data might lead to unreliable performance measurements [20]. As using the same strategy on the `MQ2008` dataset would requires us to remove over half the data, we do not filter out any queries.

4.3. PROTOCOL

To avoid problems like over-fitting and selection bias, cross validation is applied. The 5-fold partitions included in the `MQ2008` dataset will be used for training and testing. For the `MovieLens` dataset, we take a similar approach and randomly split the data in to 5 splits, stratified by user. As we have a minimum of 25 relevant items per user, each user will have at least 5 relevant items in the test set, and 20 in the training set. To speed up the training process, not all irrelevant items will be included in the training and testing set. Instead, irrelevant items will be sampled with a fixed ratio with respect to the relevant items for a user. We denote this ratio as the negative sample ratio (NSR) and create three datasets with a NSR in the range $\{1, 2, 3\}$.

Preliminary research on the impact of the optimizer has been done and while Stochastic Gradient Decent often results in better performance, Adam [30] will be used to optimize the models, as it is computationally efficient, has no impact on the relative performance and requires little tuning for the learning rate [31].

As mentioned in section 3.4, the distribution based bounding method relies on pre-calculated distributions. Both the `MQ2008` and `MovieLens` dataset contain instances with more than 100 items. As a result, computing the metric score for each of the 100! possible orderings becomes infeasible. Therefore, the distributions will be approximated by means of sampling. For each instance, an approximated distribution is created by calculating the metric score for 300.000 random permutations. For the Expectation-Based bounding method, which relies on the expectation of metrics, we take a similar approach. To increase efficiency, we avoid computing the expectation during training and calculate the expectation during preprocessing.

5 RESULTS

In this chapter, we compare the effectiveness of the four bounding methods introduced in chapter 3 when applied to three widely-used listwise loss functions. More specifically, we use the performance of the models trained using the unbounded $nDCG$, AP and $nRBP$ as a baseline and analyze the difference in performance when using one of the proposed bounding methods.

In section 5.1, we analyze the difference in overall performance between models trained with bounded and unbounded losses. Then, the impact of the proposed bounding methods on the $nDCG@k$ score for different values of k is investigated. In section 5.2, we analyze and compare the distributions of scores produced by models optimized for bounded and unbounded losses. Finally, in section 5.3 and section 5.4 we investigate the effect of metric bounding on the recommendation performance on instance level.

5.1. EFFECT OF METRIC BOUNDING ON OVERALL PERFORMANCE

Table 5.1 shows the performance of the proposed bounding methods for each `loss` and `dataset`. We evaluated the models by using the same metric as used for optimization. For models optimized for $nRBP$, we choose to report the $nRBP$ .95 score, as it is best correlated with other metrics [20, 32, 33]. We also note that the reported relative results are representative for both $nRBP$ 0.8 and $nRBP$ .9.

While optimizing the bounded losses often results in similar performance across all losses on all datasets, several observations can be made. First, in the `MovieLens` dataset, a clear negative correlation exists between the negative sampling ratio and the performance of the models. This does not necessarily mean the models trained with a higher NSR are worse [20]. As the test and training sets follow the same distribution, models trained with a higher NSR are evaluated with test sets containing more non-relevant items, which makes the ranking task harder.

Second, the performance on the `MQ2008` dataset is significantly lower when compared to the `MovieLens` dataset. This indicates a difference in ranking difficulty between the two datasets, which might be a direct consequence of the difference in average ratio of relevant to non relevant items between the two datasets.

Finally, we note that applying `Distribution Based Bounding` to *nDCG* leads to a significant decrease in performance in the `MovieLens` dataset. We discuss this in more detail in section 5.5.

Table 5.1: Average performance for each `loss` and `bounding method`. Final row denotes the evaluation metric. *Italic* scores indicate an increase in performance with respect to no bounding while the best score per `loss` is highlighted in **bold**.

`MovieLens`		Score			
		NSR			
`loss`	`bounding method`	1	2	3	
nDCG	No	0.9659	0.9466	0.9294	*nDCG*
nDCG	Min-Max	0.9657	0.9465	0.9293	
nDCG	Expectation Based	0.9654	0.9463	0.9289	
nDCG	Distribution Based	0.8791	0.8534	0.8346	
nDCG	Expectation-Max	**0.9662**	**0.9475**	**0.9309**	
AP	No	0.8960	0.8448	**0.8051**	*AP*
AP	Min-Max	0.8958	0.8448	0.8050	
AP	Expectation Based	0.8960	*0.8450*	0.8044	
AP	Distribution Based	**0.9010**	0.8438	0.7965	
AP	Expectation-Max	0.8958	**0.8453**	0.8046	
nRBP	No	0.9349	0.9046	0.8749	*RBP*.95
nRBP	Min-Max	**0.9473**	**0.9158**	*0.8848*	
nRBP	Expectation Based	*0.9471*	*0.9154*	**0.8854**	
nRBP	Distribution Based	*0.9424*	*0.9104*	*0.8777*	
nRBP	Expectation-Max	*0.9471*	*0.9157*	*0.8846*	

(a) `MovieLens` datasets with $NSR \in \{1,2,3\}$.

`MQ2008`			
`loss`	`bounding method`	Score	
nDCG	No	0.7940	*nDCG*
nDCG	Min-Max	0.7932	
nDCG	Expectation Based	0.7940	
nDCG	Distribution Based	*0.7945*	
nDCG	Expectation-Max	**0.7963**	
AP	No	0.6687	*AP*
AP	Min-Max	*0.6705*	
AP	Expectation Based	0.6640	
AP	Distribution Based	**0.6726**	
AP	Expectation-Max	0.6679	
nRBP	No	0.7140	*RBP*.95
nRBP	Min-Max	*0.7173*	
nRBP	Expectation Based	**0.7181**	
nRBP	Distribution Based	0.7139	
nRBP	Expectation-Max	**0.7181**	

(b) `MQ2008` dataset.

We will in addition consider *nDCG@k* as a method to quantify the performance of the models. Figure 5.1 shows the *nDCG@k* scores as a function of k for each `bounding method`, by `loss` (rows) and `dataset` (columns).

We first note that while there exist a negative correlation between k and the *nDCG@k* score in the `MovieLens` datasets, a clear positive correlation between k and the *nDCG@k* can be observed for the `MQ2008` dataset. An analysis on why we observe a different correlation in the two datasets is left for future work and discussed in more detail in chapter 6.

Looking at the performance of the bounding methods, we can again make several observations. First, we observe minimal changes in performance when optimizing the bounded variants of *nDCG*. This observation supports the previous findings that optimizing bounded variants of *nDCG* does not result in an increase in performance.

Second, while optimizing the bounded variants of *AP* did not lead to a clear increase in overall performance, we do observe an increase in *nDCG@k* score when deploying `distribution based` bounding. Interestingly, while applying `distribution based` bounding resulted in a decrease in average performance in the `MovieLens` dataset with a negative sampling ratio of 3, a slight but consistent increase in performance can be observed when considering *nDCG@k* for $k \leq 10$. In the `MQ2008` dataset, we observe that optimizing a bounded variant of *AP* often leads to an increase in *nDCG@k*.

Finally, we observe that models optimized for bounded *nRBP* losses outperform the unbounded *nRBP* in terms of *nDCG@k* in all datasets, except the `MovieLens` dataset with a negative sampling ratio of 3, where an increase can be observed for $k \geq 3$. This observation, combined with the finding that optimizing bounded variants of *nRBP* leads to similar overall performance, suggests that applying user/query-wise metric bounding to *nRBP* loss can increase recommendation utility.

In the next sections, we analyze and compare the performance on instance-level, to further analyze the impact of applying user/query-wise metric bounding in LTR.

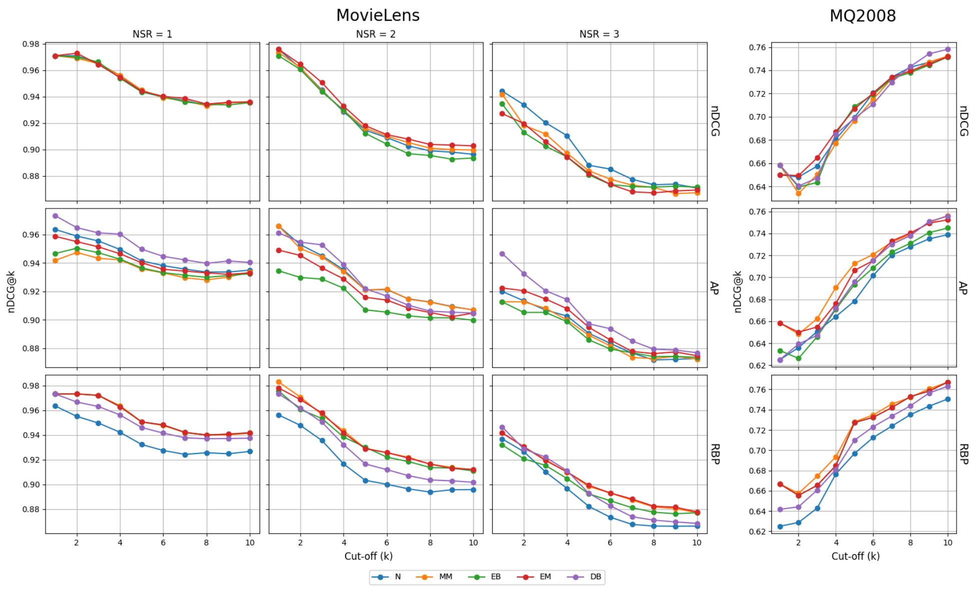


Figure 5.1: *nDCG@k* scores as a function of *k* for each `bounding method`, by `loss base` (rows) and `dataset` (columns).

5.2. EFFECT OF METRIC BOUNDING ON INDIVIDUAL USER/QUERY PERFORMANCE

The analysis in the previous sections showed that optimizing bounded losses may increases the overall performance, but only marginally. In addition, we observed that some bounding methods consistently increase the recommendation performance in terms of *nDCG@k*.

A big drawback of evaluating a system using the average *nDCG*, *AP* or *nRBP* is that the performance of a system is assessed using a single value and the recommendation utility for individual users or specific groups of users cannot be analyzed. Especially when comparing systems, a more in-depth analysis of the performance for different users or groups of users might be of relevance.

In this section, we analyze the performance of the bounding methods by comparing the distribution of scores produces by models optimized for bounded and unbounded losses. This gives greater insights into the performance of the model on instance-level.

Figure 5.2, Figure 5.3 and Figure 5.4 show the comparison between the distribution of scores produced by the models optimized for bounded and unbounded losses for each `loss`, by `bounding method` (rows) and `dataset` (columns). The x-axis represents the instance score, the y-axis shows the fraction of instances that achieve a $score \leq x$.

As the plot might be hard to interpret, score intervals in which the fraction of instances that achieve those scores is lower for the model trained on the bounded loss, are highlighted in gray. In the optimal case, gray areas appear on the left side of the plot, indicating low scores are less likely.

We can make several interesting observations. First, we observe that applying `expectation-max bounding` to *nDCG* results in an slight increase in the fraction of instances that achieve the highest scores. This apparent increase in performance is, however, at the cost of an increase in the fraction of instances that achieve the lowest scores. We further observe minimal changes in the distribution of scores when optimizing a bounded variant of *nDCG*. This supports the findings in previous section that optimizing a bounded variant of *nDCG* does not lead to an increase in performance.

Second, applying `distribution based bounding` to *AP* results in a decrease in the fraction of instances that achieve the lowest scores, not at the cost of instances that achieve high scores. The impact of `distribution based bounding` does seem to decrease with an increase in NSR. This observation supports the findings in the previous sections that `distribution based bounding` works best in datasets with a low NSR. We further observe that optimizing other bounded variants of *AP* do not improve the distribution of scores.

Finally, the results in the previous sections showed that optimizing a bounded variant of *nRBP* could increase the recommendation utility. This observation is supported by Figure 5.4, where we observe a consistent decrease of the fraction of instances that achieve low scores in the `MovieLens` dataset. Also, in line with previous findings, we observe a decrease in the effectiveness of `distribution based bounding` for higher negative sampling ratios.

In the next sections, we continue to analyze the performance of the models optimized for bounded and unbounded losses and aim to identify which instances benefit most from metric bounding.

5.3. WHERE DOES METRIC BOUNDING HAVE THE LARGEST EFFECT?

In this section, we aim to identify which instances benefit from metric bounding by analyzing and comparing the performance on instance level. As the models are trained and tested on the same dataset, we can compare scores on instance level by plotting the performance of the model trained with a bounded loss as a function of the performance of the model trained with an unbounded loss. Figure 5.5, Figure 5.6 and Figure 5.7 show the instance scores as a function of the performance of the model optimized for the unbounded loss for each `loss`, by `bounding method` (rows) and `dataset` (columns).

For all losses, we observe a clear and strong correlation between the instance scores achieved by the models optimized for a bounded loss and the scores achieved by the models optimized for a unbounded loss. This indicates that high scoring instances remain to achieve high scores and any change in performance in not correlated with the scores achieved by model trained on unbounded losses. However, in the `MovieLens` dataset with a *NSR* of 1, we observe that metric bounding may increase the performance for the lowest scoring instances.

In addition, in the `MovieLens` dataset we observe that the variance in score increases

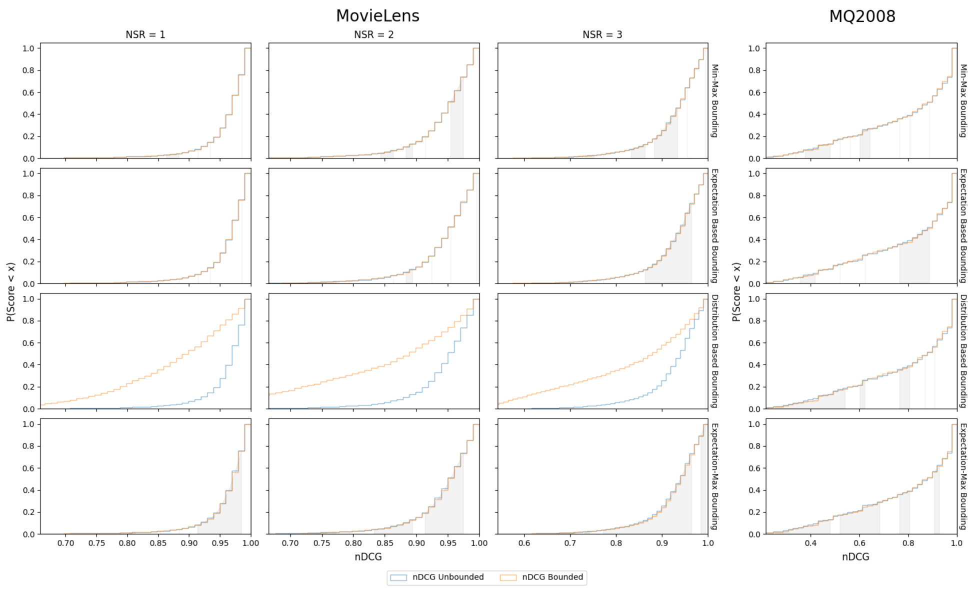


Figure 5.2: Comparing the distributions of scores produced by models optimized on bounded and unbounded *nDCG*, by `bounding method` (rows) and `dataset` (columns).

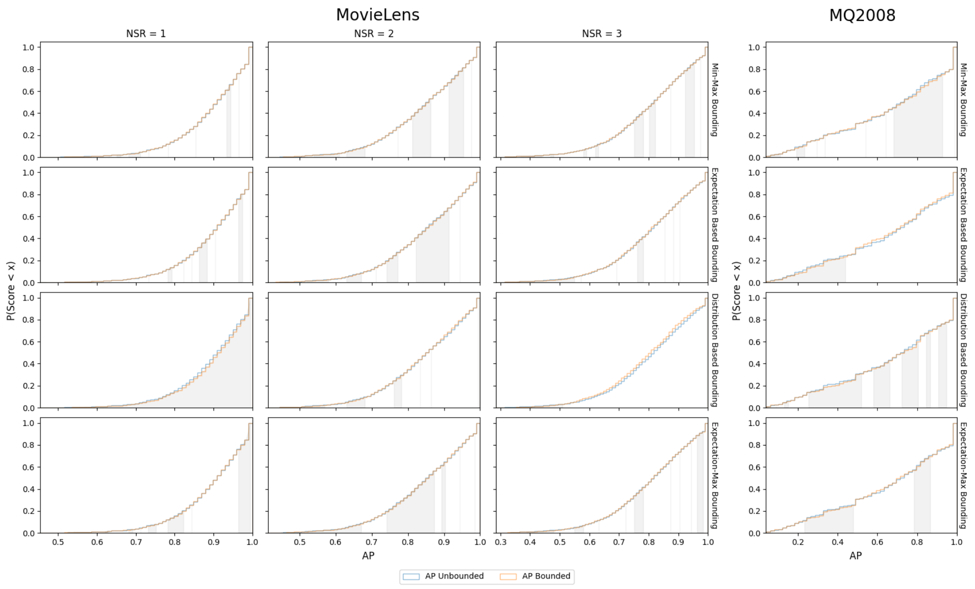


Figure 5.3: Comparing the distributions of scores produced by models optimized on bounded and unbounded *AP*, by `bounding method` (rows) and `dataset` (columns).

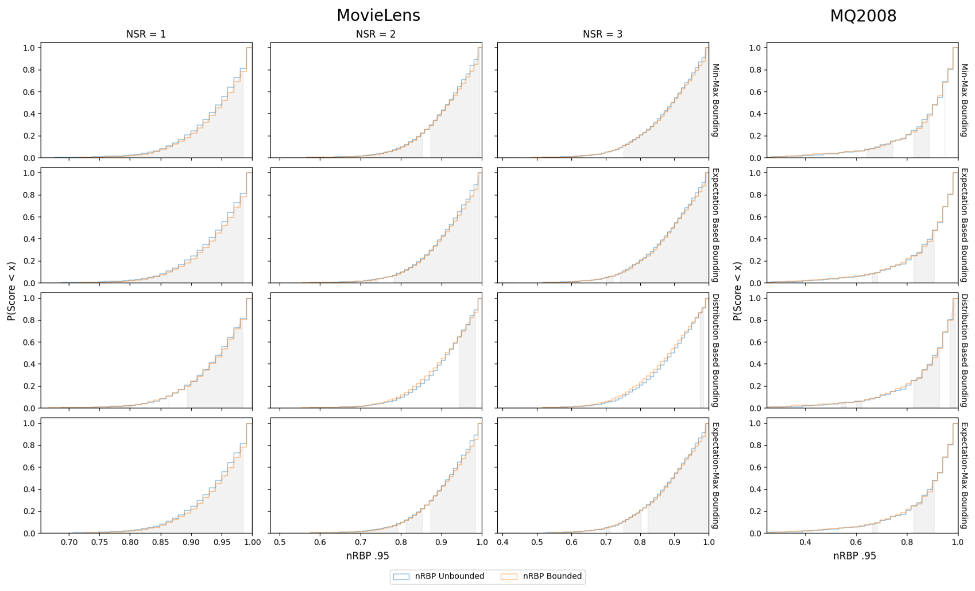


Figure 5.4: Comparing the distributions of scores produced by models optimized on bounded and unbounded *nRBP*, by `bounding method` (rows) and `dataset` (columns).

with the *NSR*. This observation holds true for all losses and indicates that the proposed bounding methods have greater impact when used on datasets with a higher *NSR*. Interestingly, this observation challenges the finding in the previous section that the impact of `distribution based bounding` decreases with a higher NSR.

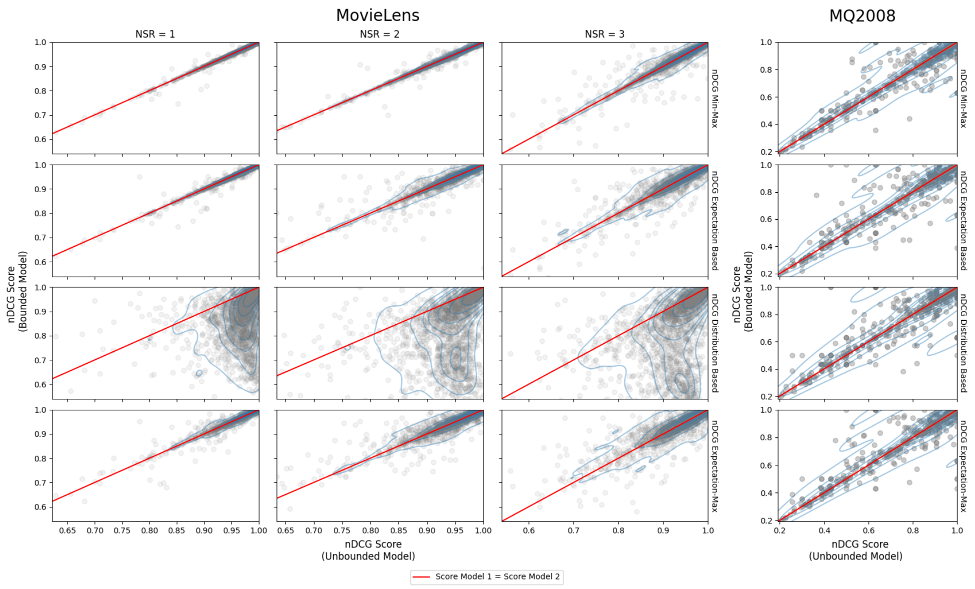


Figure 5.5: Instance scores when optimizing bounded variants of *nDCG*, as a function of the instance score when optimizing for unbounded *nDCG*.

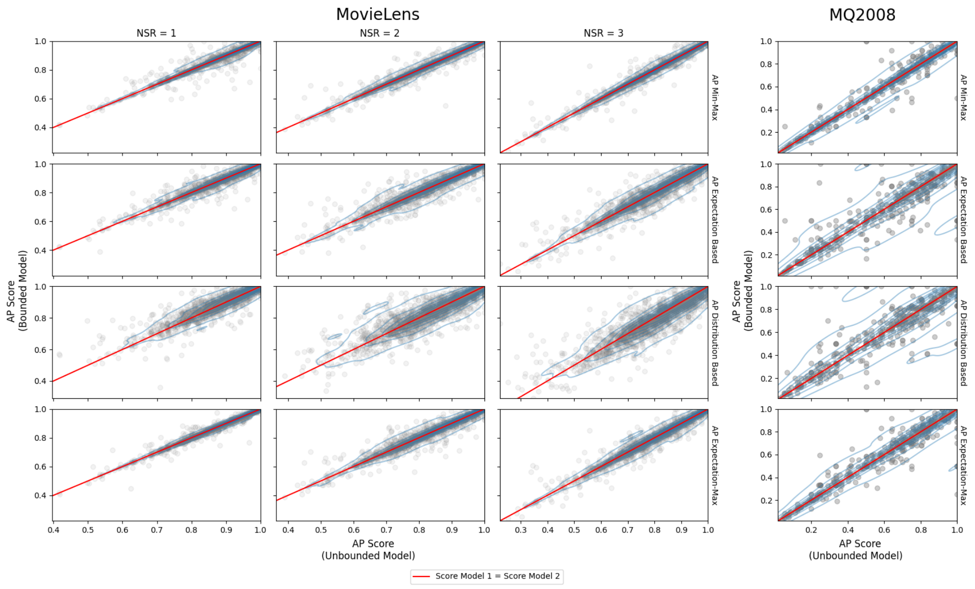


Figure 5.6: Instance scores when optimizing bounded variants of *AP*, as a function of the instance score when optimizing for unbounded *AP*.

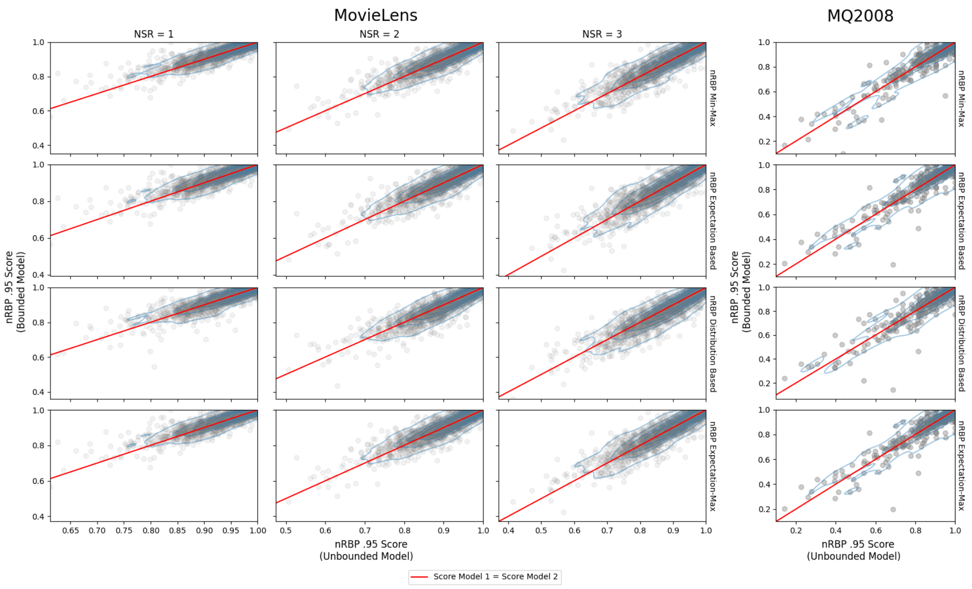


Figure 5.7: Instance scores when optimizing bounded variants of $nRBP$, as a function of the instance score when optimizing for unbounded $nRBP$.

5.4. WHO BENEFITS FROM METRIC BOUNDING?

The analysis in the previous sections indicated that optimizing a bounded loss can increase the recommendation utility. The proposed bounding methods utilize some notion of instance-difficulty to create difficulty-aware losses. As a result, difficult instances are weighted more heavily during training, which could increase the recommendation utility for these difficult instances, possibly at the cost of the utility for easier instances. As shown in chapter 3, the bounding methods heavily depend on the number of relevant and non relevant items in an instance. We therefore investigate whether we observe a correlation between the performance difference as a function of the number of relevant items in an instance. Figure 5.8, Figure 5.9 and Figure 5.10 show the performance difference between optimizing the bounded and unbounded loss as a function of the number of relevant items in an instance. In addition, we provided a kernel density estimate, which is a non-parametric fit to the data.

For models trained on bounded variants of $nDCG$ and AP, we do not observe a clear correlation between the performance change and number of relevant items in an instance. This means that instances with a high number of relevant items do not benefit from the proposed bounding methods.

In contrast, in the `MovieLens` dataset with a higher NSR, we observe that while optimizing a bounded variant of $nRBP$ improves the performance for all instances, instances with a higher number of relevant items may benefit more. This increase in performance for instances with a higher number of relevant items is not at the cost of instances with a lower number of relevant items.

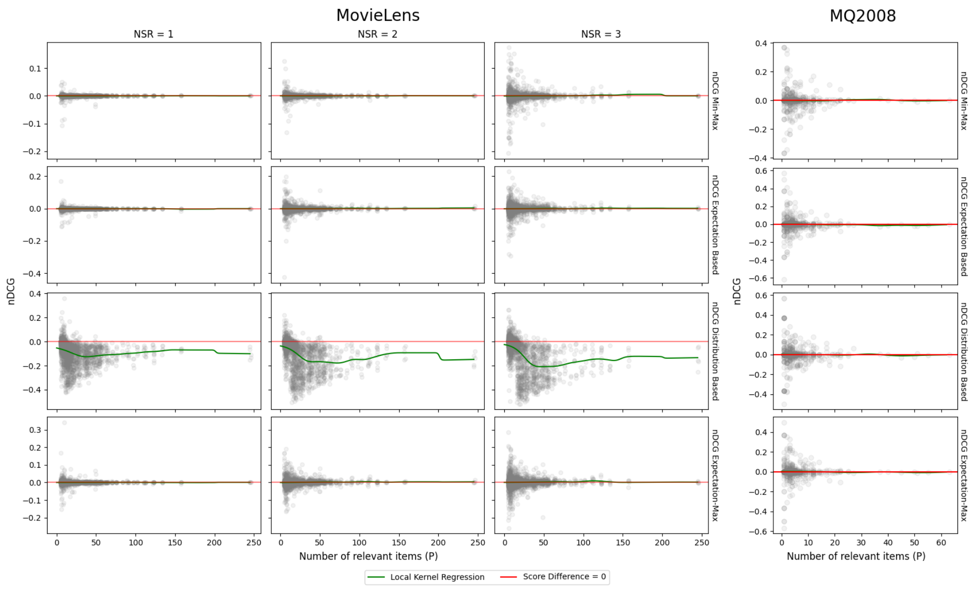


Figure 5.8: Score difference between optimizing the bounded *nDCG* and unbounded *nDCG* (higher is better), as a function of the number of relevant items in an instance, by dataset (columns) and bounding method (rows).

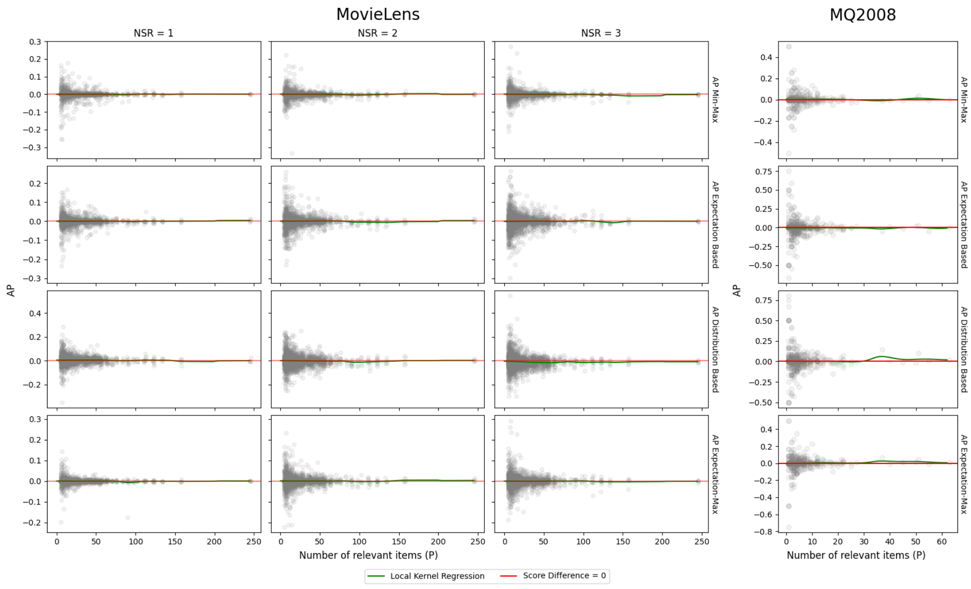


Figure 5.9: Score difference between optimizing the bounded *AP* and unbounded *AP* (higher is better), as a function of the number of relevant items in an instance, by dataset (columns) and bounding method (rows).

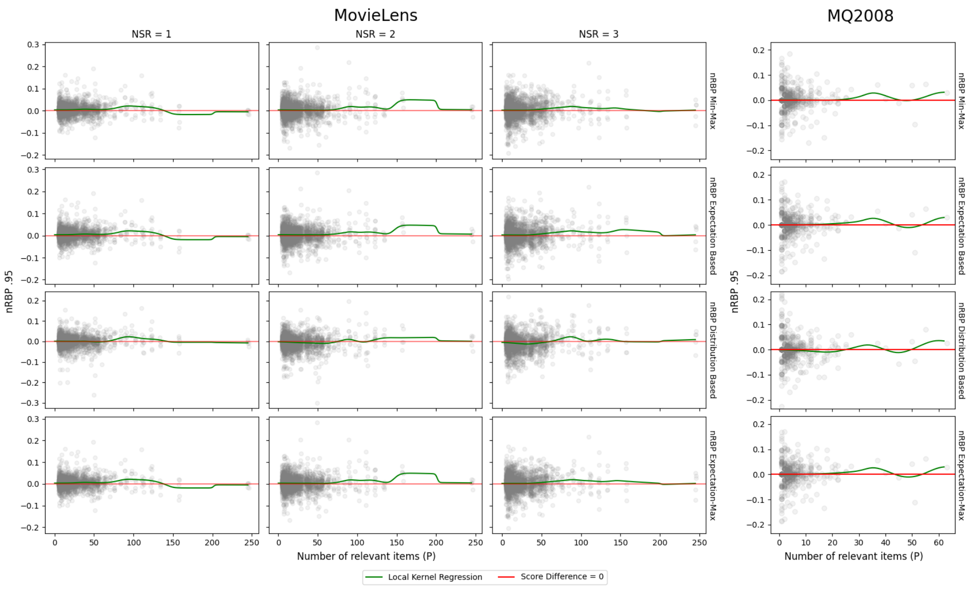


Figure 5.10: Score difference between optimizing the bounded *nRBP* and unbounded *nRBP* (higher is better), as a function of the number of relevant items in an instance, by dataset (columns) and bounding method (rows).

5.5. SUMMARY

To summarize, the empirical evidence in this chapter showed that optimizing a bounded loss is often no less effective than optimizing unbounded losses. In addition, we observed a marginal but consistent increase in average performance when optimizing a bounded variant of *nRBP*. Interestingly, additional analysis on the impact of metric bounding on the performance of ranking models in terms of *nDCG@k* showed an increase in performance for both *AP* and *nRBP*. These observations are further supported by a comparison of the distributions of scores produced by models optimized for bounded and unbounded losses.

Furthermore, the obtained results show a strong correlation between the instance scores achieved by the models optimized for a bounded and unbounded losses. Interestingly, we observed that optimizing a bounded variant of *nRBP* resulted in an increase in performance for the lowest scoring instances. In addition, we observed that metric bounding may benefit instances with a higher number of relevant items more.

Finally, the results in this chapter showed that applying `distribution based bounding` to *nDCG* leads to a significant decrease in performance in the `MovieLens` dataset. In contrast, `distribution based bounding` showed promising results in other datasets, or when applied to other losses. Additional analysis on the performance in different `MovieLens` datasets showed similar ranking performance. In addition, an analysis on the approximated distributions did not result in new insights on this decrease in performance. Further analysis is outside the scope of this thesis and is left for future work.

6 CONCLUSION

Learning to Rank is the application of Machine Learning in order to create and optimize ranking functions. Most learning to rank methods follow a listwise approach and optimize a listwise loss function which closely resembles the same metric used in the evaluation. The optimization of such ranking functions is an iterative process which utilizes a loss function and its derivatives with respect to the models parameters to decrease the average loss on the current batch. Popular listwise loss functions such as $nDCG$, AP and $nRBP$ do not have consistent bounds across topics and do not account for instance-difficulty. As a result, the loss score does not solely reflect the performance of the model, but also depend on the instance properties. During training, each instance is assumed to be equally informative, while in reality, this informativeness might depend on the difficulty of the instance.

In this thesis, we proposed four bounding methods which utilize some notion of instance-difficulty to produce difficulty-aware losses and showed the impact on the produced loss scores. We applied the four proposed bounding methods to three popular listwise loss functions, namely $nDCG$, AP and $nRBP$ and analyzed the impact on the recommendation effectiveness of two ranking models. Experimental results based on two datasets showed that, in most cases, optimizing a bounded loss function results in a consistent but marginal increase in overall performance. More interestingly, we showed that optimizing a bounded variant of $nRBP$ and AP may increase the $nDCG@k$, increasing the recommendation utility.

To further investigate the impact of user/query-wise metric bounding on the performance of the ranking models, we analyzed the performance on instance level. We showed that user/query-wise metric bounding can increase the recommendation performance for instances with a higher number of relevant items, without a negative impact on the performance for instances with a lower number of relevant items.

6.1. RECOMMENDATIONS

The empirical results suggest that, in most cases, optimizing a bounded loss leads to a consistent but marginal improvement in average performance of the ranking functions. Furthermore, we observed that optimizing a bounded loss can lead to a consistent increase in the $nDCG@k$ score for $k \leq 10$, which makes the optimization of a bounded loss interesting for applications where only a small list of recommendations is provided to a user. In addition, the results suggest that optimizing a bounded variant of $nRBP$ can benefit instances with a higher number of relevant items, without a negative effect on the average performance of other instances. This observation of the proposed bounding methods further motivates the use of user/query-wise metric bounding.

Finally, while optimizing a bounded variant of $nRBP$ generally outperforms optimizing bare $nRBP$, the best bounding method for $nDCG$ and AP seems more dependent on the dataset.

6.2. FUTURE WORK

The research in this thesis can be extended on the following aspects. First, we can extend the experiment and analyze the impact of user/query-wise metric bounding in different datasets. In addition, we can extend the experiment with more complex ranking models. This allows us to analyze whether the observations made in this thesis hold true in different datasets and for more complex ranking functions.

Secondly, `distribution based bounding` currently relies on distributions generated from sampled data. These distributions are generated by calculating the metric score of a fixed amount of random permutations of an instance. As a consequence, each order is taken as equally likely, while in reality most ranking functions easily outperform a random ranking. We would like to analyze the impact of `distribution based bounding` when utilizing a distribution which favors ranking relevant items higher. This makes for a more realistic distribution of scores and in addition increases the resolution of the distribution at higher scores.

Third, the results showed a negative correlation between k and the $nDCG@k$ score in the `MovieLens` dataset. In contrast, in the `MQ2008` dataset we observed a positive correlation. It would be interesting to analyze what is the reason behind this observation. We suspect this to be either a consequence of the used ranking function or dataset, but leave the analysis for future work.

Furthermore, in this thesis, we focused on the direct optimization of IR metrics and thus evaluated the ranking functions using the same metric as used in the optimization process. However, other research challenges the assumption that optimizing the same metric used in the evaluation leads to the best performance [20]. It would therefore be interesting to analyze to which extent the observations made in this thesis hold true when evaluating and optimizing a ranking function using different metrics.

Finally, the current training and test sets follow the same distribution of relevant to non-relevant items. It would be interesting to analyze to which extent the ranking models can benefit from being trained on training sets with a higher negative sampling ratio when optimizing a bounded loss. To analyze this, we would increase the NSR in the training set, without increasing the NSR in the test set.

REFERENCES

[1] Hang Li. A short introduction to learning to rank. *IEICE Transactions on Information and Systems*, E94-D(10):1854–1862, 2011.

[2] David Cossock and Tong Zhang. Subset ranking using regression. volume 4005, pages 605–619, 06 2006.

[3] Ping Li, Qiang Wu, and Christopher Burges. Mcrank: Learning to rank using multiple classification and gradient boosting. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2008.

[4] Koby Crammer and Yoram Singer. Pranking with ranking. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, page 641–647, Cambridge, MA, USA, 2001. MIT Press.

[5] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4(null):933–969, dec 2003.

[6] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. *Proceedings of the 22nd international conference on Machine learning - ICML 05*, 2005.

[7] Christopher Burges, Robert Ragno, and Quoc Le. Learning to rank with nonsmooth cost functions. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2007.

[8] Christopher Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11, 01 2010.

[9] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: From pairwise approach to listwise approach. *Proceedings of the 24th international conference on Machine learning - ICML 07*, 2007.

[10] Guang-He Lee and Shou-De Lin. Lambdamf: Learning nonsmooth ranking functions in matrix factorization using lambda. In *2015 IEEE International Conference on Data Mining*, pages 823–828, 2015.

[11] Junjie Liang, Jinlong Hu, Shoubin Dong, and Vasant G. Honavar. Top-n-rank: A scalable list-wise ranking method for recommender systems. *CoRR*, abs/1812.04109, 2018.

[12] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[13] Gintare Karolina Dziugaite and Daniel M. Roy. Neural network matrix factorization. *CoRR*, abs/1511.06443, 2015.

[14] Jun Xu and Hang Li. Adarank. *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR 07*, 2007.

[15] Tie-Yan Liu. The listwise approach. *Learning to Rank for Information Retrieval*, page 71–88, 2011.

[16] Yue Shi, Martha Larson, and Alan Hanjalic. List-wise learning to rank with matrix factorization for collaborative filtering. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, page 269–272, New York, NY, USA, 2010. Association for Computing Machinery.

[17] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. Softrank. *Proceedings of the international conference on Web search and web data mining - WSDM 08*, 2008.

6

[18] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

[19] Alistair Moffat and Justin Zobel. Rank-biased precision for measurement of retrieval effectiveness. *ACM Trans. Inf. Syst.*, 27(1), dec 2008.

[20] Roger Zhe Li, Julián Urbano, and Alan Hanjalic. New insights into metric optimization for ranking-based recommendation. *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021.

[21] Mu Zhu. Recall, precision and average precision. *Department of Statistics and Actuarial Science*, 2, 2004.

[22] Tao Qin, Xu-Dong Zhang, Ming-Feng Tsai, De-Sheng Wang, Tie-Yan Liu, and Hang Li. Query-level loss functions for information retrieval. *Information Processing & Management*, 44(2):838–855, 2008.

[23] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Alan Hanjalic, and Nuria Oliver. Tfmap. *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval - SIGIR 12*, 2012.

[24] Lukas Gienapp, Benno Stein, Matthias Hagen, and Martin Potthast. Estimating topic difficulty using normalized discounted cumulated gain. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020.

[25] Yves Bestgen. Exact expected average precision of the random baseline for system evaluation. *The Prague Bulletin of Mathematical Linguistics*, 103(1):131–138, 2015.

[26] Benjamin A. Carterette. Low-cost and robust evaluation of information retrieval systems. *ACM SIGIR Forum*, 42(2):104–104, 2008.

[27] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zach DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[28] Tao Qin and Tie-Yan Liu. Introducing letor 4.0 datasets, 2013.

[29] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), December 2015.

[30] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[31] Derya Soydaner. A comparison of optimization algorithms for deep learning. *International Journal of Pattern Recognition and Artificial Intelligence*, 34(13):2052013, 2020.

[32] Tetsuya Sakai and Noriko Kando. On information retrieval metrics designed for evaluation with incomplete relevance assessments. *Information Retrieval*, 11(5):447–470, 2008.

[33] Alistair Moffat, Falk Scholer, and Paul Thomas. Models and metrics. *Proceedings of the Seventeenth Australasian Document Computing Symposium on - ADCS '12*, 2012.