University of Amsterdam

MSc Artificial Intelligence
Master Thesis

Relevance-Based Click Models and Adversarial Training

---

by
Bram van den Akker
10434100

Monday 5th August, 2019

Supervisors:
Dr. Ilya Markov
Harrie Oosterhuis MSc

Assessor:
Prof. Dr. Hinda Haned

Faculteit der Natuurkunde, Wiskunde en Informatica

Abstract

Being able to predict the user click behavior on search results is essential for both optimizing and evaluating ranking algorithms. This click behavior is often estimated using click models, which are optimized to predict the click probability based on existing click logs. However, these click models can only predict clicks on documents that have already appeared in existing click logs, which limits their applications for unseen documents. To overcome this limitation, many researchers create hand-picked click probabilities based on the editorial relevance labels of document-query pairs. In this work, we demonstrate the performance of various existing click models when training them on editorial relevance. Additionally, we provide a detailed analysis of the different kinds of click behaviors that are learned by the click models to explain the differences in performance. Finally, we also introduce a novel relevance-based click model architecture based on Generative Adversarial Networks (GAN) and show that it can significantly outperform existing click models.

Acknowledgements

I would like to thank Ilya Markov and Harrie Oosterhuis for supervising me during these past months. You're fruitful discussions and feedback have helped me tremendously level up my research skills. My gratitude goes to Hinda Haned for taking the time to be the assessor on my defense committee.

Additionally, I would like to thank everyone who supported me during these last few months. Thanks to Haitam Ben Yahia, Dana Kianfar, and Suzanne van der Tweel for taking the time to proofread and provide feedback on my work. Also, I want to thank Nuno Mota, Victor Milewski, and Alexandra Arkut for always be ready to provide feedback, have insightful discussions, and keeping up the motivation for my thesis every day of the week. Additionally, I want to thank Jorn Peters, Emiel Hoogeboom, Tycho van der Ouderaa, and Haitam Ben Yahia for sharing your extensive knowledge of various key topics that allowed me to verify and kickstart new ideas. Thanks to my parents for your continuous support that has made it possible to be where I am at today.

Finally, I want to give a special thanks to my girlfriend Sharon Gieske for not only supporting and encouraging me throughout the whole thesis project, but also for providing me with endless feedback on my work and proofreading every single page of this thesis multiple times.

Contents

Chapter 1

Introduction

The field of Information Retrieval consists of numerous methods to present a user with the most relevant information for a given context. A well-known Information Retrieval application is that of the search engine, where a user is presented with a list of ranked documents based on their relevance with respect to a user-provided query. Methods that determine the relevance of each document can be based on various aspects, such as the document-query similarity (e.g. BM-25, TF-IDF) and the importance of the document (e.g. PageRank [36], Visual Features [1]).

Creating ranking methods is a complex task, as the objective of the user (e.g. gathering inspiration) might not be identical to the direct objective of the ranking functions (e.g. determining similarity). To optimize and verify the ranking, the document preferences from actual users are needed. However, collecting explicit document preferences from users is time-consuming and expensive. Moreover, user intents can vary from seemingly identical queries, which makes creating a single source of truth complicated. A relatively cheap approach to get implicit feedback about which documents are preferred is by recording the clicks made by various users. In the rest of this thesis, we will refer to these recorded clicks as click data.

However, using click data to optimize ranking algorithms is non-trivial. User clicks are subject to various biases such as a tendency to click higher-ranked documents (position bias) and documents from well-known sources (trust bias). Over the past years, various models [20] have been developed to model click data with minimal bias. Click models attempt to estimate the click probability for each position in the query ranking. To optimize and evaluate click models, we usually look at both the log-likelihood (Section 4.2) and perplexity (Section 4.4) for observed click sequences or the models re-ranking capabilities. The trained models can then be used for tasks such as re-ranking query results [8] and evaluating ranking performance [9].

Based on the click patterns and behavior highlighted by work such as Joachims et al. [26], various variations of click models have been developed. Early click models (Section 2.1.2) are modeled using Probabilistic Graphical Models (PGMs), which directly model handcrafted heuristics based on known click behavior. These heuristics represent various indicators of click behavior such as the probability of examining a document or the probability of a document being attractive for a given query. These probabilities are then learned by optimizing the model heuristics using click data. Although these explicit heuristic parameters make the click prediction behavior of these models very interpretable, they also limit the model to the patterns explicitly captured by the parameters. To address this limitation, Borisov et al. [6] recently introduced the Neural Click Model Framework which demonstrated a clear improvement in performance compared to the traditional PGM based click models.

A limitation of current click model research is that the datasets used for these models, such as the Yandex Relevance Prediction Dataset[1], do not contain information about the queries and documents beyond their unique identifier and the click counts captured in their query sessions. This means that the content of the documents and queries is omitted by the publisher. This limits the trained click models to only be used on document-query pairs that have already been observed in the dataset. Although datasets such as TREC Web[2] do contain the document and query content, they lack click log data. At the time of writing, no dataset exists that contains both click data and query-document content, which limits information retrieval

[1] `https://academy.yandex.ru/events/data_analysis/relpred2011/`

[2] `https://trec.nist.gov/data/webmain.html`

researchers in applying click data to their work.

For research topics such as online- and counterfactual learning to rank [19, 27], it is important to understand the probability of a user clicking on a particular document. However, these ranking models require a wide variety of document, query, user, and click features to be optimized. Because there are no datasets available that satisfy these requirements, works such as Oosterhuis et al. [34] and Mehrotra et al. [32] use relevance-based click models as introduced by Hofmann et al. [22]. These relevance-based click models work by hand-picking a click probability for each available relevance label given to the query-document pairs in the dataset. To our knowledge, no study has shown how well the hand-picked probabilities reflect actual user click behavior.

1.1 Thesis Outline

In this work, we introduce a collection of click models that are optimized to predict the click probability based on available relevance labels. By optimizing and evaluating these models on actual user clicks, we can compare their ability to model actual click behavior. Additionally, we analyze the various click probabilities produced by each click model to get a better understanding of the learned click behavior. Finally, we introduce a novel neural click model based on Generative Adversarial Networks (GANs) and compare its click prediction performance to existing click models.

To motivate our work, we set out to answer the following research questions:

RQ_1 Which model has the best performance when using relevance judgments as an input feature?

RQ_2 How can we explain the differences in performance between the different relevance-based click models?

RQ_3 Does training a Neural Click Model with an adversarial loss improve performance?

RQ_4 If any, what is causing the performance differences between a regular and adversarial trained Neural Click Model?

This thesis begins by outlining the necessary background on click models, relevance-based click models, and GANs in Chapter 2. Based on this background, we introduce our relevance-based click model methods in Chapter 3. Details about the dataset, optimization methods, evaluation metrics, and baselines are provided in Chapter 4. The research questions are answered using the results presented in Chapter 5. Finally, we conclude our work together with a view on future work in Chapter 6.

Chapter 2

Background

In this chapter, we introduce the necessary background to support the work in this thesis.

First, Section 2.1 starts with a general history of click models, followed by research on relevance-based click models in Section 2.1.1. After the introduction of relevance-based click models, we go into more details about various existing Probabilistic Graphical Click Models (Section 2.1.2) and Neural Click Models (Section 2.1.3). Finally, in Section 2.2 we highlight various methods of Sequential Adversarial Training (Section 2.2.1), and Generative Adversarial Networks (Section 2.2.2).

2.1 Click Models

Click models are a method of estimating the click behavior of users on search engine result pages. Various works exist where click models are used to improve [8] and evaluate [9] search results or simulate users in offline settings [22, 34].

The work of Joachims et al. [26] was fundamental to many insights that are used in click models for web-search today. The authors combine eye-tracking and click-through data to get a better understanding of user behavior on the search engine result page (SERP). The results show that user clicks are influenced by the actual relevance of the documents, but are biased by the position, the user's trust in the retrieval function and the overall quality of the results.

According to several studies [20, 11], Craswell et al. [13] were likely the first to use the term *click model*. Since then, various click models have been proposed several which are discussed in this section.

In this section we first provide a background on existing relevance-based click model research (Section 2.1.1), followed by various existing click model implementations such as Probablistic Graphical Model (PGM) based click models (Section 2.1.2) and Neural Click Models (Section 2.1.3).

2.1.1 Relevance-Based Click Models

In chapter 1, we mentioned that click models trained on existing click datasets are limited to predicting clicks on documents already seen within the dataset. To overcome this limitation research has used document relevance labels to predict user clicks. The methods of collecting these document relevance labels vary per dataset but are often created by combing crowdsourcing and pooling techniques [40].

Turpin et al. [39] demonstrated that the document relevance reasonably estimates both its probability to be clicked (attractive) and being satisfactory for a user. This notion was used by Hofmann et al. [22] to predict the user clicks based on the document relevance. The authors handpicked probabilities for observing a click $p(c|R)$ and stopping examining subsequent documents $p(s|R)$ based on the relevance label of each document in the dataset. The probabilities are then used as the parameters of a Dependent Click Model (DCM)[21] as described in Section 2.1.2. The authors Hofmann et al. [22] pick probabilities to reflect various types of user click behavior based on the available binary relevance. In Table 2.1, we highlight some of the types of click behavior introduced by Hofmann et al. [22] such as the perfect, navigational, and informational setups.

	$p(c\|R)$	$p(c\|NR)$	$p(s\|R)$	$p(s\|NR)$
perfect	1.0	0.0	0.0	0.0
navigational	0.95	0.05	0.9	0.2
informational	0.9	0.4	0.5	0.1

Table 2.1: The click and stop probablities used by Hofmann et al. [22]. Perfect, navigational, and informational represent how closely the user clicks represent the relevance judgments. $p(c|R)$ and $p(c|NR)$ represent the probability of a click on relevant and non-relevant documents, while $p(s|R)$ and $p(s|NR)$ represent the probability of the user stopping examining subsequent documents after observing a relevance or non-relevant document.

Later work, such as Hofmann et al. [23], often use graded relevance to set the model parameters. In graded relevance, there are more than two possible relevance labels, providing more information about the level of relevance of each document. In both binary and graded relevance, we most commonly see the DCM model being used. However, the parameters associated with the various relevance labels varies widely between researchers.

Work by Chen et al. [10] introduce a derivative of the PGM-based User Browsing Model (UBM) and Dynamic Bayesian Network (DBN) click models (Section 2.1.2) with an additional noise parameter. This noise parameter is a value in $[0 - 1]$ where 0 represents a user that always clicks on relevant documents, while 1 represents a user that clicks completely at random. The principle of the noise parameter is very similar to Hofmann et al. [22] but is learned instead of manually set. The noise parameter is learned in two steps: first, the relevance labels are used to learn a noise parameter, while in step two the models learn a parameter for relevance estimation using the learned noise parameter. The authors show that introducing the noise parameter improves the performance of both the UBM and DBN model.

Although the various works described above do use relevance judgments in their click models, we are unaware of any work that learns a click model from relevance judgments alone.

2.1.2 Probabilistic Graphical Model-Based Click Models

Traditionally, click models are defined by explicitly modeling the dependencies of the features and parameters using a Probabilistic Graphical Model (PGM). The dependencies are based on assumptions about the click behavior of users interacting with the SERPs. After defining these dependencies, the model parameters are learned through either Maximum Likelihood Estimation (MLE) or Expectation-Maximization (EM). In this section, we explain the intuition and definitions of four different PGM-based click models that are often used in click model research.

Click-Through Rate Models

The Click-Through Rate (CTR) models are a collection of click models that explicitly learn the click ratio for a single input feature. Two examples of CTR models are the rank-based model by Dupret and Piwowarski [14], and the document-based model by Craswell et al. [13]. Because CTR models only optimize a single parameter, this can be achieved by using MLE which is relatively easy to implement.

The rank-based model has a single CTR parameter p_r for each rank to estimate the probability of a click C_r at rank r, which makes it both easy to interpret and robust to overfitting.

$$P(C_r = 1) = p_r \tag{2.1}$$

The document-based model has a parameter p_{qd} to estimate the probability of a click C_{qd} for each document-query pair in the training set. Other than the rank-based model, the document-based model can learn click probabilities specifically for each document. Although this allows the model to predict specific click probabilities for each query-document pair, this also makes the model sensitive to overfitting. Moreover, when document-query pairs have not been seen in the training set, the model will not have trained parameters to make a prediction.

$$P(C_{qd} = 1) = p_{qd} \tag{2.2}$$

Dependent Click Model

The Dependent Click Model (DCM) assumes that a user always examines documents from top to bottom until they click. After each click, the user either stops or goes further down the SERP. This behavior is modeled by the probability of a click and the user abandoning the search. These parameters are also known as the attractiveness parameter α_{qd_r} and a continuation parameter λ_r. The continuation parameter estimates the satisfaction S_r of a user about the clicked document. In the DCM model, we assume that a click at each rank C_r is only possible if the document at rank r is both examined E_r and attractive A_r. Moreover, a document is only examined if the previous document as examined and did not satisfy the user. The DCM model can formally be described as shown in Equations 2.3 through 2.9. Additionally, we show a graphical representation of the model in Figure 2.1.

$$C_r = 1 \Leftrightarrow E_r = 1 \text{ and } A_r = 1 \tag{2.3}$$
$$P(A_r = 1) = \alpha_{qd_r} \tag{2.4}$$
$$P(E_1 = 1) = 1 \tag{2.5}$$
$$P(E_r = 1 | E_{r-1} = 0) = 0 \tag{2.6}$$
$$P(S_r = 1 | C_{r-1} = 0) = 0 \tag{2.7}$$
$$P(S_r = 1 | C_{r-1} = 1) = 1 - \lambda_r \tag{2.8}$$
$$P(E_r = 1 | E_{r-1} = 1, S_{r-1} = 0) = 1 \tag{2.9}$$

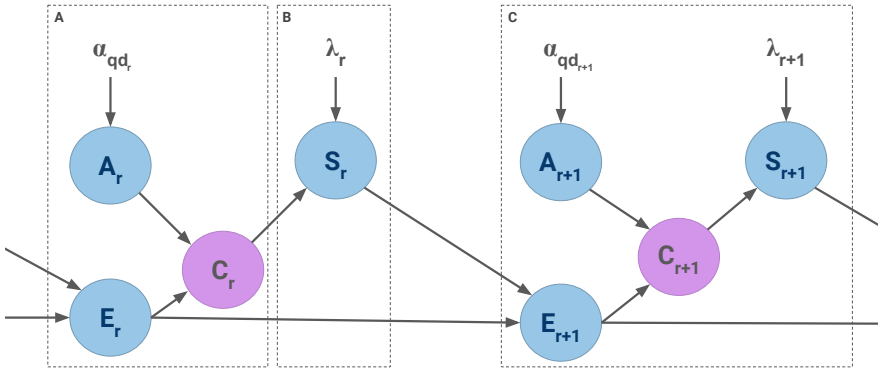


*Figure 2.1: A graphical representation of the DCM model. Area **A** shows how the click probability C_r at rank r is determined by the latent attractiveness A_r and examination E_r variables. A_r is based on the historic attractiveness parameter α_{qd_r} with the document at r, while E_r is based on whether the user continued browsing after examining the previous document. This continuation, as highlighted in area **B**, is a combination of the click probability and the satisfaction S_r modeled by the continuation parameter γ_r for rank r. Area **C** highlights the parameters at r influence the subsequent rank $r + 1$.*

User Browsing Model

The User Browsing Model (UBM) by Dupret and Piwowarski [14] aims to model the relation between the current click probability and the last observed clicks. In order to do so, the UBM model has the same attractiveness α_{qd_r} and continuation $1 - \lambda_r$ parameters as the DCM model. However, rather than having a continuation $1 - \lambda_r$ parameter based just on the current rank r, it also takes the rank of the previously clicked document r' (Equation 2.10) into account. The continuation parameter of the UBM model is therefore referred to as $\gamma_{rr'}$. Similar to the DCM model, we assume that a click at each rank C_r is only possible if the document at rank r is both examined E_r and attractive A_r. The UBM model can be optimized using the EM algorithm and can formally be described as seen in Equation 2.11.

$$r' = max\{k \in \{0, ..., r-1\} : c_k = 1\} \tag{2.10}$$

$$C_r = 1 \Leftrightarrow E_r = 1 \text{ and } A_r = 1 \tag{2.11}$$

$$P(E_r = 1 | C_1 = c_1, ..., C_{r-1} = c_{r-1}) = \gamma_{rr'} \tag{2.12}$$

$$P(A_r = 1) = \alpha_{qd_r} \tag{2.13}$$

$$\tag{2.14}$$

The UBM model can also be visualized as a PGM graph as shown in Figure 2.2.

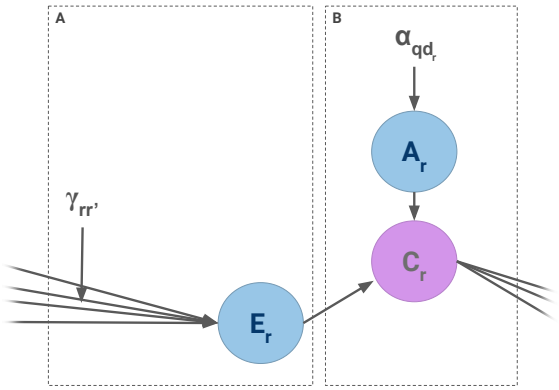


*Figure 2.2: A graphical representation of the UBM model. In area **A** we show how the latent examination variable E_r is based on the previous observed click parameter $\gamma_{rr'}$. In area **B** we show how the click probability for C_r is a combination E_r and the latent attractive variable A_r based on historic query-document interactions parameter α_{qd_r}. Finally, the lines coming from C_r show that the click is used to construct the $\gamma_{rr'}$ parameter in subsequent documents.*

Dynamic Bayesian Network Model

Additionally to the attractiveness α_{qd_r} and continuation parameters γ in the DCM and UBM models, the Dynamic Bayesian Network Model (DBN) by Chapelle and Zhang [8] has a satisfaction parameter σ_{qd_r}. As with the attractiveness parameter, the satisfaction is based on the query-document information. However, in contrast to the DCM and UBM model, the DBN model uses a single stochastic parameter to represent the continuation parameter. Although the continuation parameter is not conditioned on the rank, the model still assumes the user to go from top to bottom. This means that the total probability of examining the next document decreases after each rank. Because of the added parameters, the DBN model has to be optimized using the EM algorithm. Similar to the DCM and UBM models, we assume that a click at each rank C_r is only possible if the document at rank r is both examined E_r and attractive A_r. The DBN model can be formalized as shown Equations 2.15 through 2.21. A graphical representation of the DBN model can be found in Figure 2.3.

$$C_r = 1 \Leftrightarrow E_r = 1 \text{ and } A_r = 1 \tag{2.15}$$
$$P(A_r = 1) = \alpha_{qd_r} \tag{2.16}$$
$$P(E_1 = 1) = 1 \tag{2.17}$$
$$P(E_r = 1 | E_{r-1} = 0) = 0 \tag{2.18}$$
$$P(S_r = 1 | C_r = 1) = \sigma_{qd_r} \tag{2.19}$$
$$P(E_r = 1 | S_{r-1} = 1) = 0 \tag{2.20}$$
$$P(E_r = 1 | E_{r-1} = 1, S_{r-1} = 0) = \gamma \tag{2.21}$$

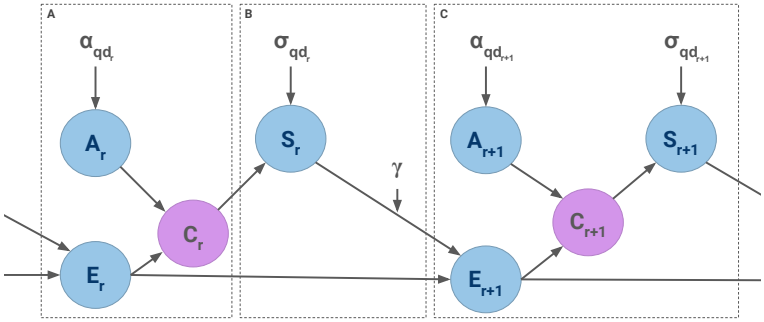


Figure 2.3: A graphical representation of the DBN model. Area ***A*** *shows how the click probability $P(C_r)$ at rank r is determined by the latent attractiveness A_r and examination E_r variables. A_r is based on the historic query-document interactions parameter α_{qd_r}, while E_r is based on whether the user continued browsing after examining the previous document. This continuation, as highlighted in area* ***B****, is a combination of the click probability and historic interactions of document at t weighted by continuation parameter γ. Area* ***C*** *highlights how the parameters at r influence the subsequent rank $r + 1$.*

2.1.3 Neural Click Models

A limitation of the probabilistic graphical click models is their dependency on handcrafted features such as the examination and attractiveness parameters. In this section, we describe work that overcomes this limitation by using a Neural Network architecture.

Neural Click Model

The Neural Click Model (NCM) framework by Borisov et al. [6] was proposed to learn click patterns from user clicks directly. The model uses Recurrent Neural Networks (RNNs) to create an intermediate vector representation based on the historic clicks for each query and ranked document.

The NCM framework as used in the paper is illustrated in Figure 2.4. The area highlighted with **A** shows the initial setup of the RNN. In this step, the model receives a vector q representing observed historic clicks on the provided query. Variables 0_c and 0_d represent the zero vectors with the shapes of the document and interaction vectors needed at the later timesteps. The area highlighted with **B** shows how the click probability at each position is created. At every position, a document vector d_i, representing the historic click observations of the given document-query pair together with the previous interaction c_{i-1}, is passed into the LSTM node. At the first position, there is no previous observed interaction, so it passes an empty interaction 0_c. The final click probabilities $P(c_i)$ are computed by a single feed-forward neural network. Finally, the probabilities are combined in area **C** to get the full conditional probability $P(C_1 = c_1, C_2 = c_2, ..., C_N = c_N)$. This final fully conditional probability is then used with the log-likelihood function as described in Section (4.3.2) in order to update and optimize the model parameters.

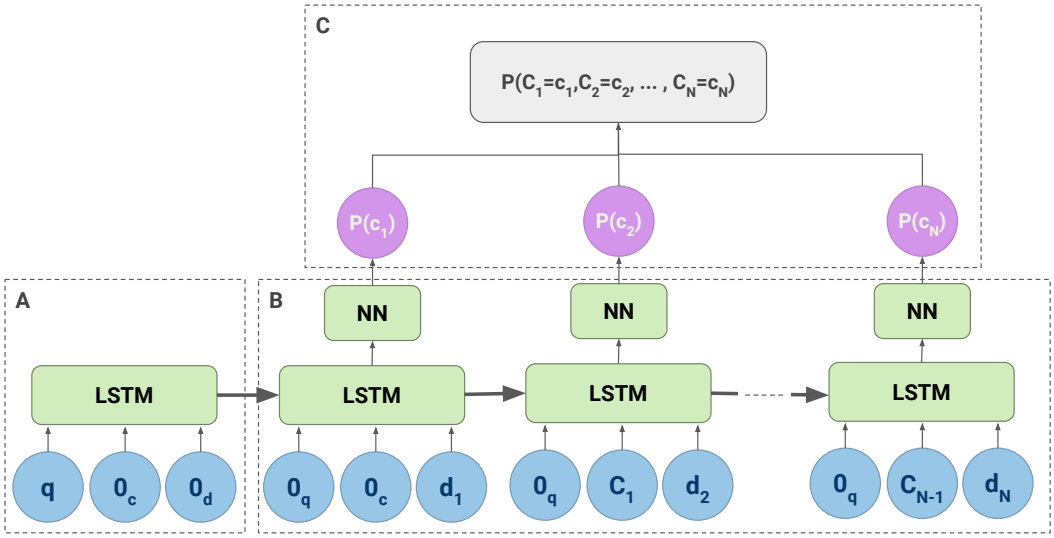


*Figure 2.4: The NCM architecture as introduced by Borisov et al. [6]. Highlighted area **A** is the query initialization, area **B** the prediction of click probabilities, and area **C** is the combination of click probabilities for the log-likelihood loss function.*

For their model, they use both a vanilla RNN and RNN-LSTM [16] configuration. The authors demonstrate that both configurations of the NCM Framework significantly improve performance compared to the PGM-based click models.

2.2 Adversarial Training

In this section, we highlight the potential of using adversarial training for optimizing sequential neural networks such as the NCM model (Section 2.2.1), followed by discussing the fundamentals of adversarial training with Generative Adversarial Networks (GANs) (Section 2.2.2).

2.2.1 Adversarial for Sequential and Discrete Sequential Models

In adversarial training, the loss function of a machine learning model is learned as an additional neural network known as the discriminator. Adversarial training was introduced by Goodfellow et al. [18] as part of the GAN model. Adversarial training is often associated with the generation of naturally looking images [17]. However, many other applications have shown an increase in performance from applying adversarial training [41, 42, 31, 15].

Moreover, Huszár [24] mentions that when doing sequential predictions, such as with the NCM model, the use of a likelihood loss is inappropriate. This is because models tend to overgeneralize and produce implausible samples when maximizing the likelihood. Scheduled sampling by Bengio et al. [3] is designed to overcome the shortcomings of maximum likelihood for sequence modeling, which according to Huszár [24] is not addressing the fundamental problems and is an inconsistent training strategy. Instead, they derive a more optimal objective function based on a generalization of adversarial training. This generalization of adversarial training has shown promising results for various sequential predictions problems such as text generation [41], stock market prediction [42], demand forecasting [31], and prediction of medical data [15].

However, in contrast to user clicks, which are represented as binary values, the above-mentioned works are all trained using samples with continuous values. Moreover, Huszár [24] mentions that at their time of writing it is unclear how to apply adversarial training on discrete probabilistic models because the sampling process cannot be described as a differential operation. However, recent work by Maddison et al. [30] introduces

continuous relaxation of discrete random variables which can be used to address the discrete differentiation problem.

2.2.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) were first introduced by Goodfellow et al. [18]. GANs are a framework for producing more natural-looking samples compared to methods that rely on e.g. maximum likelihood estimation. GANs are optimized using a minimax two-player game between a generative model G and discriminative model D. In the original GAN framework, a generator maps a random input noise variable z to an adversarial sample with $G(z)$. At the same time, the discriminator is trained to distinguish real from adversarial samples. This discriminator is often referred to as an adversarial loss because it replaces the function of an explicit loss. In practice, the GANs are trained by optimizing the value function (Equation 2.22) $V(G, D)$ in a two-player minimax game.

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.22}$$

Although the original GAN framework is able to produce realistic samples, training it to convergence can be challenging. Models often suffer from mode collapse during training, a state where the GAN learns to fool the discriminator by producing a limited set of highly realistic samples, instead of generalizing the problem. This happens when the discriminator is much better than the generator, which causes the gradients of the generator to diminish. The Wasserstein-GAN by Arjovsky et al. [2] solves this problem by changing the cost function of the GAN to use a Wasserstein distance. The Wasserstein distance has smoother gradients, allowing the GAN to learn regardless of whether the generator is performing.

In practice, changing the GAN to a Wasserstein-GAN is achieved by removing the sigmoid function applied to the discriminator output. This causes the model to predict a quality score of the generated sample, rather than a probability. Additionally, this discriminator needs to be a 1-Lipschitz function. This can be achieved by clipping the gradients with a constant value, which is set as an additional hyperparameter. In Wasserstein-GAN terminology, the discriminator is often called a critic, because it judges the output distribution instead of samples being real or fake. The value function for the Wasserstein-GAN can be found in Equation 2.23, where we define the critic function as f_c.

$$\min_G \max_{f_c} V(f_c, G) = \mathbb{E}_{x \sim p_{data}(x)}[f_c(x)] + \mathbb{E}_{z \sim p_z(z)}[1 - f_c(G(z))] \tag{2.23}$$

Another limitation of the original GAN framework is that any generated sample is conditioned only on random input noise. However, many applications such as click models have observed features that directly map to a possible sample space. The ability to condition generative models is essential for these types of applications. Mirza and Osindero [33] demonstrate that the GANs can easily be conditioned on any additional data to direct the data generation process. In their work, the authors condition both the generator and discriminator resulting in the value function $V(G, D)$, which we show as a Wasserstein-GAN in Equation 2.24.

$$\min_G \max_{f_c} V(f_c, G) = \mathbb{E}_{x \sim p_{data}(x)}[f_c(x|y)] + \mathbb{E}_{z \sim p_z(z)}[1 - f_c(G(z|y))] \tag{2.24}$$

Both the generator and discriminator are often trained completely from scratch. To reduce the burden of the reconstruction tasks, several studies [37, 25] suggest training the generator on the discriminator together with a reconstruction loss such as L2. In this case, the discriminator loss is defined as Equation 2.25 and the reconstruction loss as $\mathcal{L}_r$. Both loss functions are added together as shown in 2.26, where both components are weighted by the hyperparameters λ_r and λ_a.

$$\mathcal{L}_a = \max_{f_c} \mathbb{E}_{x \sim p_{data}(x)}[f_c(x|y)] + \mathbb{E}_{z \sim p_z(z)}[f_l(1 - f_c(z|y))] \tag{2.25}$$

$$\mathcal{L} = \lambda_r \mathcal{L}_r + \lambda_a \mathcal{L}_a \tag{2.26}$$

Chapter 3

Method

In this chapter, we discuss the various relevance-based click models that will be used in this work. We first describe how various existing click models can be adjusted to become relevance-based click models in Section 3.1. This is followed by the introduction of our GAN-based click model in Section 3.2.

3.1 Relevance-Based Click Prediction

In this section, we describe how we adjusted various existing click models to relevance-based click models. In Section 3.1.1 we describe how the PGM-based click models can be adjusted to relevance-based click models, followed by Neural Click Models in Section 3.1.2.

3.1.1 PGM-Based Click Models

To get a better understanding of how various click models would perform in relevance-based click predictions, we introduce relevance-based variations of the PGM-based models described in Section 2.1.2. In this section, we formalize the necessary changes to the PGM models to create predictions based on relevance. Each of the models is implemented using the PyClick Python library[1]. The relevance is captured in the variable p_{rel} and can be assigned either -1 (non-relevant), 0 (relevance unknown), or 1 (relevant).

Relevance-Based CTR

The CTR models learn the click probability based on a single input feature, such as the rank-CTR and document-CTR models (Section 2.1.2). In this work, we create an additional CTR model with a parameter p_{rel} based on the available relevance labels for each document. The resulting model can predict three different click probabilities and can be described by Equation 3.1.

$$P(C_{rel} = 1) = p_{rel} \tag{3.1}$$

In line with the document-CTR Model, the relevance-CTR Model will not have any dependency on the document rank.

Relevance-Based DCM

As mentioned in the background Section 2.1.2, the DCM model is often used for relevance-based click predictions. In the mentioned examples, both the attractiveness and continuation parameters of the model are hand-picked for the available relevance grades. However, in the original DCM model by Guo et al. [21], the continuation parameter is based on the rank rather than the document-query information. This provides two possible parameter configurations for our relevance-based DCM model: i) use relevance for both the attractiveness and continuation parameters, or ii) use relevance for the attractiveness parameter and rank for the continuation parameter.

[1] `https://github.com/markovi/PyClick`

To limit the number of models for our experiments, we performed an empirical study between these two configurations. Based on this, we found that using the rank for the continuation parameter outperformed relevance-based continuation (Appendix Figure A.3). Therefore, for our relevance-based DCM model, we only replace the query-document-based attractiveness parameter α_{qd_r} with with a relevance-based attractiveness parameter α_{rel_r}. The rest of the model stays identical to the original DCM model, except for Equation 3.2. A graphical representation of the relevance-based DCM model can be found in Figure 3.1.

$$P(A_r = 1) = \alpha_{rel_r} \tag{3.2}$$

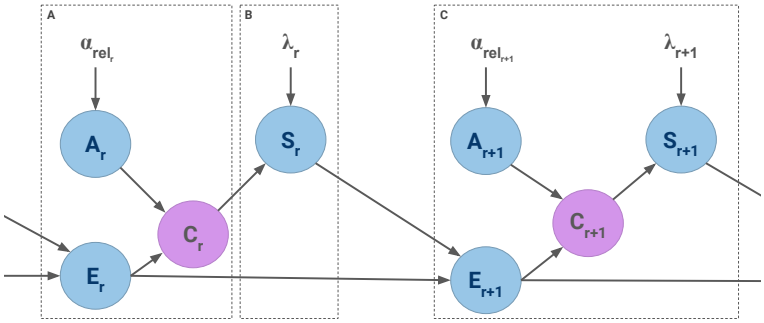


*Figure 3.1: A graphical representation of the relevance-based DCM model. Area **A** shows how the click probability C_r at rank r is determined by the latent attractiveness A_r and examination E_r variables. A_r is based on the relevance parameter α_{rel_r} with the document at r, while E_r is based on whether the user continued browsing after examining the previous document. This continuation, as highlighted in area **B**, is a combination of the click probability and the continuation parameter γ_r for rank r. Area **C** highlights the parameters at r influence the subsequent rank $r + 1$.*

Relevance-Based UBM

Similar to the DCM model, the UBM model has both an attractiveness and continuation parameter. The only difference is that, rather than just the current rank, the continuation parameter is also dependent on the previous click. Because we already demonstrated that it is not beneficial to replace the continuation parameter with the DCM model, we also replace the attractiveness parameter α_{qd_r} with α_{rel} for the UBM model as previously shown in Equation 3.2. A graphical representation of the relevance-based UBM model can be found in Figure 3.2.

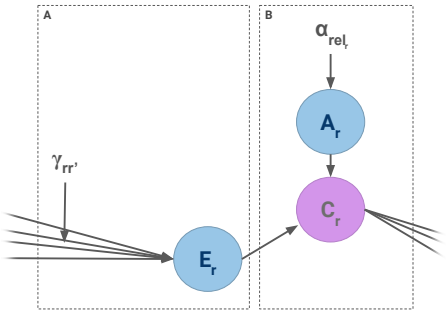


*Figure 3.2: A graphical representation of the relevance-based UBM model. In area **A** we show how the latent examination variable E_r is based on the previous observed click parameter $\gamma_{rr'}$. In area **B** we show how the click probability for C_r is a combination E_r and the latent attractive variable A_r based on relevance parameter α_{rel_r}. Finally, the lines coming from C_r show that the click is used to construct the $\gamma_{rr'}$ parameter in subsequent documents.*

Relevance-Based DBN

In addition to the attractiveness and continuation parameters, the DBN model also has a satisfaction parameter. In the original model, both the attractiveness and satisfaction parameters are based on the query-document information, while continuation is a single stochastic parameter. To keep the changes consistent with the other PGM-based click models, we replace the parameters based on the query-document information with the relevance grades. In contract to changing the attractiveness parameter α_{qd_r} as mentioned in Equation 3.2, we also replace the satisfaction parameter σ_{qd_r} with a relevance-based parameter σ_{rel_r} shown in Equation 3.3. A graphical representation of the relevance-based DBN model can be found in Figure 3.3.

$$P(S_r = 1 \mid C_r = 1) = \sigma_{rel} \tag{3.3}$$

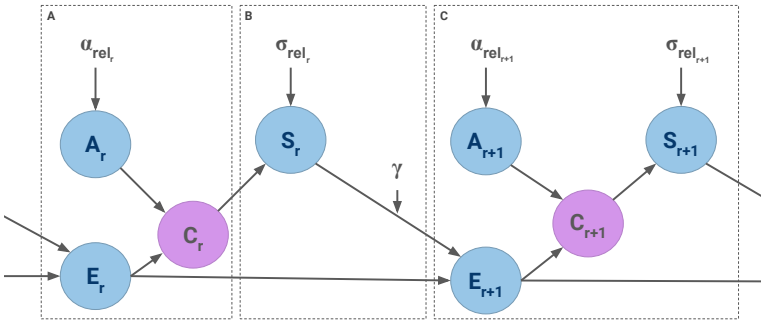


*Figure 3.3: A graphical representation of the relevance DBN model. Area **A** shows how the click probability $P(C_r)$ at rank r is determined by the latent attractiveness A_r and examination E_r variables. A_r is based on the relevance parameter α_{rel_r}, while E_r is based on whether the user continued browsing after examining the previous document. This continuation, as highlighted in area **B**, is a combination of the click probability and historic interactions of document at t weighted by continuation parameter γ. Area **C** highlights how the parameters at r influence the subsequent rank $r + 1$.*

3.1.2 Neural Click Model

In addition to the PGM-based click models, we also introduce a relevance-based Neural Click Model (NCM). However, the NCM Framework as introduced by Borisov et al. [6] is not directly applicable, because there is no query representation in relevance-based click prediction. Additionally, the NCM Framework converts all the historic query-document interactions into one large input feature vector, which is also not possible for the relevance-based click models.

To create a relevance-based NCM model, we first remove the query dependency throughout the whole model. First, we remove the first RNN node from the original NCM model which handled the query initialization. Additionally, we remove the query vector input from each remaining RNN step.

We also completely replace the query-document input vector with a single relevance feature. This leaves us with a vector of size 1×2 consisting of the previous observed click c_{r-1} and current document relevance rel_r as an input. Additionally, we replace the LSTM units used by Borisov et al. [6] with Gated Recurrent Units (GRU) units, which were shown by Chung et al. [12] to be computationally more efficient than LSTM units without compromising performance.

As with the original NCM Framework, the final click probabilities $P(c_r = 1)$ are computed by passing the hidden vector states at each RNN step through a single feedforward layer $NN(x)$. A formalization of the model can be found in Equation 3.4. We show the final relevance-based NCM model in Figure 3.4.

$$GRU(rel_r, c_{r-1}, h_{r-1}) = h_r \tag{3.4}$$

$$NN(h_r) = P(C_r = 1) \tag{3.5}$$

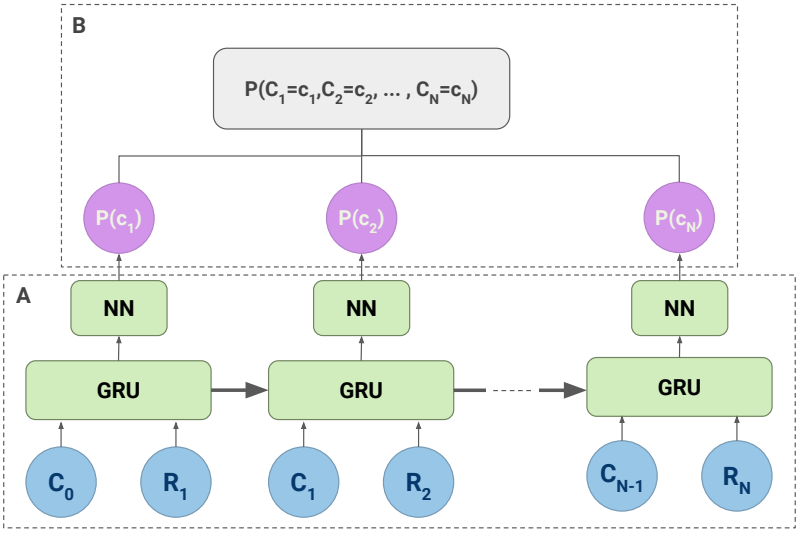


*Figure 3.4: The relevance-based NCM model architecture. Highlighted area **A** predicts a click probability at each timestep given the previous observations, and highlighted area **B** combines the click probabilities to create a full conditional probability for the log-likelihood loss function.*

3.2 Generative Adversarial Click Model

In Section 2.2, we introduced how according to Huszár [24] the use of a log-likelihood loss is inappropriate due to its tendency to overgeneralize. Instead, they introduce a generalization of adversarial training based on the Generative Adversarial Networks (GANs) by Goodfellow [17]. In this section, we introduce how we create an NCM model based on adversarial training. First, we introduce the generator model that generates click sequences based on the available relevance labels (Section 3.2.1). Based on this generator, we introduce a conditional discriminator that uses the produced click sequences and available relevance labels to provide an adversarial loss (Section 3.2.2). Finally, we show how the adversarial loss can be combined with a log-likelihood loss to increase the training capabilities (Section 3.2.3).

3.2.1 Generator

When applying an adversarial loss, our goal is to create a discriminator that can judge the probability of a produced click sequence. However, in contrast to the NCM model, we cannot use the observed clicks at each previous time-step to compute the click probabilities. Doing so would result in: i) a sequence that has been influenced by the actual clicks in the sequence, and ii) probability predictions rather than binary click values.

To overcome these issues, we will need to sample a click at each time-step in the generator sequence. However, these samples need to be both discrete and differentiable, which is non-trivial. A combination of two tricks can be used to achieve both requirements.

First, we need to use a re-parameterized Bernoulli by Maddison et al. [30] named the relaxed Bernoulli [2]. The relaxed Bernoulli works by first taking a uniform sample $L \in \mathbb{R}$ as shown in Equation 3.6.

$$U \sim \text{Uniform}(0, 1) \tag{3.6}$$

$$L = log(U) - log(1 - U) \tag{3.7}$$

[2]The authors refer to this as BinConcrete.

This uniform sample L can now be used to sample a binary value using a Bernoulli parameter α and a temperature parameter λ. The temperature parameter controls how close the samples are to 0 and 1, with a higher λ resulting in a sample value closer to 0 and 1 and vice versa. Equation 3.8 shows how this sample can be taken.

$$X = \sigma(\frac{L + log(\alpha)}{\lambda}) \tag{3.8}$$

Although the resulting sample represents binary values, they are not integers, making it trivial for the discriminator to identify fake click sequences.

Therefore, the second step is to round the samples to actual integer values. However, it is not possible to differentiate through this round operation. Instead, we create a round straight-through operator in PyTorch. This operator rounds the samples on the forward pass but does not reverse the round operation in the backward pass. As long as the temperature parameter is picked such that the difference between the sample and the rounded sample is minimal, this should not influence the overall performance of the model.

```
class RoundST(torch.autograd.Function):

    @staticmethod
    def forward(ctx, input):
        return input.round()

    @staticmethod
    def backward(ctx, grad_output):
        return grad_output
```

The combination of the Relaxed Bernoulli and round Straight-Through can now be used to construct a click sequence generator.

The basis of the generator is identical to the NCM model baseline as described in Section 3.1.2. However, instead of the previous observed click c_{r-1}, we pass the previous sampled click $\tilde{c}_{r-1}$ as an input to the GRU. The produced hidden state vector h_r is then passed through a feed-forward neural network $NN(x)$ to produce the click probability for the current rank. The actual click is then sampled from the Relaxed Bernoulli under the produced click probability.

$$GRU(rel_r, \tilde{c}_{r-1}, h_{r-1}) = h_r \tag{3.9}$$

$$NN(h_r) = P(C_r = 1) \tag{3.10}$$

$$\tilde{c_r} \sim RelaxBern(P(C_r = 1)) \tag{3.11}$$

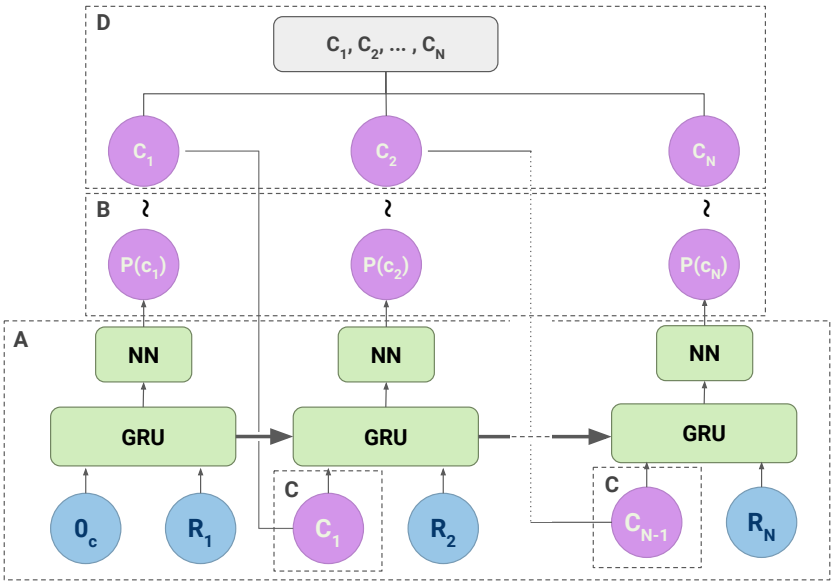


*Figure 3.5: The relevance-based generator architecture. Highlighted area **A** shows how the click probabilities at each steps are computed, area **B** shows where the click samples are drawn from the click probabilities, these samples are then used in area **C** as an input to the next state, and in area **D** to create a sampled click sequence that can be used in the discriminator*

3.2.2 Discriminator

In adversarial training, the discriminator takes over the role of the loss function by determining the direction of gradients for the generator and judging the produced samples. We define our discriminator function $D(x)$ as a simple feed-forward neural network. We model the discriminator based on a conditional Wasserstein GAN, which means that: i) both the generator and discriminator are conditioned on the document relevance R, and ii) the discriminator does not have any activation in the final layer. To keep the parameter K-Lipschitz continuous, the weight should be clamped after each update to be within a small fixed range of $[-c, c]$. We fixed this value c to be the default of 0.01 as mentioned by the original authors [2].

In Equations 3.12 through 3.14, we show how the adversarial loss $\mathcal{L}_a$ is computed using the relevance labels R and sampled click sequence $\tilde{C}$.

$$R = \{rel_1, ...rel_n\} \tag{3.12}$$

$$\tilde{C} = \{\tilde{c}_1, ...\tilde{c}_n\} \tag{3.13}$$

$$D(R, \tilde{C}) = \mathcal{L}_a \tag{3.14}$$

A visual representation of the adversarial loss $\mathcal{L}_a$ construction can be found in Figure 3.6.

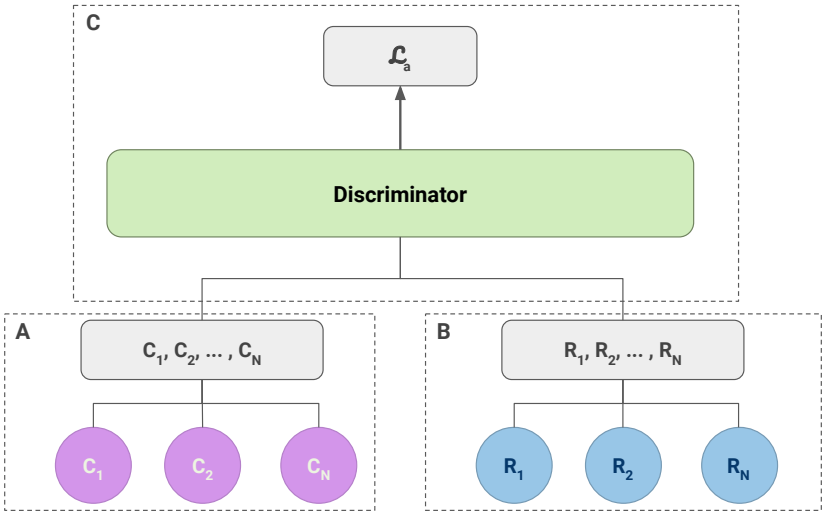


*Figure 3.6: A visual representation of the discriminator loss. The area highlight with **A** represents the sampled click sequence from the generator. The area highlighted with **B** represents the observed relevance labels which act as conditional for the discriminator. Finally, the two inputs are combined and passed through the discriminator network to compute the adversarial loss $\mathcal{L}_a$ in the area highlighted with **C**. The discriminator is a simple feed-forward neural network.*

3.2.3 Joint Adversarial and Log-likelihood Loss

As described by Pathak et al. [37], the adversarial loss can also be used in combination with another existing loss. In our case, we can combine the adversarial loss with the log-likelihood loss. To achieve this joint loss, we need to train the network both using an adversarial loss as described in Section 3.1.2 and using the log-likelihood loss as described in Section 3.2.2. In Figure 3.7, we show how these two models come together. Both networks have the same parameters, which can be shared and updated simultaneously.

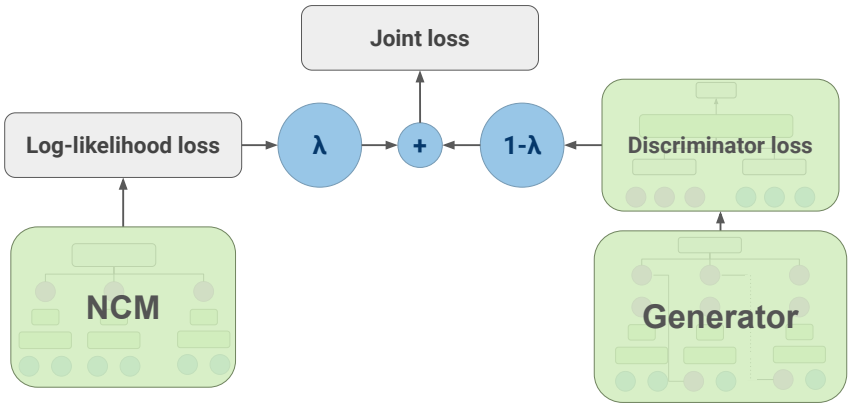


Figure 3.7: When calculating the joint adversarial and log-likelihood loss, both models perform their probability prediction and gradient calculations separately. The final gradients are then combined according to the weight parameter λ.

By tuning attribution parameter λ, we can control how much of the updates are coming from the adversarial loss $\mathcal{L}_{adv}$ versus the log-likelihood loss $\mathcal{L}_{LL}$. The formalization of this joint loss can be found Equation 3.15.

$$\mathcal{L}_{joint} = \lambda \mathcal{L}_{LL} + (1 - \lambda)\mathcal{L}_{adv} \tag{3.15}$$

Chapter 4

Experimental Setup

In this chapter, we introduce our experimental setup. First, we highlight various aspects of the dataset used in the experiments (Section 4.1). Second, we introduce the methods and parameters that were used to optimize our neural-based models (Section 4.2). Finally, we explain how our results are evaluated by introducing a set of baselines and evaluation metrics (Section 4.3).

4.1 Dataset

In this section, we go over the various aspects of the dataset used in this work.

Yandex Relevance Prediction Dataset

The Yandex Relevance Prediction dataset is a large click log on Yandex search queries submitted in 2009 and contains roughly 44 million query sessions split into a train and test set. Additionally, the dataset also contains $41,275$ binary relevance judgments based on the query, document and region id. The region represents the country from which the query was requested. In our work, we removed this region for each of the judgments by always taking the maximum ranking between all regions for any query-document pair. For example, if a query-document pair is judged as both relevant and non-relevant for two different regions, then the document is considered relevant.

Total judgments	41,275
Total judged queries	4,991
Pairs with different judgments per region	302
Queries with 10 or more judgments*	1,671
Queries with less than 10 judgments*	3,320

*Table 4.1: This table shows how the relevance judgments were distributed over different queries. The * indicates the counts after removing the region duplicates.*

To improve the turnover time of experiments, we only use a subset of the actual dataset. Initial experimental results showed that about 1 million training samples are enough to converge. In addition to these 1 million training samples, we also use 100,000 samples for validation and 1 million samples for testing.

To leave room for expansion, the training data uses the first 1 million queries from the original train split, the validation data uses the first 100,000 queries from the original test split, and the test data uses the second 1 million queries from the test split.

The original judgments were made as either 1 for relevant and 0 for not relevant. However, we want to also feed queries that are missing judgments into our model. Because the values are fed into the model as a single discrete input feature, we remap the judgments to be on a discrete scale with 1 (relevant), -1 (not relevant), and 0 (not available). The availability of these judgments can be found in Table 4.2 and Table 4.3.

Judgments	*Train (1m)*	*Validation (100k)*	*Test (1m)*
0	74.4%	75.9%	75.4%
1	0.6%	0.5%	0.6%
2	1.6%	0.8%	0.8%
3	2.0%	1.1%	1.1%
4	1.6%	1.9%	2.2%
5	4.5%	2.9%	3.0%
6	3.2%	4.6%	4.8%
7	3.9%	4.4%	4.4%
8	2.5%	3.8%	3.3%
9	4.0%	2.5%	3.0%
10	1.2%	1.0%	1.0%

Table 4.2: The amount of relevance judgments per query in the train, validation and test set. In each set, about 25% of queries has at least one relevance judgment.

Judgment	*Train*	*Validation*	*Test*
Relevant (1)	11.3%	11.0%	11.0%
Unknown Relevance (0)	84.5%	84.8%	84.5%
Not Relevant (−1)	4.2%	4.2%	4.5%

Table 4.3: The distribution of relevant (1), not relevant (−1), and not available (0) judgments in the different splits.

Correlation Between Clicks and Relevance Labels

In Figure 4.1, we highlight the existing correlation between the relevance judgments and observed interactions in the Yandex dataset. For comparison, we also show the correlation with the position and interaction/relevance on the previous document. We have added these features because, except for previous relevance, they are already used in click models such as CTR, DCM, and UBM.

In the figure, a positive correlation indicates that a higher value for the feature correlates to a higher probability of a click being observed as well. For the position, we inverted the values by subtracting the number of documents N with the document rank i, such that a positive correlation can be plotted.

Figure 4.1: This plot shows the correlations of the observed interactions with i) the relevance and position of the current document, and ii) the relevance and observed interaction of the document at the previous rank. The correlations were computed on the test set.

Figure 4.1 clearly shows that the position and previous interaction have the strongest correlation with the observed interactions. However, the interactions are also clearly correlated with the relevance judgments, indicating that it indeed could be used as an indicative feature. Additionally, we also see a relatively small negative correlation with the relevance of the previous document, which could point to a potential indicator for satisfaction.

HDF5

To increase productivity while working with the click logs, we converted the final dataset as described in Section 4.1 to HDF5[1] (Hierarchical Data Format 5). HDF5 is a cross-language and cross-platform storage format which can efficiently store data in matrices on disk. The matrices can be sliced with continuous write speed and efficiently loaded into memory. This allows us to i) load the data in parallel during training, and ii) easily load and analyze the data in various environments and platforms.

4.2 Hyperparameter Optimization

Optimizing the hyperparameters is a key step of maximizing the performance of neural networks. In this section, we discuss the hyperparameter optimization method and setup used in our work. Finally, we present the hyperparameters that were used for the presented results.

Tree of Parzen Estimators

For hyperparameter optimization, we are using the automated optimization framework Hyperopt[2] by Bergstra et al. [4]. Hyperopt is Python library that can optimize any model or function using either Random Search or Tree of Parzen Estimators (TPE) by Bergstra et al. [5]. The TPE algorithm can optimize the Expected Improvement (EI) [28] of a graph-structured hyperparameter configuration space. This configuration space defines the distributions from which each of the hyperparameters can be sampled while the graph-structure allows certain parameters to be sampled in a sequence (eg. first sample number of layers, then sample the configuration for each of the layers).

Before determining the shape of the hyperparameter space, Hyperopt will first do several trials with random parameters. After these trials, each set of hyperparameter is sampled according to the hyperparameter space based on the previous trials.

To increase the turnover of experiments, we implement Hyperopt's ability to evaluate multiple hyperparameter samples in parallel. The machine we used for training was able to run three trials in parallel. It is important to note that because these three trials were running in parallel, the hyperparameter space is not updated for each instance which can slightly affect optimizing performance. However, according to the authors, this is easily outweighed by the number of trials that can be performed in a now shorter period.

Optimization Setup

To use the optimization methods described in Section 4.2, we need to define a hyperparameter space. For each hyperparameter, we define a range (e.g. $[1 - 5]$) and one of the following sampling methods: log uniform, continuous uniform or discrete uniform. The sampled hyperparameters for both the NCM and GAN model are optimized together with the Adam optimizer by Kingma and Ba [29] to find the optimal model parameters.

For the NCM model, we vary three hyperparameters: learning-rate, hidden-size and batch size. Based on early empirical results, we decided not to add any parameters for the number of layers in the NCM. The parameter space used for optimizing the NCM can be found in Table 4.4.

Parameter	*Sample type*	*Space*
Learning rate	Log uniform	$[1e^{-5}, 1e^{-2}]$
Hidden size	Discrete uniform	[20, 256]
Batch size	Discrete uniform	[1, 128]

Table 4.4: The parameter space used for optimizing the NCM model.

Because the GAN-based model extends the NCM model, it also inherits all its hyperparameters. Additionally, the GAN-based model has a hyperparameter for the contribution of the log-likelihood, a parameter

[1] `https://www.hdfgroup.org/solutions/hdf5/`
[2] `https://github.com/hyperopt/hyperopt`

for gradient clipping, and two hyperparameters for the dropout and size of the discriminator. For consistency, the shared hyperparameters with the NCM are kept identical between the two hyperparameter spaces. We set the gradient clipping to the default (0.01) used in the original Wasserstein-GAN paper [2]. The full list of tuned hyperparameters for the GAN-based model can be found in Table 4.5.

Parameter	*Sample type*	*Space*
Learning rate	Log uniform	$[1e^{-5}, 1e^{-2}]$
Hidden size	Discrete uniform	[20, 256]
Batch size	Discrete uniform	[1, 128]
Discriminator dropout	Continuous uniform	[0.0, 0.9]
Discriminator hidden size	Discrete uniform	[5, 264]
Log-likelihood contribution	Continuous uniform	[0.2, 0.8]

Table 4.5: The parameter space used for optimizing the GAN model.

Both models are tuned for 100 different configurations before picking the optimal hyperparameters. The final optimized hyperparameters after tuning are displayed in Table 4.6.

Parameter	*NCM*	*GAN*
Learning rate	$5.644e^{-05}$	$3.544e^{-03}$
Hidden size	239	25
Batch size	18	95
Discriminator dropout	-	0.0841
Discriminator hidden size	-	91
Log-likelihood contribution	-	0.3514

Table 4.6: The optimized hyperparameters for the NCM and GAN-based model after testing 100 different configuration using the Tree of Parzen Estimators.

4.3 Evaluation

In this section, we discuss the baselines and metrics used for evaluating the proposed models in this work. The baseline methods are discussed in Section 4.3.1, which is followed by the metrics in Section 4.3.2.

4.3.1 Baselines

In this work, we compare the performance of various relevance-based click models. To make sure that the relevance parameters add any value, we compare all results to a set of baselines based on the rank-based CTR model and most occurring patterns. In this section, we introduce and motivate the baselines used in this work.

Most occurring patterns

The click patterns that are observed in the Yandex Relevance Prediction Dataset are very skewed. From the 1 million sessions in the test set, we observe $311,908$ sessions with only 1 single click at the first rank and $296,142$ sessions with no clicks at all.

To make sure that our model is not doing worse than predicting the obvious click patterns, we use these two click patterns as baselines. In the rest of this thesis, we will refer to these baselines as *Click First Rank* when we always predict a click probability of 1.0 at the first position and 0.0 for the remaining positions, and *Click None* when a click probability of 0.0 is predicted for all positions.

Rank-based CTR

The rank-based CTR model is one of the few models that can be optimized without any additional input features. Using the rank-based CTR model as a baseline ensures that the relevance-based models are not doing worse than just modeling the position bias. We refer to the rank-based CTR model as Rank-CTR or RNK-CTR in the rest of this thesis.

4.3.2 Metrics

To evaluate the performance for a collection of models, it is important to have performance metrics that are aligned with the key objectives. In the context of relevance-based click models, we are interested in predicting the probability of a user clicking document d at position r of query q conditioned on the observations at the previous positions. In this section, we will explain and motivate the metrics used to report on our relevance-based click models.

Log-likelihood

The likelihood is a method to describe how likely the parameters of a model are for the observations made in a dataset. For click models, the likelihood is computed by taking the product of the predicted probabilities of each observed click sequence. The likelihood function $\mathcal{L}(M)$ is described in Equation 4.1. The equation shows the likelihood over each query session s in the total set of query sessions S.

$$\mathcal{L}(M) = \prod_{s \in S} P_M(C_1 = c_1^{(s)}, ..., C_n = c_n^s) \tag{4.1}$$

Because the product in the likelihood function can result in very small numbers, we use the log-likelihood instead which uses a natural logarithm to make the formula more numerical stable. Additionally, it also normalizes the log-likelihood to make it easier to compare performance for different datasets. The full log-likelihood is formalized in Equation 4.2.

$$log\mathcal{L}(M) = \frac{1}{|S|} \sum_{s \in S} logP_M(C_1 = c_1^{(s)}, ..., C_n = c_n^s) \tag{4.2}$$

All of the models used in this work, except for the baselines and CTR model, are auto-regressive in nature, meaning that each prediction is conditioned on the previous observed clicks and relevance. Although the log-likelihood function stays identical for the auto-regressive models, the method of calculating the likelihood of observing a particular click sequence should explicitly take the dependencies in mind. Equation 4.3 formalizes how this probability should be computed, with $c_{<i}$ as all previous observed clicks, and $r_{<=i}^{(s)}$ representing both the current and previous observed relevance grades.

$$P_M(C_1 = c_1^{(s)}, ..., C_n = c_n^s) = \prod_{i=0}^{N} P_M(C_i = c_i^{(s)} \mid c_{<i}^{(s)}, r_{<=i}^{(s)}) \tag{4.3}$$

Although the GAN-based model is not trained auto-regressive, the evaluation is still performed using the auto-regressive assumptions.

It is important to note that, compared a metric like accuracy, the log-likelihood penalizes extreme errors more heavily. Because user clicks are very noisy and only observed on about 12% of the documents, it would be easy to cheat the accuracy metric by predicting a click probability of 0.0 (Appendix Figure B.1). Because of this, the likelihood is more suitable for the application of click models.

Area Under Curve (AUC)

As an additional metric, we report the Area Under Curve (AUC) scores for each of the models. The AUC is an evaluation metric, often used in classification tasks, which reflects the rate of true and false positive predictions made by a model. In contrast to a metric such as the F1-score, which also reflects true and

false-positive rates, the AUC model is calculated using all possible decision boundaries[3]. Because click models predict click-through rates, rather than classifying the clicks, there is no decision boundary to be set. Moreover, decision boundaries need to be selected based on assumptions about the importance of true positives compared to false positive, which cannot be made for tasks other than classification. However, because the AUC score is determined over all possible decision boundaries, it gives an understanding of the ability of the model to make a distinction between true and false positives.

To give an understanding of how the AUC score is determined, we show an example Receiver Operating Characteristic (ROC) curve in Figure 4.2. The area under this ROC curve is what we previously described as the AUC.

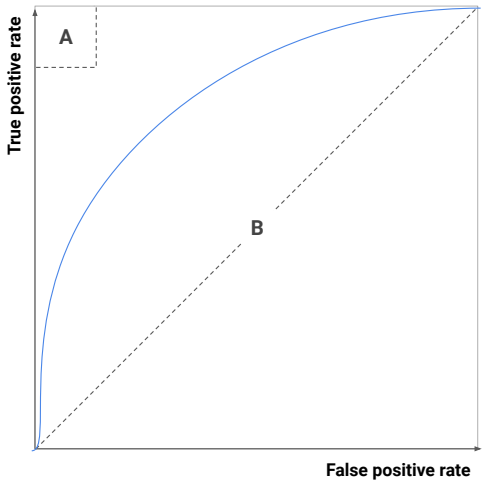


*Figure 4.2: An example Receiver Operating Characteristic (ROC) plot. The blue curved line represents the ROC curve. A high-quality model would have a curve within the area highlighted with **A**. The dotted line highlighted with **B** represents the ROC line when predictions which are either constant or random. The area under the blue curve represents the AUC score.*

Each point on the ROC curve represents the rate between true and false positives for a specific decision boundary. When the curve goes through the area highlighted with **A**, this means that a decision boundary exists which can achieve a high true positive rate without introducing a high false-positive rate as well. This would result in an AUC score close to 1. Moreover, when predictions are either constant or random we expect a curve identical to the dotted line highlighted with **B**, resulting in an AUC score of 0.5. A higher AUC score indicates a better distinctive performance. Moreover, an AUC score of 1 indicates a perfect classifier while any score below 0.5 would indicate that the model would do worse than random.

Perplexity

Click Model research often uses the perplexity, shown in Equation 4.4, as a metric to evaluate model quality. The perplexity is calculated by using the full click probability of observing a click at each position $P_M(C_r = c_r^{(s)})$. To compute this probability for the click models used in this work, we need to marginalize over all previously observed clicks. This is because these models are trained as auto-regressive models such that they optimize $P_M(C_i = c_i^{(s)} \mid c_{<i}^{(s)}, r_{<=i}^{(s)})$. This causes models that closely resemble optimizing a full-click probability, such as the rank-based CTR model, to have a significant advantage. To avoid confusion, we decided to omit the perplexity from our results.

[3]A decision boundary defines the probability threshold which is used to convert a predicted probability to a class. For example, when doing binary classification with a decision boundary of 0.5, all predictions below 0.5 are classified as 0 while all predictions above 0.5 are classified as 1.

$$p_r(M) = 2^{-\frac{1}{|S|}\sum_{s\in S} log_2 P_M(C_r=c_r^{(s)})} \tag{4.4}$$

Significance testing

To determine the statistical significance for model results, we need a form of significance testing. Because we have a consistent set of samples, we can use a paired t-test to determine significance. For all our experiments, we consider a significant difference when $p < 0.05$.

Chapter 5

Results

In this section, we will discuss the results of the performed experiments and answer our research questions.

The chapter is split up into two sections. First, we present and discuss the results to answer RQ_1 and RQ_2 in Section 5.1. Secondly, we present the additional results needed to answer RQ_3 and RQ_4 in Section 5.2.

5.1 Relevance-Based Click Prediction

In this section, we provide the results that answer RQ_1 and RQ_2.

First, we present the performance of the various relevance-based click models and baselines (Section 5.1.1). To motivate these results, we provide an analysis of the different behaviors captured by each click model (Section 5.1.2).

5.1.1 Model Results

In this section, we answer RQ_1 as formulated below:

- *Which model has the best performance when using relevance judgments as an input feature?*

To address RQ_1, we compare the log-likelihood and AUC (Section 4.3.2) performance of the various relevance-based click models. The relevance-based click models are categorized in: i) the PGM click models, namely the UBM, DCM, DBN, and relevance-CTR models (Section 3.1.1), and ii) the neural click model, namely the NCM model (Section 3.1.2). For comparison, we also report the performance of three baselines methods (Section 4.3.1) representing: i) two naive methods of predicting the two most commonly seen click patterns, being no clicks at all or a single click at the first rank, and ii) the rank-based CTR model. The predictions made by the baseline methods are not conditioned on the relevance labels. The performance of all baselines and models is shown in Table 5.1. All the reported results are tested as significantly different.

Model	*Log-likelihood*	*AUC*
Click First Rank	−10.634	0.6611
Click None	−8.472	0.5000
Rank-CTR	−3.160	0.7565
NCM	**−2.833**	**0.8380**
UBM	−2.877	0.8326
DCM	−3.393	0.7483
DBN	−3.541	0.7283
Rel-CTR	−3.690	0.5434

Table 5.1: The performance of the baseline methods and relevance-based click models. All the differences in performance are significant. For both the AUC and log-likelihood a higher value indicates a better model quality.

From the table, we answer RQ_1 by concluding that the best performance is achieved by the NCM model. Additionally, the table shows that the performance differs among the various reported models. In the next section we address this performance difference by answering RQ_2. For readability, we omit from reporting the most common click pattern baselines in the subsequent sections.

5.1.2 Explaining Relevance-Based Click Model Behavior

In this section, we answer RQ_2 as formulated below.

- *How can we explain the differences in performance between the relevance-based click models?*

To answer this question, we address the differences between the relevance-based click models using both quantitative and qualitative analyses. For our quantitative analysis, we highlight the various correlations between the predicted click probabilities from each model compared to their corresponding input features. These input features are identical to the features in Figure 4.1 which reported the correlations between the observed clicks and their corresponding observed features in the Yandex dataset (Section 4.1). These correlations provide a general understanding of the differences between each of the relevance-based click models.

Our qualitative analysis is set out to highlight the more complex differences between the various click models. Because all models are deterministic in nature, we can test the model behavior in a controlled environment. In these controlled environments, we explicitly set the document relevance and observed clicks at each position. By comparing the differences in predicted click probabilities after changing specific parts of the environment, we provide an intuition of the behavior learned by the models. Although it would be possible to learn these interpretations directly by inspecting the parameters of the PGM-based models (Appendix A) we refrain from doing such an analysis because these interpretations cannot be made with the parameters of the NCM model.

In Figure 5.1, we show the correlations between the click probabilities made by each click model and their corresponding observed input features. For reference, we also add the correlation between the input features and the observed click in the Yandex dataset.

Figure 5.1: The correlation between the input features and predicted click probabilities from the various relevance-based click models. For reference, we also added the correlation between the input features and the observed clicks.

These correlations provide us with a general understanding of the differences between each model. First, this highlights that the Rank-CTR and Rel-CTR model only take into account the rank and relevance, respectively. Additionally, we see a clear difference in behavior between the various PGM-based models, while the UBM and NCM model seem to be very similar. To answer RQ_2, we analyze how the relative differences between each of the correlations affect the model performance. First, the relevance-CTR and DBN models, which reported the lowest performance, show a higher correlation to document relevance than to both the document rank and previous interaction input features. The DCM model, which reported

a significantly higher performance, has a relatively higher correlation to the document relevance than to the previous interaction. For the UBM and NCM models, which reported the best performance, we see a relatively similar correlation between the predicted probability and each of the individual input features as seen with the correlations to the observed clicks in the actual dataset. These observations indicate that the performance difference as addressed in RQ_2 can, at least partially, be explained by the ability of each click model to learn the influence of document rank, relevance and previous observed interaction on its predicted click probability.

In the subsequent paragraphs, we visualize the behavior of the relevance-based click models in several controlled environments. The configurations of these controlled environments are based on each of the correlations found in Figure 5.1. These visualizations highlight some of the complex behavior learned by the various relevance-based click models.

Position Bias

From Figure 5.1, we can see that, except for the Rel-CTR model, all models predict the click probability with a relatively high correlation to the rank. To get an understanding of how the click probability changes over time, we can look at Figure 5.2. In this figure, we show the click prediction behavior for each of the models for a sequence where the relevance for each document is unknown and there are no observed clicks.

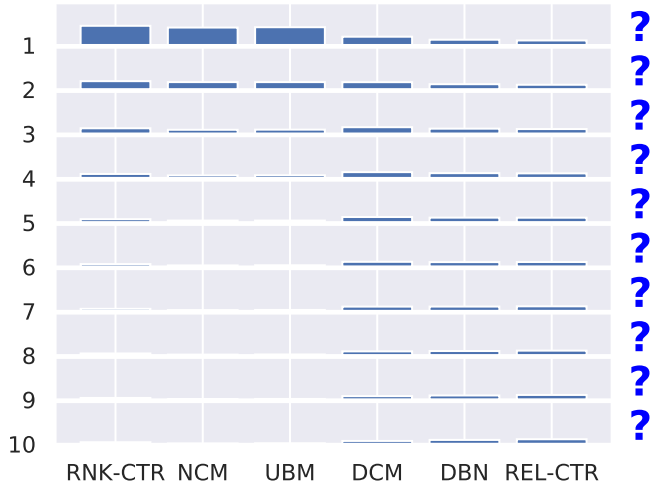


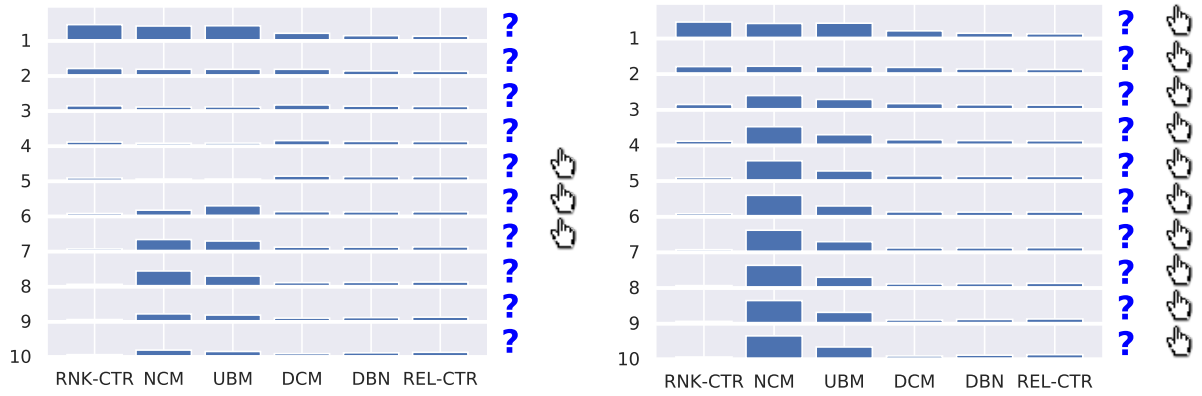
Figure 5.2: The position bias for each of the relevance-based click models. Predicted click probabilities for each rank are shown as a bar from top to bottom. The blue question marks indicate that the relevance for each position is unknown. These probabilities were predicted without observing any clicks.

In the figure, we see that the NCM and UBM models are very similar to the Rank-CTR model, which is merely a reflection of the average probability in training dataset. However, the DCM and DBN models seem to have a visibly lower bias towards position than the UBM and NCM model, which is likely because they do not have an explicit parameter for rank. Finally, we can see that the Rel-CTR model is predicting a consistent click probability based on the missing relevance label regardless of the rank, which is expected.

In both Figure 5.1 and Figure 5.2, we see that attributing rank for predicting click probability is a strong indicator of performance. Moreover, the Rank-CTR, UBM and NCM models, which have the best performance, not only predict click probabilities with a high correlation with position but also show a much stronger position bias than the other relevance-based click models.

Previous Observed Interactions

Figure 5.1 showed that the predictions from NCM and UBM model both have a relatively high correlation with the previous interaction. This is also observed when we look at Figure 5.3, where we show the effect of observing a set of clicks for a sequence of documents with unknown relevance.



(a) A sequence of documents with unknown relevance, where clicks on document 5,6 and 7 were observed.

(b) A sequence of documents with unknown relevance, where a click was observed on all 10 documents.

Figure 5.3: The effect of observing a sequence of clicks on the predicted click probabilities for each model. Because of its long-term memory, the NCM model is the only model that can capture a sequence of clicks. The mouse icon indicates an observed click at the given position.

In this figure, we see that for both models the click probability is higher when a click is observed at the previous rank. Because we control for the document relevance, we can see that both models learn that a click is more likely to be observed after the previous document has been clicked. However, the UBM model only has information about the last click while the NCM model also models long-term dependencies. This long-term dependency allows the NCM model to further increase the click probability when more consecutive clicks are observed. This behavior is likely because the models find that a click on later ranks implies that these documents have been examined. When the user has shown to examine these later documents, it is more likely that the users will also examine the subsequent documents. Figure 5.3a also shows that when no more clicks are observed, the click probability goes down again. For the Rank-CTR, Rel-CTR, DCM and DBN model, we know that there are no parameters that depend on previous observed interactions, which is reflected in the results.

Based on the observations made with the UBM and NCM model, which report the best performance, we can assume that the ability to learn click probabilities from previous interactions is a strong indicator for performance.

Relevance-Based Patterns

From Figure 5.1, we can see that all of the relevance-based click model predictions have a stronger correlation to relevance than the observed interactions. This indicates that the models are using the relevance labels to a certain extent to predict click probabilities.

Figure 5.4 shows the behavior of each relevance-based click model when relevant or non-relevant labels are introduced. The figure shows clicks and probabilities similarly as in Figure 5.3a together with relevant and non-relevant documents at given positions, indicated with green and red arrows, respectively.

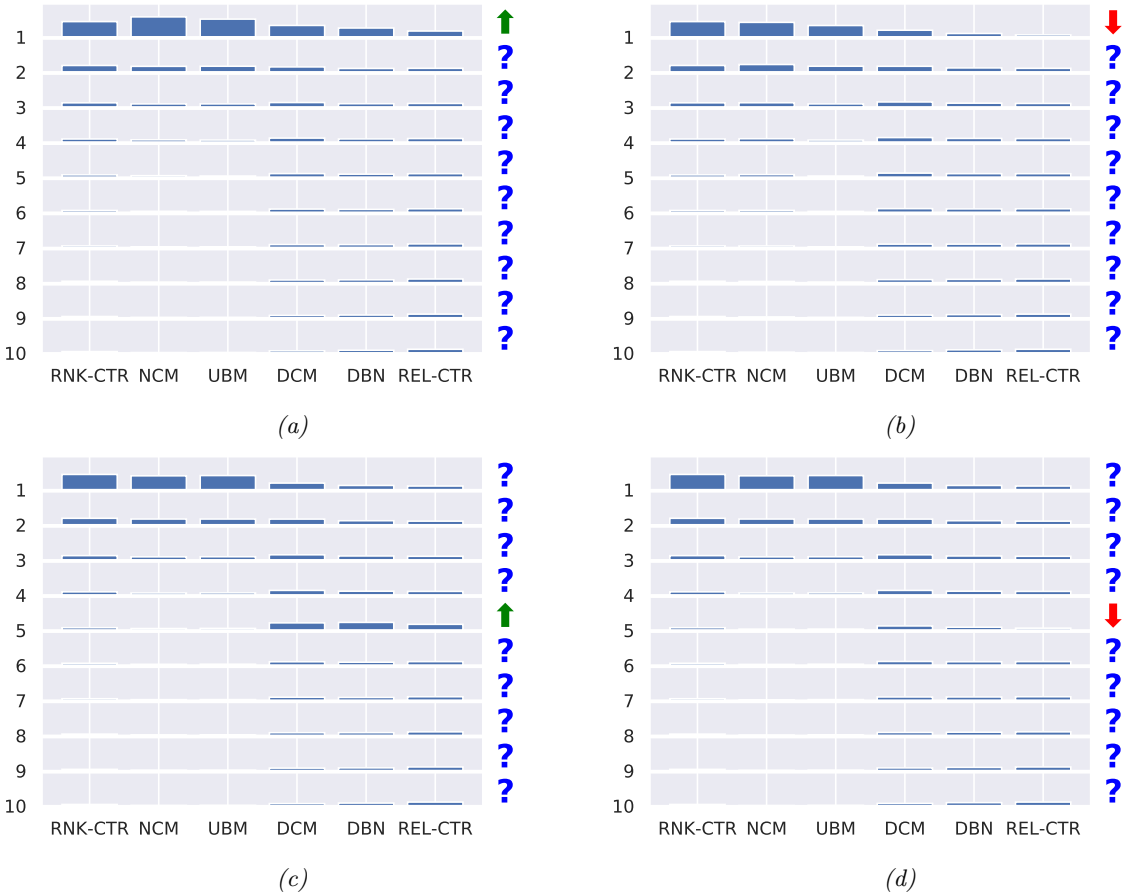


Figure 5.4: The effect of a positive or negative relevance label at various positions. DCM, DBN, and Rel-CTR have a visible increase in click probability regardless of the position of the relevance label, while the absolute increase in click probability is visibly lower for the NCM and UBM models when the relevance label is at a later position. The green arrow indicates relevance, the red arrow indicates non-relevance, and the blue question mark indicates unknown relevance for each given rank.

When we look at Figure 5.4a and 5.4b, we see that the relevance label for the document at the first position has a visible influence on the click probability for all relevance-based models. This shows that all models learn that relevant documents are more likely to be clicked than non-relevant documents. However, when we observe a relevance label at a later rank, as seen for rank 5 in Figure 5.4c and 5.4d, we see that for the UBM and NCM models there is no visible change in click probability. In contrast, the DCM, DBN, and CTR models, which do not strictly have a rank-based parameter, still visibly increase their predicted click probability. The behavior we observe with NCM and UBM shows how the models reflect the position bias, even when the document is relevant.

While the PGM-based models can learn a limited set of behavior due to their explicitly modeled dependencies, the neural architecture of the NCM models can learn additional click behavior. This is shown in Figure 5.4a and 5.4b, where we see that the click probability goes down when the first document is skipped and relevant, compared to when the first document is skipped and not relevant. This seems to reflects that users often intentionally skip the non-relevant document while skipping relevant documents could be a sign of abandoning the search.

Additionally, we observe that the NCM model learns a form of satisfaction from the relevance labels as seen in Figure 5.5.

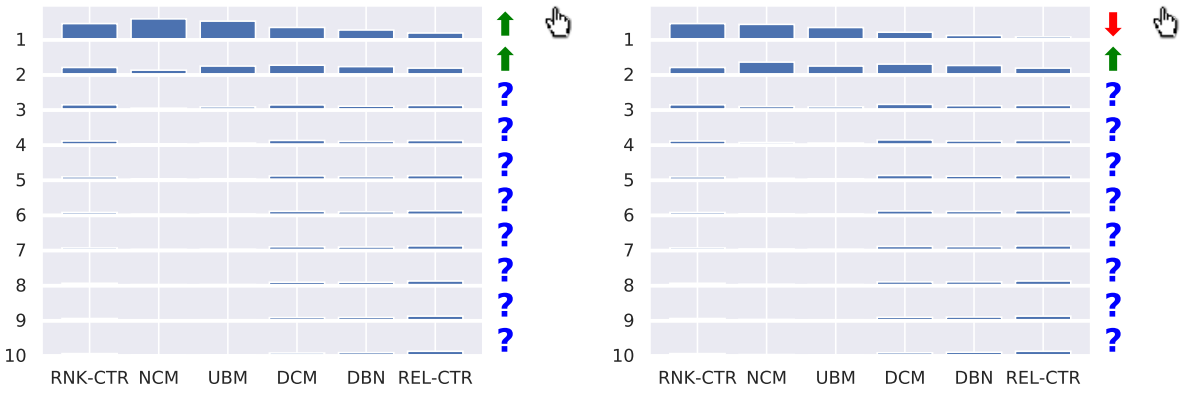


Figure 5.5: Satisfaction behavior modeled by the NCM model. Because the PGM-based models are not able to capture the relevance of the previous document, they are not able to learn similar satisfaction behavior.

When comparing the click probability at rank 2 between Figure 5.5a and 5.5b, we see the click probability increase when the non-relevant document at rank 1 is clicked, while it decreases when the previous document was relevant.

From these results, we can assume modeling the click probability from relevance alone is not a strong indicator for performance. However, the ability of the NCM model to capturing complex click behavior using a combination of relevance labels, clicks and rank sets it apart from the other models. Moreover, the ability to capturing this complex click behavior is a strong indicator of increased performance.

To conclude, we answer RQ_2 by summarizing the differences highlighted in this section. First, we showed that being able to correctly estimate the relative correlations between the various input features and click probability predictions is a strong indicator for good performance. Models such as the DCM and DBN model predict click probabilities that are strongly correlated to the relevance labels, which is at the expense of stronger click indicators such as document rank and the previous interaction. Moreover, models such as NCM and UBM, which demonstrate a strong bias towards document rank and the previous interaction, demonstrate significantly higher performance than the other models. Finally, we conclude that although the behaviors of the UBM and NCM models are very similar, the NCM model has a higher performance due to its ability to capture various complex click behaviors such as multiple clicks and click satisfaction.

5.2 Generative Adversarial Click Models

In this section, we provide the results that answer RQ_3 and RQ_4. Throughout this section we refer to the model trained with an adversarial loss (Section 3.2) as the GAN model.

First, we compare the performance of the GAN model to the NCM and UBM model (Section 5.2.1). To motivate these results, we provide an analysis of what is causing the performance difference between the GAN and NCM model (Section 5.2.2).

5.2.1 Model Results

In this section, we answer RQ_3 as formulated below.

- *Does training a Neural Click Model with an adversarial loss improve performance?*

We address this question by comparing the performance of the GAN model with the UBM and NCM models as shown in Section 5.1.1. The GAN model uses a joint loss (Section 2.26) consisting of a log-likelihood loss (Section 3.1.2) and an adversarial loss (Section 3.2). The contributions of both loss functions to the total joint loss is controlled by a hyper-parameter as described in Section 4.2.

In Table 5.2, we show the log-likelihood and AUC performance of the three models.

Model	*Log-likelihood*	*AUC*
GAN	**−2.825**	**0.8392**
NCM	−2.833	0.8380
UBM	−2.877	0.8326

Table 5.2: The difference in log-likelihood and AUC performance between the GAN, NCM and UBM models. All results are significantly different. For both the AUC and log-likelihood a higher value indicates a better model quality.

From the table, we see that the GAN model significantly improves both the log-likelihood and AUC performance compared to the UBM and NCM models. Although the performance is significant, we do see that the increase of the GAN compared to the NCM model is relatively smaller than between the NCM and UBM model.

In the next section, we highlight what is causing the performance difference between the GAN and NCM models. For readability, we limit the analyzes to the log-likelihood performance.

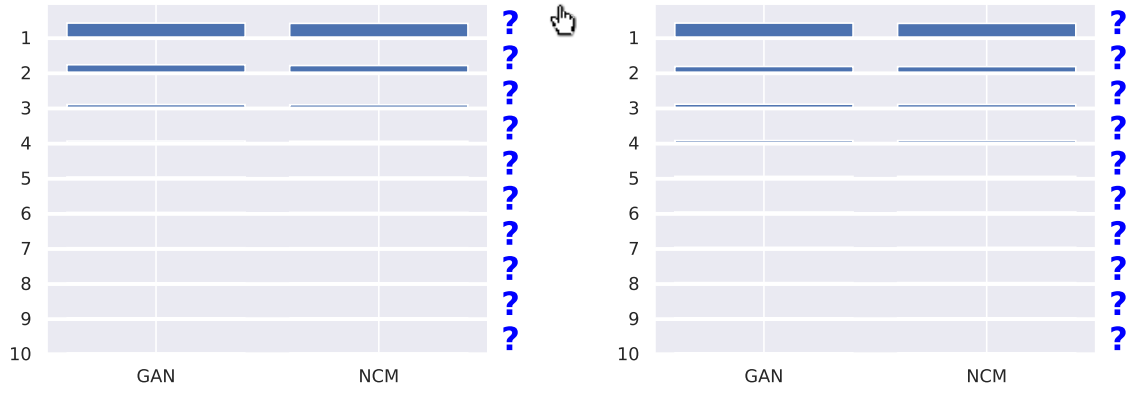
5.2.2 Difference Between GAN and NCM

In this section, we provide the insights to answer RQ_4 as formulated below.

- *If any, what is causing the performance difference between a regular and adversarial trained Neural Click Model?*

From Section 5.2.1, we learned that the GAN significantly outperforms the other relevance-based click models. Because the GAN model is using the same neural architecture as the NCM model, the differences in qualitative analyses are less apparent than the differences seen before in Section 5.1.2. Instead, we address the differences between the two model by using only quantitative approaches. To get an understanding of these differences, we analyze the performance difference between the two models in various segments of the dataset. These segments are based on various aspects of the individual sequences within the dataset. Moreover, the segments provide an understanding of which types of click behavior is causing the difference in performance. To do so, we first analyze the difference in performance between the two models for the two most common click patterns. Additionally, we analyze how the rank and number of observed clicks attribute to the performance difference between the two models.

In Figure 5.6, we show the click probability visualizations of the two most occurring click patterns in the dataset. Because the click probabilities are visually very similar, we report the log-likelihood performance of the individual session in the captions.



*(a) log-likelihood: GAN (-1.090), NCM (**-1.061**)* *(b) log-likelihood: GAN (-1.348), NCM (**-1.328**)*

Figure 5.6: The click predictions from the GAN and NCM model for the two most common click patterns. We see that the NCM model outperforms the GAN in log-likelihood in both examples.

From this figure, we can see that the NCM model is actually outperforming the GAN model for both click patterns. Moreover, in Table 5.3 we show how much these two click patterns attribute to the total log-likelihood performance of both models. This is demonstrated by showing the average log-likelihood performance for all sessions where this click pattern was observed, accompanied by how much these click patterns account for in the total dataset.

Model	*No clicks*	*Click first rank*	*Remaining sessions*
GAN	-1.160	-1.172	**-5.399**
NCM	**-1.129**	**-1.150**	-5.460
% of Sessions	29.6%	31.2%	39.2%

Table 5.3: The log-likelihood performance of the GAN and NCM models for the two most occuring click patterns and when these two patterns are taken out of the dataset.

From this table, we can see that the two most common click patterns account for approximately 61% of the total dataset. Moreover, for this 61% the GAN is actually being outperformed by the NCM models, which means that the GAN has to compensate for this decrease in performance using (part of) the remaining 39%.

To identify where the GAN compensates for this decrease in performance, we analyze the differences in performance in various other segments of the dataset. By doing so, we can highlight areas where the GAN has potentially improved or decreased its performance compared to the regular Neural Click Model. First, we analyze how both models differ in performance at predicting the correct click probability for each rank. In Figure 5.7, we show the increase in log-likelihood at each rank for either model segmented on whether a click was observed at the rank. The red line shows the total difference in performance at that rank, regardless of the observation. Positive scores show an increase in GAN log-likelihood performance and a decrease in NCM log-likelihood performance. For each segment, only the total increase in performance is reported for the best model.

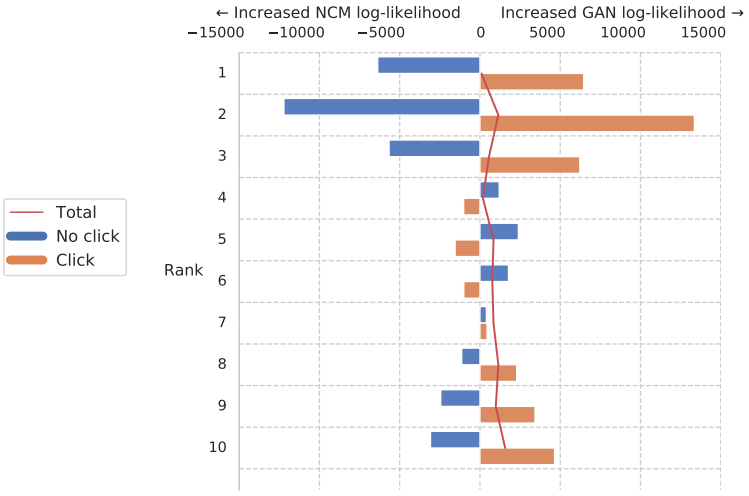
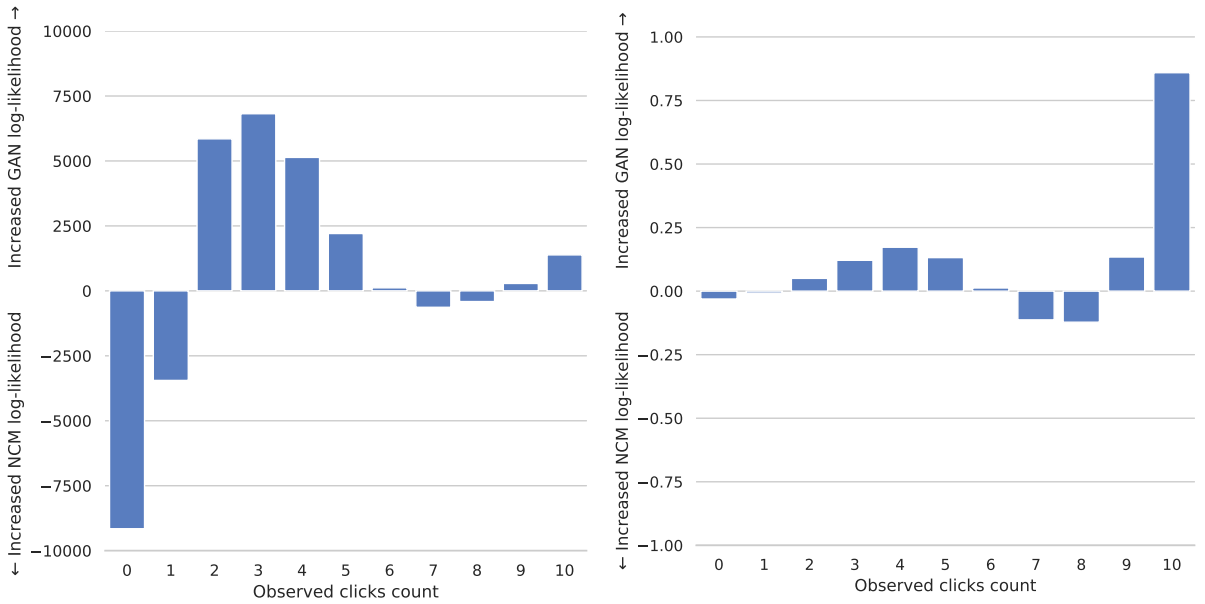


*Figure 5.7: The **total** log-likelihood performance difference between the GAN and NCM model at each rank. Negative values show total net improvements in log-likelihood for the NCM model, while positive number show total net improvements in log-likelihood for the GAN model. The bars segment documents in observed clicked and non-clicked document, with the red line showing the total performance difference at the given rank. Although not visible at each rank, the total difference is always in favor of the GAN model.*

In this figure, we can see that overall the GAN has increased performance at all ranks. However, we also see that at ranks $1, 2, 3$ and $7, 8, 9, 10$ the GAN gains performance when clicks are observed, indicating that the GAN is more confident in predicting clicks. The improvement at the first three ranks indicates that the GAN assigns a bigger position bias to these three documents. At ranks $4, 5, 6$ we see the opposite, where

the GAN is more confident that observing a click is less likely. The increased prediction confidence at each position increases the overall log-likelihood performance of the GAN, although it reduces performance when the opposite is observed. Moreover, at rank 7 the GAN seems to improve both its ability to predict a click and predict a non-click.

Additionally, we plot the total (Figure 5.8a) and mean (Figure 5.8b) difference of log-likelihood performance segmented in the amount of observed clicks per sequence.



*(a) The **total** difference in log-likelihood performance per amount of observed clicks.*

*(b) The **mean** difference in log-likelihood performance per amount of observed clicks.*

Figure 5.8

From this figure, we observe that the NCM model only has a relatively small average performance improvement for sequences with one or zero clicks. However, these sequences still account for the biggest performance increase in favor of the NCM models, because these two segments represent 46.2% and 29.1% of the total dataset. However, the GAN gains most of its performance by improving the predictions for sequences with $2, 3, 4, 5$ or 10 clicks. Because these five segments only represent 22.1% of the total dataset, the GAN gains a relatively high relative performance increase for these segments as can be seen in Figure 5.8b. These results are confirmed by the overall predicted click probability from the GAN, which shows that the GAN is 2.2% more likely to predict a click over a non-click than the NCM model (Appendix Figure B.2).

To conclude, we answer RQ_4 by summarizing the differences highlighted in this section. We addressed the difference in performance by segmenting the dataset by both the most occurring click patterns, and the ability of each model to predict the click probability both at each rank and given the amount of observed clicks. From these results, it shows that the GAN model spreads its predictions across a more diverse set of patterns, whereas the NCM model weights its predictions more heavily on the most frequently observed click patterns. Moreover, because the dataset is very skewed towards sessions with either no or one observed click, we observe an increase in performance for the GAN in the long tail of sessions with more than one observed click.

Chapter 6

Conclusion & Future Work

6.1 Conclusion

In this work, we demonstrated that existing click models can easily be adjusted to be trained with relevance labels. Although other features such as position and previous click observations are more important indicators of click probability, the relevance labels do prove to be a distinct and useful input feature. Moreover, we introduce a method that uses the relevance labels to highlight the patterns learned by the click models, which show that some click models learn various complex biases and patterns expected in real user behavior.

Additionally, we introduce a neural click model based on GANs. With this model, we show that by adding an adversarial loss we can more accurately predict click patterns than using a log-likelihood loss alone. However, because the observed click patterns are very skewed, we see that the log-likelihood loss is outperforming the GAN model on the most common click patterns, while the GAN can compensate for this with on the more long-tailed patterns.

Choosing the optimal relevance-based click model depends on the application. When it is important to produce high-quality click patterns that reflect real user behavior, then the GAN and NCM models are clear winners. However, even though the UBM model is unable to model complex user behavior, it does seem to capture the most prominent patterns. Because the UBM model is much more lightweight and trivial to implement, it is worth considering when speed and simplicity are important.

6.2 Future Work

In this section, we will reflect on some of the limitations and promising directions from this thesis which could be addressed in future work.

One of the limitations that this work has been set out to address, was enabling click models to be used on previously unobserved documents. However, collecting relevance labels still requires a document to be seen by one or more users or experts. Future work could further address this limitation by using a dataset which replaces the relevance labels with features describing the document and query content. Although the PGM-based click models are not designed to work with a wide variety of input features, the NCM and GAN models could easily be adjusted by expanding the input features. Future work could also use the same visualization techniques used in this thesis to understand the influence of the various added features. However, it might be worth using more sophisticated feature importance highlighting tools such as [38] when many features need to be analyzed.

Future work could also apply the relevance-based click models to other variations of click models. For example, relevance-based click models could be used to predict click sequences that are not constrained by users going from top to bottom, similar to the click sequence models as described by Borisov et al. [7]. Moreover, the click models describe in this work are restricted to SERP layouts where the results are presented vertically. Using relevance-based click models for other SERP layouts, such as horizontal, as described by Oosterhuis et al. [35] could potentially help to interpret the user behavior.

Finally, we show that using an adversarial loss outperforms using a log-likelihood loss for predicting user clicks. These adversarial loss methods could potentially be used beyond predicting these click probability and directly optimizing underlying ranking algorithms.

Bibliography

[1] Bram van den Akker, Ilya Markov, and Maarten de Rijke. Vitor: Learning to rank webpages based on visual features. In *Proceedings of the 2019 World Wide Web Conference (WWW 19)*, 2019.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223, 2017.

[3] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179, 2015.

[4] James Bergstra, Daniel Yamins, and David Daniel Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *JMLR*, 2013.

[5] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[6] Alexey Borisov, Ilya Markov, Maarten de Rijke, and Pavel Serdyukov. A neural click model for web search. In *Proceedings of the 25th International Conference on World Wide Web*, pages 531–541. International World Wide Web Conferences Steering Committee, 2016.

[7] Alexey Borisov, Martijn Wardenaar, Ilya Markov, and Maarten de Rijke. A click sequence model for web search. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 45–54. ACM, 2018.

[8] Olivier Chapelle and Ya Zhang. A dynamic bayesian network click model for web search ranking. In *Proceedings of the 18th international conference on World wide web*, pages 1–10. ACM, 2009.

[9] Olivier Chapelle, Donald Metlzer, Ya Zhang, and Pierre Grinspan. Expected reciprocal rank for graded relevance. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 621–630. ACM, 2009.

[10] Weizhu Chen, Dong Wang, Yuchen Zhang, Zheng Chen, Adish Singla, and Qiang Yang. A noise-aware click model for web search. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 313–322. ACM, 2012.

[11] Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. Click models for web search. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 7(3):1–115, 2015.

[12] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[13] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. An experimental comparison of click position-bias models. In *Proceedings of the 2008 international conference on web search and data mining*, pages 87–94. ACM, 2008.

[14] Georges E Dupret and Benjamin Piwowarski. A user browsing model to predict search engine click data from past observations. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 331–338. ACM, 2008.

[15] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*, 2017.

[16] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[17] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[18] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[19] Artem Grotov and Maarten de Rijke. Online learning to rank for information retrieval: Sigir 2016 tutorial. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 1215–1218. ACM, 2016.

[20] Artem Grotov, Aleksandr Chuklin, Ilya Markov, Luka Stout, Finde Xumara, and Maarten de Rijke. A comparative study of click models for web search. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 78–90. Springer, 2015.

[21] 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*, pages 124–131. ACM, 2009.

[22] Katja Hofmann, Shimon Whiteson, and Maarten De Rijke. Balancing exploration and exploitation in learning to rank online. In *European Conference on Information Retrieval*, pages 251–263. Springer, 2011.

[23] Katja Hofmann, Shimon Whiteson, and Maarten De Rijke. A probabilistic method for inferring preferences from clicks. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 249–258. ACM, 2011.

[24] Ferenc Huszár. How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *arXiv preprint arXiv:1511.05101*, 2015.

[25] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[26] Thorsten Joachims, Laura A Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *Sigir*, volume 5, pages 154–161, 2005.

[27] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. Unbiased learning-to-rank with biased feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 781–789. ACM, 2017.

[28] Donald R Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization*, 21(4):345–383, 2001.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[30] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *ICML*, 2017.

[31] Dhruv Madeka, Lucas Swiniarski, Dean Foster, Leo Razoumov, Kari Torkkola, and Ruofeng Wen. Sample path generation for probabilistic demand forecasting. *Github*, 2018.

[32] Rishabh Mehrotra, James McInerney, Hugues Bouchard, Mounia Lalmas, and Fernando Diaz. Towards a fair marketplace: Counterfactual evaluation of the trade-off between relevance, fairness & satisfaction in recommendation systems. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 2243–2251. ACM, 2018.

[33] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[34] Harrie Oosterhuis and Maarten de Rijke. Differentiable unbiased online learning to rank. In *Proceedings of the 2018 ACM on Conference on Information and Knowledge Management*. ACM, 2018.

[35] Harrie Oosterhuis and Maarten de Rijke. Ranking for relevance and display preferences in complex presentation layouts. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 845–854. ACM, 2018.

[36] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[37] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016.

[38] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3145–3153. JMLR. org, 2017.

[39] Andrew Turpin, Falk Scholer, Kalvero Jarvelin, Mingfang Wu, and J Shane Culpepper. Including summaries in system evaluation. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 508–515. ACM, 2009.

[40] Ellen M Voorhees, Donna K Harman, et al. *TREC: Experiment and evaluation in information retrieval*, volume 63. MIT press Cambridge, 2005.

[41] Yizhe Zhang, Zhe Gan, and Lawrence Carin. Generating text via adversarial training. In *NIPS workshop on Adversarial Training*, volume 21, 2016.

[42] Xingyu Zhou, Zhisong Pan, Guyu Hu, Siqi Tang, and Cheng Zhao. Stock market prediction on high-frequency data using generative adversarial nets. *Mathematical Problems in Engineering*, 2018, 2018.

Appendices

Appendix A

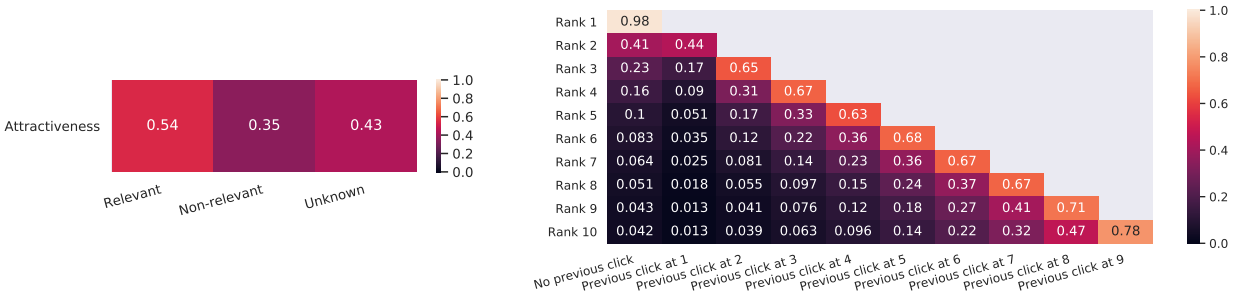
PGM-Based Click Model Parameters

In this appendix, we highlight the parameters learned by each of the PGM-based click models. These parameters can be used to both interpret and apply the click behavior learned by the various models.

Rel-CTR

Figure A.1: The relevance-based parameters learned by the CTR model. The click probabilities reflect a clear difference between the relevant, non-relevant and unknown relevance labels.

UBM

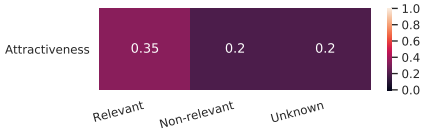


*(a) The **attractiveness** parameter learned by the UBM model. We see that relevant documents are more likely to be clicked than non-relevant and unknown documents.*

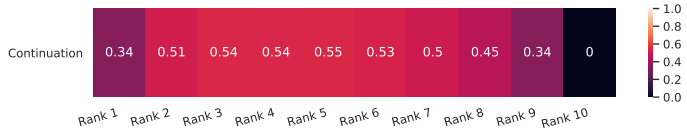
*(b) The **examination** parameter learned by the UBM model. We can see that the highest probabilities are assigned to documents where the previous documents were clicked. After each rank without an observed click, the probabilities steadily decrease.*

Figure A.2: The attractiveness and examination parameters of the UBM model.

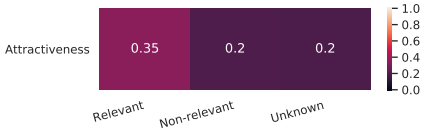
DCM



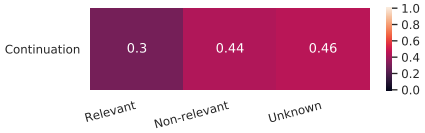
*(a) The probability of clicking an examined document (**attractiveness** parameter) based on the available relevance labels, as learned by the relevance and rank-based DCM model.*



*(b) The probability of continuing to examine the next document after each rank (**continuation** parameter), as learned by the relevance and rank-based DCM model.*



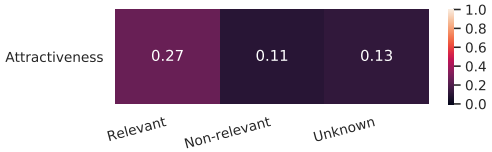
*(c) The probability of clicking an examined document based on the available relevance labels (**attractiveness** parameter), as learned by relevance-only-based DCM model.*



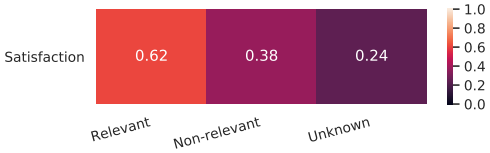
*(d) The probability of continuing to examine the next document (**continuation** parameter) after examining a document with given relevance label, as learned by the relevance-only-based DCM model.*

*Figure A.3: The **attractiveness** and **continuation** parameters learned by the DCM model either by using both relevance labels and document rank (Figure A.3a and A.3b) or the relevance labels for both parameters (Figure A.3c and A.3d). In this thesis, we use the DCM model with both relevance and rank-based parameters, because with a Log-likelihood of -3.533 it outperforms the other DCM variant with a Log-likelihood of -3.511*

DBN



(a) The attractiveness parameter learned by the DBN model.



(b) The satisfaction parameter learned by the DBN model.

Figure A.4: The Attractiveness and Satisfaction parameters learned by the DBN model. We omitted from showing the continuation parameter, as it actually gave a continuation probability of 1. Instead, the model solely relies on the attractiveness and satisfaction parameters.

Appendix B

Additional Results

Model	Accuracy	AUC	Log-likelihood
Click First Rank	87.1%	0.661	−10.634
Click None	**87.6%**	0.5	−8.472
Rank-CTR	81.5%	0.756	−3.160
GAN	82.8%	**0.839**	**−2.825**
NCM	83.0%	0.838	−2.833
UBM	82.6%	0.832	−2.877
DCM	78.8%	0.748	−3.393
DBN	79.5%	0.728	−3.541
Rel-CTR	78.9%	0.543	−3.690

Table B.1: The Accuracy compared to the AUC and Log-likelihood performance for each of the relevance-based click models and baselines. These results highlight that the highest accuracy is achieved by always predicting that no click will be observed.

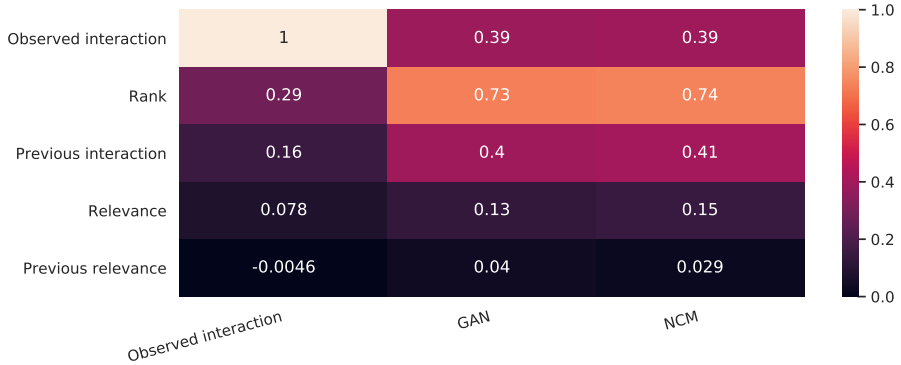
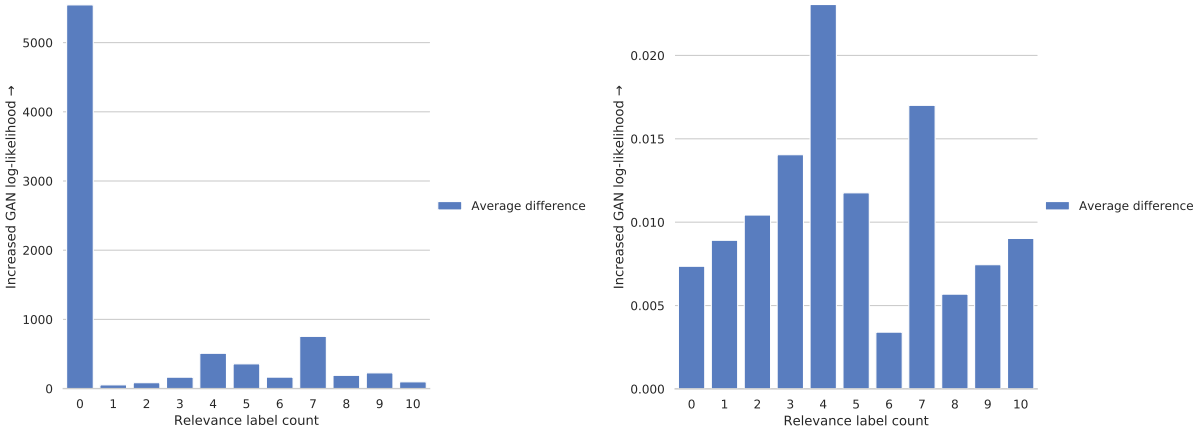


Figure B.1: The correlation between various input features and the predicted click probability by both the GAN and NCM model. For reference, we also added the correlations between the input features and the actual click observed. The figure shows that the correlations found in the NCM and GAN model are very similar.

Model	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6	Rank 7	Rank 8	Rank 9	Rank 10
NCM	0.4526	0.1843	0.1276	**0.1022**	**0.0827**	**0.0679**	**0.05667**	0.04895	0.04272	0.03830
GAN	**0.4644**	**0.1986**	**0.1339**	0.1012	0.0809	0.0663	0.05655	**0.05030**	**0.04519**	**0.04178**

Table B.2: The average click probability at each rank for both the NCM and GAN model. These click probabilities agree with the gain in performance per rank as seen in Figure 5.7.



*(a) The **total** difference in log-likelihood performance per amount of available relevance labels.*

*(b) The **mean** difference in log-likelihood performance per amount of available relevance labels.*

Figure B.2: The gain in log-likelihood performance for the GAN model for the various available relevance counts per sequence. The graphs show that the GAN outperforms the NCM model, regardless of the number of relevance labels.

Appendix C

Additional Dataset Information

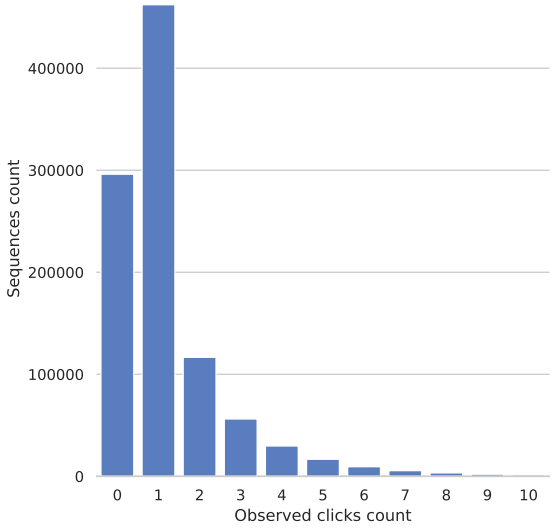


Figure C.1: The distribution of click counts per sequence in the test set. We can see that most sequences either observe one or no clicks.

Figure C.2: The correlations between each of the individual features of the dataset. This shows that features such as example relevance and position, and relevance and previous relevance are already correlated.