MASTER'S THESIS 2022

Personalizing the Order of Search Results Using Machine Learning

Liam Fahlstad, Max Gustafson

ISSN 1650-2884
LU-CS-EX: 2022-39
DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2022-39

Personalizing the Order of Search Results Using Machine Learning

Personalisering av sökträffars ordning
m.h.a. maskininlärning

Liam Fahlstad, Max Gustafson

Personalizing the Order of Search Results Using Machine Learning

Liam Fahlstad
li3576fa-s@student.lu.se

Max Gustafson
ma2565gu-s@student.lu.se

June 23, 2022

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Emelie Lundh, emelie.lundh@ingka.ikea.com
Pierre Nugues, pierre.nugues@cs.lth.se

Examiner: Flavius Gruian, flavius.gruian@cs.lth.se

Abstract

With the growth of the internet, search algorithms such as ‘Google’ have been developed which help the user navigate the web. The high standard set by such search engines is transferred to enterprise internal search, which unfortunately is often inferior.

One aspect that is utilized extensively by commercial search engines, but rarely by large companies, is personalization. Together with IKEA, we study the effect of personalization in enterprise search by modeling a *reranker*. Such a reranker reorders the result list returned by the search engine to place documents relevant to the user towards the top. The reranker is built using supervised machine learning which is trained using the ‘learning-to-rank’ algorithm’ *LambdaMART*.

We implemented this algorithm using three frameworks. These frameworks are *XGBoost* (Xtreme Gradient Boosting), *LightGBM* (Light Gradient Boosting Machine), and a *neural network* on two different datasets, one from IKEA and one public from Yandex, a search engine popular in eastern Europe, used in a Kaggle competition. For the IKEA dataset, there was a significant improvement, while for Yandex there was a slight one. In particular, there was an improvement in our main metric ***NDCG*** (Normalized Dicounted Cumulative Gain) of ~1.3% for Yandex and ~5.2% for IKEA.

Keywords: Personalization, Search, Machine Learning, LambdaMART, Learning to Rank

Acknowledgements

> “Happiness is not reaching your goal. Happiness is being on the way.”
> **– Ingvar Kamprad**

Firstly, we would like to thank all our coworkers at the INGKA ‘Internal Comms. Team’ for welcoming us as thesis students. In particular, we would like to thank Fredrik Ekström and Johannes Sörensen for helping us with the data collection. An additional thanks goes to Johannes for all the help he provided. Finally, a special thanks goes to Emelie Lundh for supervising and supporting us throughout this thesis.

Secondly, thanks also go to all our friends and family supporting us, and giving input, during our work.

Lastly, we would like to thank our LTH supervisor Pierre Nugues for all the advice and assistance he provided.

Contents

Glossary

click When a user has issued a search query, they can chose to click a result. 18, 19

document A document is, in an more general sense, something that can be indexed by the search engine and can be clicked by user in response to a search query. URL is an example of a document. 19

domain The domain of the URL e.g. `www.lth.se` is a domain while `https://www.lth.se/utbildning/teknisk-matematik/` is a URL. 19, 28

query A query is one or more words which is used by the search engine to retrieve information and is what the user enters into the search-field. 18, 19

SERP The search engine is indexing a set of documents returning a list to the user. This list is called a result page or SERP (Search Engine Result Page) abbreviated. 12, 19, 20, 25, 26, 28, 53, 54, 57, 59, 60, 63

session A session is a sequence of actions where a user issues one or more queries and may click on one or more results in response. A user may have multiple sessions, but a session does not extend across users.. 18, 19

term A query is built up by one or more terms. The query 'Java Coffee' consists of terms 'Java' and 'Coffee'. 19

URL A link that i.e. a reference to a web-source or a document in a database. An example is `https://www.lth.se/utbildning/teknisk-matematik/`. 18, 19, 35

user A user can be seen as an agent that issues search queries and clicks on results i.e. someone who uses the search engine. 19

Chapter 1

Introduction

In the introduction, we provide the background for our work. In addition, we present related literature as well as a summary of our findings. Lastly we provide the specification of contribution from both authors.

1.1 Background

1.1.1 General

The last years have seen an expansion of the internet of unquestionable magnitude. As such, the sheer amount of available information has put heavy demands on commercial search engines. Some very sophisticated search engines, such as ‘Google’, have emerged with high user satisfaction. This, in turn, has led to higher expectations for enterprise search in internal databases and file systems. Unfortunately, when people use search at work, it is more difficult to find the relevant information when compared to a commercial search platform.

One aspect that is utilized extensively by commercial search engines, but rarely by companies, is *personalization*. By using information about previous searches, an engine can present a personalized result relevant to the specific user. Companies may not be able to implement this approach due to a lack of data. However, large companies, such as IKEA with ~160,000 coworkers, may have enough data to derive personalization models from previous queries. While the amount data per user may still be small, personalization can still be applied by looking at related users e.g. field of work, country etc. We can then wonder how to build models so that search at work becomes more similar to commercial search engines using personalization.

To perform a search, a user writes a search query consisting of up to a few words, and the search engine returns a *search engine result page* (SERP) which is a list of related documents. However, the order in the list might be non-optimal and relevant results might be lost at the bottom of the list. Personalization can be applied to move documents more relevant to the user up to a higher rank. Earlier research has shown that the position of a document has an impact on the "consumers click behavior" in e-commerce (Ursu, 2018). In essence, presenting relevant documents at positions higher in the list increases the probability that the user clicks the link. The worst case is that the user abandons the session by simply not clicking any document.

The task seems deceivingly simple: ranking the documents and returning them in order of rank gives the optimal solution. While this is true for users with *identical preferences*, usually different users expect different results when providing the same search query. The preferences are *heterogeneous* as initially described by de Vrieze (2006). A user presenting the search query "Java" might be interested in:

1. Java coffee,
2. the programming language Java,
3. a vacation to the Java island.

As such, as explained by Yoganarasimhan (2020), the relevance of the presented documents is user-specific. The framework presented needs to take this into account.

1.1.2 Problem formulation

With this background, the goal of this thesis is to study the effect of personalization using a *reranker*. Such a reranker ranks a list of documents in the order most relevant to the specific user issuing the search query. A schematic example of a reranker is shown in Figure 1.1. It takes the original list produced by the search engine, processes it through a model, which returns a new order based on the particular user's preferences.

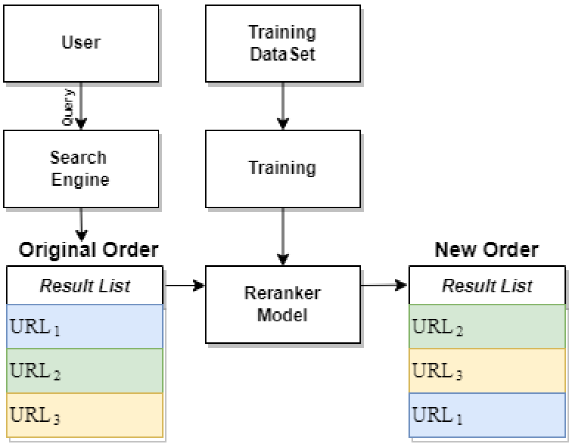


Figure 1.1: A schematic example of a re-ranker.

In particular, we can formulate reranking as a binary categorization problem between documents. Given two documents, we want to predict which documents is more relevant to the user and rank that one higher. In the case of a whole result list, this is done for all possible pairs.

In this thesis, we explore and implement a machine-learning framework to power the reranker. We evaluated the resulting models using multiple metrics with support in literature.

The reranker should operate in real time, thus efficiency is an aspect we consider important. Efficiency is the notion of returning a good reranking given a list within milliseconds, as required by IKEA. Since we use large datasets and we need to extract a lot of information, time complexity should be kept in mind when developing the framework.

To summarize:

- **Main goal:** explore if search experience at work can be improved using *personalization*. In particular, is it possible to create a *reranker* using 'learning to rank' algorithms, a supervised machine learning problem for *ranking* documents, to serve this purpose. It is considered an improvement if relevant documents, for a particular user, appears higher in a reranked result page compared to the original one.
- As requested by IKEA, the program and model should be *accurate*, *time-efficient* and *scalable*. We hence set the following **sub-goals:**
  - compare different training algorithms and resulting models,
  - compare different machine learning framework,
  - implementation is scalable in the sense that it should run in linear time,
  - investigate what information the models deem relevant.

1.1.3 Findings

In this thesis, we present a machine learning framework which can be used for reranking. We first analyzed the state of the art in reranking. From this analysis, we concluded that the **LambdaMART** algorithm obtained the best accuracy, while still being efficient. LambdaMART solutions have won many Learning to Rank challenges, such as the 'YAHOO! Learning to Rank Challenge (Track 1)' (Chapelle and Chang, 2011). We thus selected this algorithm, based on boosted regression trees, for our subsequent implementations.

To implement LambdaMART, we evaluated two state-of-the-art implementations of gradient boosting machines, **XGBoost** and **LightGBM**. A **neural network** was implemented as comparison.

In Chap. 3, we describe the LambdaMART algorithm. We then describe the models we trained using the frameworks *XGBoost* and *LightGBM*. These models are used to rerank documents for personalized search. The full models includes 500+ *features* and a suggestion how to prune a full model to avoid overfitting and increase efficiency is further provided.

In this thesis, we used three datasets on which we applied our models: one publicly available and two from IKEA. To measure their performance, we used the Normalized Discounted Cumulative Gain (***NDCG***) metric, Mean Reciprocal Rank (***MRR***), Average Error in the Rank of a Click (***AERC***) and Click Through Rate ***CTR***. For our LambdaMART models, we observed an improvement in all the metrics. We saw an increase in the main metric $NDCG$ by 1.3% on the public `yandex_dataset` from a *Kaggle* competition, which motivated the validity of our model since a top competitor scored 2.3% using similar labels as us, on much more data. On the two IKEA datasets, we improved $NDCG$ by 5.2% for one dataset (`entrypoint_dataset`), and deemed the other too small to properly evaluate. We also saw an increase in the other metrics such as MRR, $AERC$ and CTR. In particular, the LambdaMART implementations outperformed the Neural Network on all datasets.

1.2 Previous Work

Before building a personalization model, it is important to understand when and how to do personalization. Dou et al. (2007) present a large-scale framework, where the authors show that personalized search indeed improves search engines. In particular, click-based personalization (i.e. dependent on a user's click history) is more effective than user-interest or group-interest personalization. However, not every query benefits equally from personalization. Some queries might even be harmed. An important notion is *click entropy*. A query can only effectively benefit from personalization if different users click on many different results. A query such as 'Google' leads in a large majority to clicks at `www.google.com`, no matter the number of users. This is what is known as a 'navigational query' and personalization is insufficient for such queries. This is important to consider when we evaluate the datasets we use in our thesis.

The notion of 'learning-to-rank algorithms' (LTR) is an application of machine learning in the construction of ranking models, where the goal is to rank documents in a list. Burges (2010) introduced the theory of *LambdaMART*, the state-of-the-art machine learning framework used for learning-to-rank problems. This is applicable to personalization since we want to rerank based on user specific information. The author wrote a concise report about *LambdaMART* and the algorithms leading up to it. In particular, the mathematical aspect of the algorithms is discussed. In addition to explaining the mathematical baseline, the author motivates the machine learning specifics in *LambdaMART* such as using boosted regression trees when ranking as well as which metric to use when we evaluate a list of documents. *LambdaMART* is state-of-the-art in reranking, and we will compare implementations of it in our thesis. Burges (2010) report is important to understand the algorithms underlying mathematics.

Yoganarasimhan (2020) provides implementation details for *LambdaMART* and discusses the implementation of a program which outperformed the winner in Kaggle's *Personalized Web Search Challenge* (Kaggle, 2013). The author discusses how data can be partitioned into 'past data' serving as information on earlier searches by users in addition to train, validation, and test data. Yoganarasimhan (2020) also presents a baseline of *features* to use when personalizing search, many of which are used in this thesis. Since one of the datasets we used is a smaller version of the dataset used by the author, we have used some of her features, adding index related features as explained in Section 2.2.3. For the data provided by IKEA, we have extended these features to utilize the additional information provided. Further, evaluation metrics are discussed. In particular $NDCG$ (Normalized Discounted Cumulative Gain), $\Delta CTR@p$ (change in click through rate at position p) and $AERC$ (Average Error in the Rank of a Click) are introduced. These are metrics which the author has collected through a literature study, which proved their superiority when evaluation Learning-to-rank algorithms. As such, we will use the same metrics. Lastly, the report analyzes the dataset in the Kaggle competition and discusses what type of searches can/should be personalized. Further, we use the same evaluation metrics as the report and analyze the data in a similar way. Once again, this is because one of our datasets is a smaller version of the data used by Yoganarasimhan (2020).

When *XGBoost* (Xtreme Gradient Boosting) was released in 2016, it was the state-of-the-art implementation of *LambdaMART*. In their paper, Chen and Guestrin (2016) present the approach used when implementing *XGBoost*. In essence, *XGBoost* uses:

> a novel sparsity-aware algorithm for sparse data and weighted quantile sketch for approximate tree learning.

Further, the paper discusses cache access patterns, data compression and sharding. All used to “build a scalable tree boosting system”. from *LambdaMART*. This implementation’s Python API will be used in this report to train our *LambdaMART* model, since it was found to be considered state-of-the-art (Ye, 2020).

One disadvantage of *XGBoost* is that it only handles numerical data, with only experimental support for categorical values. In particular, the data provided by IKEA contains much information about users in the form of categorical values. While one-hot-encoding can be used, an alternative implementation is *LightGBM* (Ke et al., 2017) which supports *categorical features*. It is an improvement on *XGBoost* developed by Microsoft. In addition to handling categorical features, it is lightweight with high performance.

Lastly, an alternative family of machine learning algorithms used to learn how to rank are the neural networks. While training and predicting generally are slow and not used for reranking Nardini et al. (2022) provide an example where a neural network implementation has been used for reranking. While not explored in this thesis, we have compared our LambdaMART implementations to a multi-layer perceptron regressor using sklearn’s **neural network** package. This report is thus important for future work since the authors claim their neural network outperforms LambdaMART implementations, which differs from our result.

1.3 Specification of Contributions

The workload was split equally, but each author focused on different aspects. Max suggested the LambdaMART algorithm, after conducting an initial literature study. Liam suggested the *XGBoost* framework and after discussion between the authors, we decided to use this as our main framework. After suggestions from our supervisor, we also studied *LightGBM* and a *neural network*.

While Max initially wrote code for parsing the dataset, Liam revised and improved it. The process of revising each others code was present throughout this thesis. However, Liam focused more on parsing the dataset to the correct format for *XGBoost* and adding features, while Max focused on analysis of the dataset and parsing data to a structure for *LightGBM*. As before, both authors contributed to each others parts.

Considering the report, Max wrote much of Chap. 1 and 4, while Liam wrote most of Section 2.2. We contributed equally to any chapter or section not mentioned. The entire report was revised by both of us, and many sections were extensively discussed between us and thus produced together.

Chapter 2

Approach

In this chapter, we describe our approach i.e. all decisions made and the motivation behind them. This chapter is divided into these following sections:

1. 2.1 where we describe the datasets, their background and motivation for using them, as well as introducing necessary definitions,

2. 2.2 where we describe the outline of the models, how that datasets and frameworks where used to create them as well as how to evaluate our models. We also discuss our implementation of our program.

2.1 Outline and Definitions

In this thesis, we have worked with three separate datasets. Two of them are supplied by IKEA and the third is used as a baseline to validate our model. This is further discussed in the preceding section 2.1.1. The datasets are:

1. a publicly available dataset consisting of anonymized data from **Yandex**, one of eastern Europe's largest search engines. This dataset was provided for the *Kaggle web search personalization competition* (Kaggle, 2013). We call this dataset the `yandex_dataset`.

2. At IKEA, we worked on a project involving two platforms **EntryPoint** and **IntraNet**.

   IntraNet is a search engine for internal documents stored in sharepoint; we call this dataset the `IntraNet_dataset`;

EntryPoint is an aggregating engine for disparate sources; such as the products and people databases. Notably, EntryPoint indexes IntraNet, thus they share some documents. We call this dataset the `entrypoint_dataset`.

Although these two platforms share some of the documents, they present them in different ways. We have therefore decided to train two different models, one for each platform. The differences are further discussed in Sects. 2.1.2 and 2.1.3.

The models created from `yandex_dataset`, `intranet_dataset` and `entrypoint_dataset` are called `yandex_models`, `intranet_models` and `entrypoint_models` respectively.

2.1.1 Why both Yandex and IKEA

While the goal is to improve search at IKEA, the improvements made are speculative in the sense there is no point for reference of how much search can be improved. By incorporating data from a personalization competition (Kaggle, 2013), we can compare the performance of our models to those in the competition. We can therefore get an indication of how well the model performs. Then, if the model performs well, we posit that we can use the same method to create a model for the IKEA dataset.

Another issue is that we cannot disclose the IKEA datasets in this report, as they contain sensitive information. It is therefore essential to include Yandex for the reader.

2.1.2 The Yandex Dataset

The dataset from Yandex contains 27 consecutive days of data from the year of 2011 and is provided as a .txt file. Exact information about the dates is not disclosed. However, it is disclosed that the data is sampled from one large city and is preprocessed in two ways:

1. "sessions containing queries with commercial intent detected with Yandex proprietary classifier are removed",
2. "sessions with top-K most popular queries are removed. K is not disclosed". The exact method of defining sessions used by Yandex is also not disclosed (Kaggle, 2013).

The data is fully anonymized in the sense that queries, URLs etc. have been converted to anonymous numbers. The data is further sorted by sessions. In total, the dataset contains 34,573,630 sessions.

Table 2.1 shows an example of a session. The first row contains meta-data about the search, as indicated by the "M". It tells us about the sessionID (7), which day the session occurred (15) and the userID (1). In the session, rows with "Q" denotes queries and rows with "C" denotes clicks. This example consists of the following steps:

1. The user issued query with id 2685901, which contains the terms with ids 1403091, 1576407, and 709728, at the start of the session. Number 0 under column head M means that the session has just started.

7	M	15	1						
7	0	Q	0	2685901	1403091,1576407,709728	19070833,1994676	19070785,1994663	3334821,451714	
						26466527,2597528	28891144,2834037	46415647,3967156	
						7807541,943850	68619915,5138944	62948695,4828867	49680334,4165241
7	23	Q	1	2686894	1403353	53533848,4359008	62946524,4828867	3334825,451714	
						28881052,2834018	56212472,4467930	39918555,3595774	
						56177047,4466491	28858835,2833991	59532827,4640381	7981226,945625
7	33	C	1	53533848					

Table 2.1: Example session from the Yandex dataset.

2. In response ten (URL ID, Domain ID) combinations are provided with (19070833,1994676) at the top of the list. The query has index 0, indicated by the fourth column.

3. The user does not click any result but instead issues another query with ID 2686894 with term 1403353, 23 time units into the session which produces ten new (URLID, DomainID) combinations. This query has index 1.

4. Finally, 33 time units into the session, the user clicks URL with ID 53533848 placed at the top of the list in response to query with index 1.

As we are restricted to run the code on our laptops supplied by IKEA, the full `yandex_dataset` will not be used as it requires too much computational power. We thus used a subset of the full dataset, referred to as the `red_yandex_dataset` (reduced Yandex dataset) which includes the first 10^6 lines of data of the .txt file, containing the `yandex_dataset`. Table 2.2 shows some statistics about the `red_yandex_dataset`.

The Yandex dataset contains users, queries, terms, URLs, domains, clicks and sessions. In addition, a date indicating the day (1 -27) was provided. For sessions, *session start-time* and *in session time* is disclosed but not the time-unit.

Notably, after a query is issued, Yandex only provides data for the first search engine result page (SERP) since most users do not go beyond the first page.

2.1.3 Datasets: IKEA

Information about the search is collected and stored in the same database for EntryPoint and IntraNet. In particular, data is collected every time a user triggers an event, which in our case would be *clicking a link*. Similarly as in the Yandex dataset, IKEA collects information about users, queries, documents and clicks as defined in the glossary. Domains and terms can be extracted. Since we mainly index databases of documents, URLs and documents are used interchangeably when considering the IKEA datasets. The start of every session and every click is paired with a UNIX-timestamp. Sessions are reported differently to Yandex since data is stored when clicking a link at IKEA while Yandex stores data every time a session is started.

However, Yandex does not disclose its definition of a session. Thus what is considered a session will differ between datasets. In contrast to Yandex, the only anonymized information in the IKEA dataset is the userID. While the ID is anonymized, specifications about the user are not. In particular, the dataset contains the following information about the user:

Country: which country the user is located in,

Business Unit: which business unit the user is located in, i.e. if the user works in IT, management, department store, etc.,

Work Title: self defined work title by the user,

Company: what part of IKEA the user works at and is based on location,

Site: what work site the user is mainly located at e.g. Hubhult-Malmö, Barkarby Store-Stockholm,

The IntraNet Dataset. IntraNet presents a SERP as a single ordered list of documents, similarly to that of Google, Yandex, and other commercial search engines. The documents are stored in *SharePoint*, a website-based collaboration system, to store, search, and interact with documents.

One characteristic specific to IntraNet is that of *promoted* documents. These are documents added by hand by the developers, providing the most frequently sought information for certain popular queries. There could be multiple promoted documents, but as a general rule, there is only one. When a promoted document is present in the SERP, we hypothesize that users almost always click the promoted document. Due to an inconsistency in the IntraNet dataset, we could not extract reliable data on the number of clicks on promoted documents. This does not need to be accounted for it is not considered when making predictions for IntraNet. However, in the EntryPoint dataset SERPs with one or more promoted documents resulted in a click on a promoted document 96% of the time.

The EntryPoint Dataset. When a user issues a query in EntryPoint, a list of K documents is presented. In contrast to IntraNet, EntryPoint presents its results in batches, where each batch consists of indexed documents corresponding to a certain *category*. The category is an important aspect of EntryPoint that heavily influences the user experience.

The results in each category are indexed differently depending on the origin of the document and several documents are presented for the most *relevant* categories. If the user clicks "load more results", a new search is made and new documents are indexed and added to the list in the given category. As a general rule, only 3 results are initially displayed in each category. An example of the structure is presented in Fig. 2.1 and 2.2.

The category "Inside" corresponds to the result from IntraNet. These results, documents and order, would the same as those provided by a search with IntraNet. Only the number of documents shown would change. As such, promoted links are also present in *EntryPoint*.

Table 2.3 shows statistics about the IKEA datasets. We provide this data to give an understanding about the size of each dataset, and compare them to each other and the `red_yandex_dataset` in Table 2.2. Note that the `entrypoint_dataset` is contain more users than the `intranet_dataset` and tha `red_yandex_dataset`, but the `red_yandex_dataset` contains more queries.

2.1.4 Quantities

We introduce some different notations that we will use in the rest of this thesis:

Q : the list of all the queries issued in the dataset, $Q = \{q_1, q_2, ...\}$

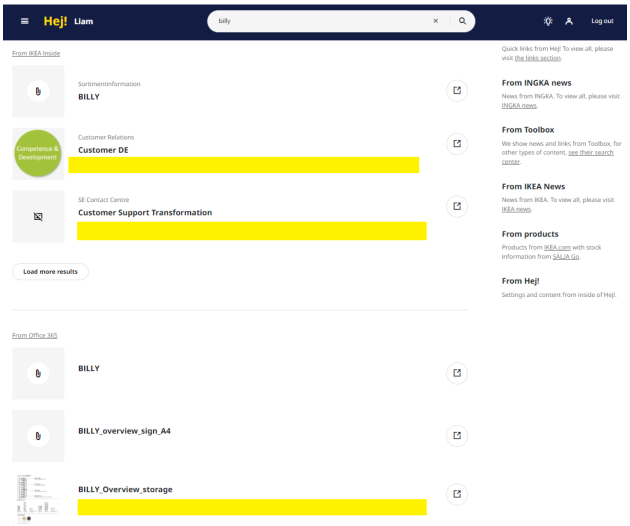
D_i : all the documents, $D_i = (d_i^1, d_i^2, ..., d_i^K)$, that appeared as the result of query $q_i \in Q$.

Dataset:	Number of Users	Number of Sessions	Number of Queries	Number of Unique Queries
Yandex	34,656	212,553	356,682	236,010

Table 2.2: Statistics about the reduced *Yandex* dataset.

Dataset:	Number of Users	Number of Sessions	Number of Queries	Number of Unique Queries
IntraNet	13,814	36,157	36,157	11,846
EntryPoint	57,884	260,596	299,712	50,206

Table 2.3: Statistics about the two IKEA datasets *IntraNet* and *EntryPoint*.



Hej! Liam

billy

Log out

From IKEA Inside

Sortimentinformation
BILLY

Customer Relations
Customer DE
[redacted]

SE Contact Centre
Customer Support Transformation
[redacted]

Load more results

From Office 365

BILLY

BILLY_overview_sign_A4

BILLY_Overview_storage
[redacted]

Quick links from Hej! To view all, please visit the links section.

From INGKA news
News from INGKA. To view all, please visit INGKA news.

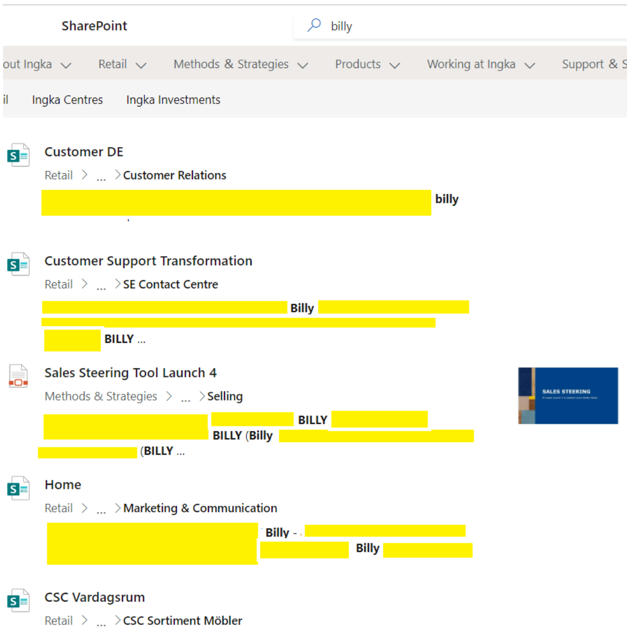
From Toolbox
We show news and links from Toolbox, for other types of content, see their search center.

From IKEA News
News from IKEA. To view all, please visit IKEA news.

From products
Products from IKEA.com with stock information from SÄLJA Go.

From Hej!
Settings and content from inside of Hej!.

Figure 2.1: Example of result page in EntryPoint for query "Billy".



SharePoint

billy

out Ingka · Retail · Methods & Strategies · Products · Working at Ingka · Support & S

il · Ingka Centres · Ingka Investments

Customer DE
Retail > ... > Customer Relations
[redacted] billy

Customer Support Transformation
Retail > ... > SE Contact Centre
[redacted] Billy [redacted]
[redacted]
[redacted] BILLY ...

Sales Steering Tool Launch 4
Methods & Strategies > ... > Selling
[redacted] [redacted] BILLY [redacted]
BILLY (Billy [redacted]
[redacted] (BILLY ...

Home
Retail > ... > Marketing & Communication
[redacted] Billy - [redacted]
[redacted] Billy [redacted]

CSC Vardagsrum
Retail > ... > CSC Sortiment Möbler

Figure 2.2: Example of result page in IntraNet for query "Billy".

2.2 Method

One of the goals of this thesis is to build a reranking model using supervised machine learning. To do this, we introduce these essential steps:

1. Partitioning,
2. Labeling,
3. Feature extraction,
4. Training the model.

We will explain each step in detail in the following sections.

2.2.1 Partitioning

As proposed by Yoganarasimhan (2020), the datasets are partitioned into the following sub-datasets:

1. **Past** data from earlier sessions by the current user and other users,
2. **Present** data where we are introducing reranking.

In this thesis, we used the past data in order to make predictions on the present data. This mimics using search history (past) to make predictions on a new search (present). In other words, this is called *across-session personalization*. It is also possible to use present data, i.e. data from earlier searches within the same session, to make predictions. This is called *within-session personalization*. We only consider the across-session personalization which is properly motivated in Section 4.1, since sessions, in general, contain a low number of queries. Thus, the past data is the only data used to make predictions.

A supervised machine learning model is a model which learns from a set (usually called the training set or training dataset) of input-label pairs. The label indicates which group the input belongs to. After training, where the model learns from the training set, the model can hopefully predict the label of a new input that it has not trained on (Russell and Norvig, 2016).

To build a supervised machine learning model, the present data was further partitioned into **train**, **validation** and **test**. *Train* is prelabeled data upon which the model parameters are inferred. We introduce the notion of **overfitting**, a condition when the model fails to generalize test data and instead memorizes the noise of training data. To avoid overfitting, the model performance in each step of training is evaluated on the *validation* data. A poor result on the validation data may indicate overfitting and we cannot expect the model to perform well on new data. The model is trained on the training dataset, but the model chosen is the one with the best performance on the validation data. As both the train- and validation datasets are used in model selection, the model is finally evaluated on a separate dataset called the *test* data (Russell and Norvig, 2016).

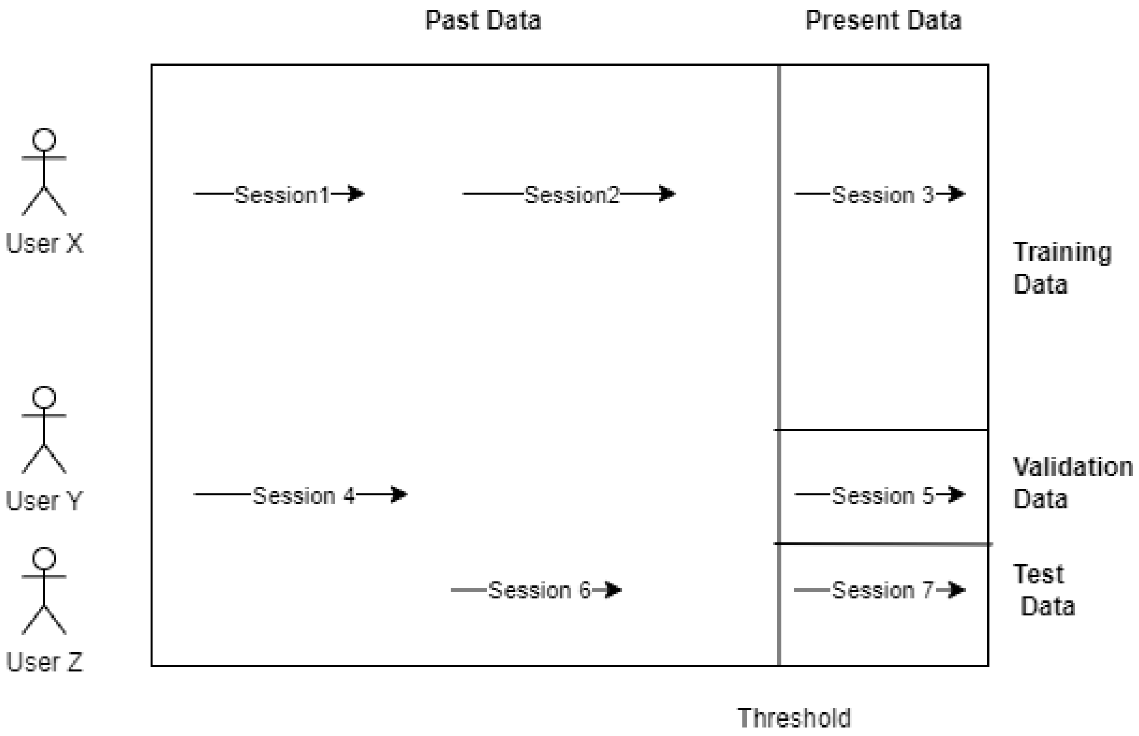


Figure 2.3: Example of data split.

In Figure 2.3, an example of the partitioning is provided. Firstly, *past* data is extracted (sessions 1, 2, 4, 6). When training the model, session 3 is used. Predictions are only made using *past* data i.e. sessions 1, 2, 4 and 6. In other words, these are the only sessions used to *extract features*. Features are further discussed in Section 2.2.3. Sessions, 5 and 7 cannot be used to make predictions.

When validating the model, session 5 is used for evaluation, and sessions 1, 2, 4, and 6 can still be used to extract features. Sessions 3 and 7 cannot be used to extract features. Similarly, session 7 is used for testing the model, and sessions 3 and 5 cannot be used to extract features. In general, many more users and sessions are present in each data set.

Similarly to Yoganarasimhan (2020), each session happening before day 25 is considered *past* data for the `red_yandex_dataset`. All users with sessions occurring on days 25 to 27 are split into train-data (60%), validation data (20%) and test data (20%). The partitioning is done with respect to users.

For the `entrypoint_dataset` and the `intranet_dataset`, the partitioning between past and present data is 85% and 15% respectively, a similar division to that of the `red_yandex_dataset`. Had more time been given, finding the optimal threshold should be explored. The partitioning for train, validation and test in the present data is 60%, 20% and 20% respectively. As with `red_yandex_dataset`, the partitioning is done with respect to users.

2.2.2 Labeling – Relevancy of a Document

Since the reranker is a supervised machine learning problem, the model is created from **labeled** training data consisting of a set of training examples. During training, the model learns to *rank* documents. As we want to find documents containing relevant information, the **relevancy** is an appropriate label for the document. We found that there are two approaches to do this:

Click-based: If a document is clicked it is considered relevant,

Dwell-time-based: An extension of the click-based approach. In addition to clicks, how much time the user spends on a document is considered. (This method is further explained in Section 5.2.1).

As mentioned in 2.1.1, we wish to compare the performance of our Yandex models to those in the competition. The one we are comparing to, infers a dwell-time-based approach, while we use a **click-based** approach due to its simplicity. Since we compare the *result* (prediction accuracy) of the two models and not the models themselves, the approaches can be different.

For the **click-based** approach, assume a document $\boldsymbol{d}$ has relevancy $\boldsymbol{r}$. Then:

$\boldsymbol{r = 0}$: The document is *irrelevant* if it has not been clicked,

$\boldsymbol{r = 1}$: The document is *relevant* if it has been clicked.

2.2.3 Features

Now that we have defined relevancy a question arises: what makes a person click a document in the first place? Is it because the user has clicked it in the past? Is it related to the user's field of work? Or simply that it happens to be high on the SERP? To deduce what makes a document relevant, information about that document, query, and user is extracted to create **features**. A feature x_{ij} is, in essence, an individual measurable piece of data. While usually *numeric*, they can be *categorical*. A *categorical feature* is a feature $x \in \mathcal{S}$ where $\mathcal{S}$ is a set of values which does not need to be ordered e.g. blood type where $x \in \{\text{A, B, AB, 0}\}$. Examples of categorical features in this report are company, country, weekday, etc. The features extracted from a single document, come in the form of a *feature vector* $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{iM})$ where $\boldsymbol{M}$ is the number of features. In order to find the most important features, defined in Section 2.2.6, we want $\boldsymbol{M}$ to initially be as large as possible and then we will prune the model if some features prove non-important. To prune the model is an important step, as we want to avoid *overfitting* which is discussed in Section 2.2.

By reconnecting to Fig. 1.1, we can extend our understanding of the reranker. In Fig. 2.4, we illustrate intermediate steps in the reranker such as feature extraction. The figure illustrates how the $\boldsymbol{k}$th document $\boldsymbol{d_i^k} \in \boldsymbol{D_i}$, is mapped to a feature vector $\mathbf{x}_i^k = (x_{i1}^k, x_{i2}^k, \ldots, x_{iM}^k)$.

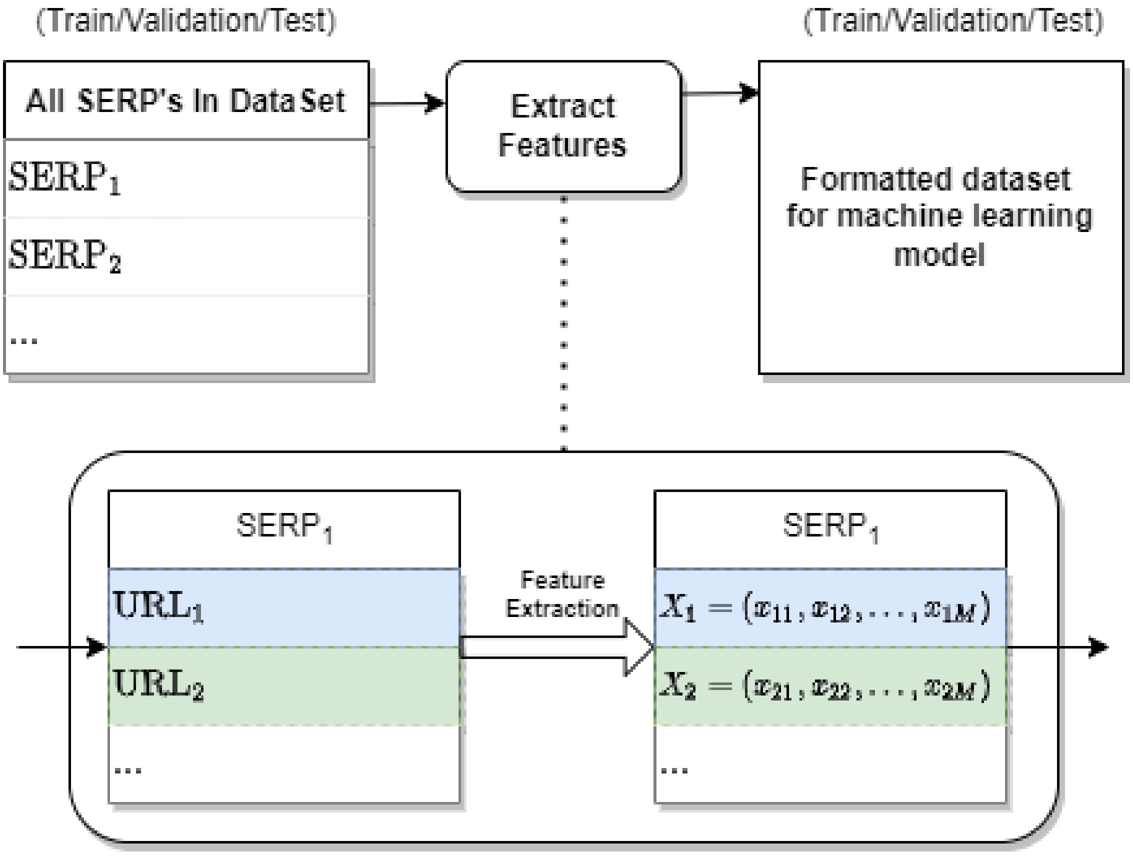


Figure 2.4: Shows how documents have to be converted into features, to format of the machine learning model.

2.2.4 Feature Selection

In this section, we will explain which features were used as input to our model. Most features are derived from a function we call $\mathit{EventListings}(\theta_1, \theta_2, \theta_3)$. Conceptually, this function produces a feature corresponding to the count of an event (θ_1) given a specific subset in the dataset (θ_2) for a document or a superset of a document (θ_3). An event is for example a click or an appearance. This function was inspired by Yoganarasimhan (2020).

Tracked Events

Given a search session and a result page in *past data*, we collect information about certain **events** during that session. Let θ_1 be such an event. The possible values for the θ_1 argument are:

Shows: how many times the document has appeared in a result list (SERP),

Weighted shows: collecting the total sum $\frac{1}{index}$, where $index + 1$ is the position in the result page where the document appeared,

Skips: how many times the document has been skipped i.e. has a document beneath it been clicked but not this one,

Clicks: how many times the document has been clicked,

Unique clicks: how many times the document has been uniquely clicked in a result page i.e. a document can only be clicked once per result page,

Weighted clicks: adds 1 if it is the first document clicked chronologically on a result page, $\frac{1}{2}$, if is the second document, $\frac{1}{3}$, if it is the third, etc.

Now assume that document d_1 has been clicked 10 times and document d_2 200 times. Then it would seem that document d_2 is more relevant than document d_1. However, assume that d_1 has appeared (number of shows) 10 times and d_2 1000 times. Then it would seem that document d_1 is more relevant than d_2 since it has been clicked every time it appeared. It would therefore be relevant to also consider the count of the tracked event divided by the *number of appearances* (Shows). We thus let θ_1 take the value of the following ratios in addition:

Weighted Shows Rate : $\frac{\text{Weighted shows}}{\text{Shows}}$

Skip Rate : $\frac{\text{Skips}}{\text{Shows}}$

Click Rate : $\frac{\text{Clicks}}{\text{Shows}}$

Unique Click Rate : $\frac{\text{Unique clicks}}{\text{Shows}}$

Weighted Click Rate : $\frac{\text{Weighted clicks}}{\text{Shows}}$

Including the events above, the dimension of θ_1 i.e. the total number of events are 11.

Subgroups

The events mentioned above can be counted in the entire *past data*. However, we assume that users related to each other can have similar preferences. Therefore, it is also relevant to consider events in **subsets** of the dataset. For example, it is interesting to see how coworkers related by country, business unit, etc. have acted. Let θ_2 denote the different ways users can be related.

The different subgroups are:

User related subgroups:

UserID defined as in Section 2.1.3,

Country: defined as in Section 2.1.3,

Business unit: defined as in Section 2.1.3,

Company: defined as in Section 2.1.3,

Job-title: defined as in Section 2.1.3,

Site: defined as in Section 2.1.3,

Session related subgroups:

Day: which weekday it is, e.g. Monday, Tuesday etc,

Query related subgroups:

Query: given this specific query,

Terms: one subgroup for each of the terms (term1 - term4) in a query. For queries with more than four terms, we disregard any term past four,

Others:

UserID and Query combination: when this user has issued this query previously,

Global: considering the entire past dataset.

Counting the different subgroups: user related (6), session related (1), query related (5), others (2) we have that $\dim(\theta_2) = 14$. We remind the reader that this is only the case for the `entrypoint_dataset` and `intranet_dataset`. The `yandex_dataset` does *not* contain information about the user, except the user ID. For Yandex we hence have the different subgroups: user related (1), session related (1), query related (5), others (2) we have that $\dim(\theta_2) = 9$

Superset of documents and Index related features.

The documents on a SERP are the results which can be can be interacted with and tracked. While information about the documents is relevant, we assume that it is also relevant to study **supersets** for documents, in our case the domains of a document similarly to Yoganarasimhan (2020), but also the *category* as introduced in Section 2.1.3. These supersets are important. If a document has no prior history, we might be able to make predictions based on the superset the document belongs to. By introducing supersets, related documents will influence the prediction of another, possibly unseen, document.

Click Bias In this thesis, the model is evaluated on the *Test-dataset*. The model is evaluated under the assumption that the user issuing the query knows which information he/she is looking for, i.e. that the same document is to be clicked no matter the ranking. This is not necessarily true, since the order in which the documents are displayed probably influences the decisions of the user. For example, users have a bias and tend to click on documents higher in the SERP regardless of relevancy! It is therefore reasonable for the model to try and capture this behavior by introducing *index* related features. These features serve the purpose of predicting which document is to be clicked, only considering where the document appears on the result list. However, since users are more likely to click on documents higher in the list (and by including index-related features) it is more likely that reranked list will be more similar to the original order. In other words, by introducing index-related features we somewhat trust the original order provided by the search engine and give less weight to our reranker. As such, we let

$$\theta_3 = \{\text{category, domain, document, index}\}.$$

The relation between these supersets is illustrated in Fig. 2.5.

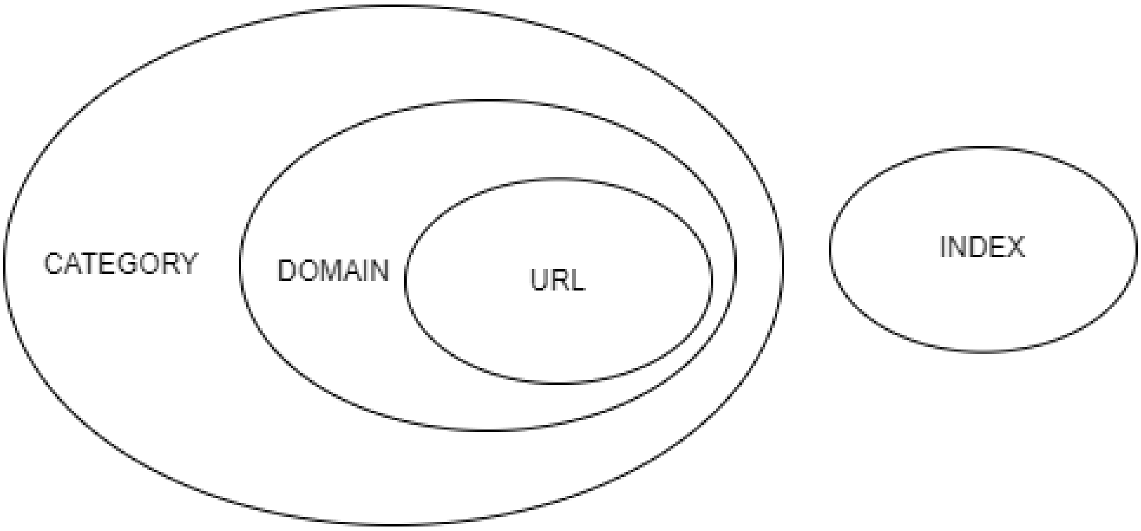


Figure 2.5: Example of supersets. This is true for every category except for *promoted* which rather is an indication if it is high on the list.

Summary. Now that θ_1, θ_2 and θ_3 have been defined, the number of features can be calculated. As we are considering all combinations of the parameters θ_1, θ_2 and θ_3, the total number of features the can be extracted using *EventListing*(θ_1, θ_2 and θ_3) is $\dim(\theta_1) \times \dim(\theta_2) \times \dim(\theta_3)$, which is:

$$11 \times 14 \times 4 = 616$$

For IKEA and

$$11 \times 8 \times 3 = 264$$

for Yandex. A few examples of function calls are given below.

EventListing(**clicks, global, document** = d) calculates the number of times d has been clicked in total in the global past dataset.

EventListing(**shows, user ID** = i, **document** = d) calculates the number of times document d has appeared in the result page of the user with userID=i.

EventListing(**clicks, country** = c, **domain** = o) calculates the number of times users in country c has clicked on documents with domain i.

EventListing(**skips, weekday** = w, **category** = c) calculates the number of times a document in category c has been skipped on weekday w.

EventListing(**Click Rate, query** = q, **index** = i) calculates the number of times index i was clicked divided by the number of result pages of at least length i, in response to query q.

Other features

SERP-rank: To further capture user behavior, and the likelihood of a document being clicked just by its location in the result page, we introduce another index-related feature *SERP-rank* Yoganarasimhan (2020).

Categorical features: For *LightGBM*, it is possible to include *categorical features* as defined in Section 2.2.3. These features can *directly* be included into the model. The drawback is that categorical features *cannot* be compared by a numerical value (lower or higher) and is instead compared in the sense that if two values are *equal* or not. If the dimension, i.e. number of possible values, is too large for a certain feature, two values are unlikely to be equal. Therefore, the dimension of the feature has to be reasonably small to avoid overfitting. The dimension depends on the size of the dataset. The categorical features included are country, site, company, job title, business unit and weekday and are included to predict relevancy given *related* users. It is worth noting that this is only valid for *LightGBM* with the IKEA datasets.

Removing redundant features directly. We have that EventListings($\frac{weighted\ shows}{shows}, \theta_2, index$) will directly produce SERP-rank no matter the value of θ_2 and all of these features are therefore removed directly. This is because the *weight* of a certain index always is the same, since it always has the same position. For example

$$\begin{aligned}\text{EventListings}(\frac{weighted\ shows}{shows}, \theta_2, index = 2) &= \frac{\frac{1}{1+2} + \frac{1}{1+2} + \ldots + \frac{1}{1+2}}{1 + 1 + \ldots + 1} \\ &= \frac{L \cdot \frac{1}{1+2}}{L \cdot 1} \\ &= \frac{1}{1+2} = \text{SERP-rank}(index = 2)\end{aligned}$$

In addition, we remove EventListings($weighted\ shows, \theta_2, index$) since

$$\begin{aligned}\text{EventListings}(weighted\ shows, \theta_2, index = 2) &= \frac{1}{1+2} + \frac{1}{1+2} + \ldots + \frac{1}{1+2} \\ &= L \cdot \frac{1}{1+2} \\ &= L \cdot \text{SERP-rank}(index = 2)\end{aligned}$$

which is SERP-rank times a constant. Thus, the maximal number of features for our IKEA models is:

$$616 + \dim(\text{SERP-rank}) - 2 \cdot \dim(\theta_2) = 616 + 1 - 2 \cdot 14 = 589$$

when considering XGBoost, plus 7 categorical when considering LightGBM. For Yandex this is

$$264 + \dim(\text{SERP-rank}) - 2 \cdot \dim(\theta_2) = 264 + 1 - 2 \cdot 9 = 249$$

In comparison, a top kaggle competitor included 293 features (Yoganarasimhan, 2020).

2.2.5 Model Outline

We wish to create a model that predicts the relevancy of a document given a feature vector. In essence, this can be reduced to finding a *scoring function* $f : \mathbb{R}^M \to \mathbb{R}$ that maps the feature vector $\mathbf{x}_i \in \mathbb{R}^M$ to a score s_i, indicating the *relevancy* of the document. The goal is to find the optimal scoring function f_{opt}, i.e. the optimal *model weights*, ω_i^{opt}, that *minimize* the *cost function* $C(\omega^1, \omega^2, \ldots, \omega^N; \mathbf{X}, \mathbf{y})$, where $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ...\}$ and $\mathbf{y} = \{y_1, y_2, ...\}$ are all the feature vectors and all the relevancy labels respectively. Specifically, $\mathbf{X} = \mathbf{X}_{\text{train}}$ and $\mathbf{y} = \mathbf{y}_{\text{train}}$. The outline of the scoring function and how it is derived is presented in Section 3.

2.2.6 Model Selection

Now that we have defined the model outline, we need a *machine learning algorithm* to train the model and a *machine learning framework* that implements it. The algorithm, **LambdaMART**, is explained in detail in Section 3.

During our research, and after consulting our supervisor, we concluded that the state-of-the-art LambdaMART implementations (Ye, 2020) to be compared are; **XGBoost** (Chen and Guestrin, 2016) and **LightGBM** (Ke et al., 2017). A multi-layer perceptron regressor using sklearn's **neural network** package was further implemented for comparison. The resulting models will be used to determine which family of algorithms (LambdaMART or Neural Network) is best to implement the reranker for IKEA. In the case of LambdaMART, we will compare the two implementations. The reasoning for comparing two different implementations of LambdaMART are different, in particular the *regression trees*. Regression trees are explained in detail in Section 3.5.1.

XGBoost

XGBoost is an open-source library and well documented. It is an open-source library that implements machine learning algorithms under the gradient boosting framework. In particular, it is an optimized distributed gradient boosting library that is highly efficient, flexible, and portable. It was released in March 27, 2014 (XGBoost, 2021).

LightGBM

LightGBM is another gradient boosting framework. It shares a lot of *XGBoost*'s advantages and is similarly designed to be distributed, efficient, accurate, and scalable (LightGBM, 2021). One important advantage is that supports *categorical features*.

Neural Network

XGBoost and LightGBM implement the LambdaMART algorithm, but we train *neural network* in a different way and is included to compare the results of the models derived from the LambdaMART implementations. The neural network train on *single* documents, trying to predict the label of that particular document. We hypothesize that we should use the LambdaMART algorithm to train our models, but if these models are proven to be worse i.e. less computationally efficient and less accurate, then there is no need to use LambdaMART and we should consider the neural network instead.

Feature Importance

For the models trained by XGBoost and LightGBM, it was possible to extract the **feature importance**. This is a measurement of how *important* the model deems certain features. Specifically, it measures **gain**: the improvement in accuracy brought by a feature. To fully understand gain as a metric, one should first understand *regression trees* introduced in Section 3.5.1. The Gain implies the relative contribution of the corresponding feature to the model calculated by taking each feature's contribution for each tree in the model. There are three main factors contributing to the feature's importance:

Correlation: how a specific feature correlates with the relevancy of the document. High correlation means high feature importance,

Feature correlation: how much correlation there is between features, which means that highly correlated features contain the same information and only one of them will have large features importance,

Existence: since most of the features are countable, if there are few instances where this feature has been incremented it is too specific and hence not reliable.

Model Pruning

When developing our models, as much information as possible was extracted and included in the model. Depending on the dataset, this may result in overfitting since there is not enough data to support the number of features. A certain feature combination may correspond to a single sample in the training set which may lead to poor model performance on new data. As such, we would like to evaluate a pruned model where many less important features have been removed.

By introducing *feature importance* as a method of scaling the model, non-important features can be removed while preserving most of the model accuracy. This method was implemented due to its simplicity. There exist more sophisticated algorithms, such as "χ^2-pruning" (Russell and Norvig, 2016) or the "Backwards-Elimination Wrapper" approach (Yoganarasimhan, 2020), but due to lack of time we very not able to implement such an algorithm. Using the feature importance, we include the N_{pruned} features, in order, corresponding to 95% of *gain*. Had more time been given, other thresholds e.g. 90% would have been explored.

If the dataset is not big enough, pruning the model in this way might improve performance due to overfitting. If the dataset is sufficiently large, removing features may decrease model accuracy even if the feature importance is low. However, the lowered training time and improvement in processing speed might support the loss in accuracy, especially when considering customers' lack of patience when waiting for search results.

2.3 Evaluation Setup

We are evaluating our models on three different sets of result pages:

Original: The original result pages without any reranking,

Prediction: The result pages when reranked using the machine learning model,

Optimal: The result pages that have been optimally reranked such that the most relevant document is highest in the list, the second most relevant document is second to highest, etc.

Optimality

In this thesis, we have a limited way of labeling documents (0 and 1). Therefore, it is important to note that there are often multiple ways to create the optimal list. Consider the example in Fig. 2.6, where there are two ways to create the optimal list. Thus, an important distinction to make is that the optimal list is decided from *label ordering* not *document ordering*. In other words, there can be multiple optimal ordering of lists. As long as no document is ordered above a document of higher relevance, the list is optimal. Hence, the internal ordering of documents with the same label is not important.

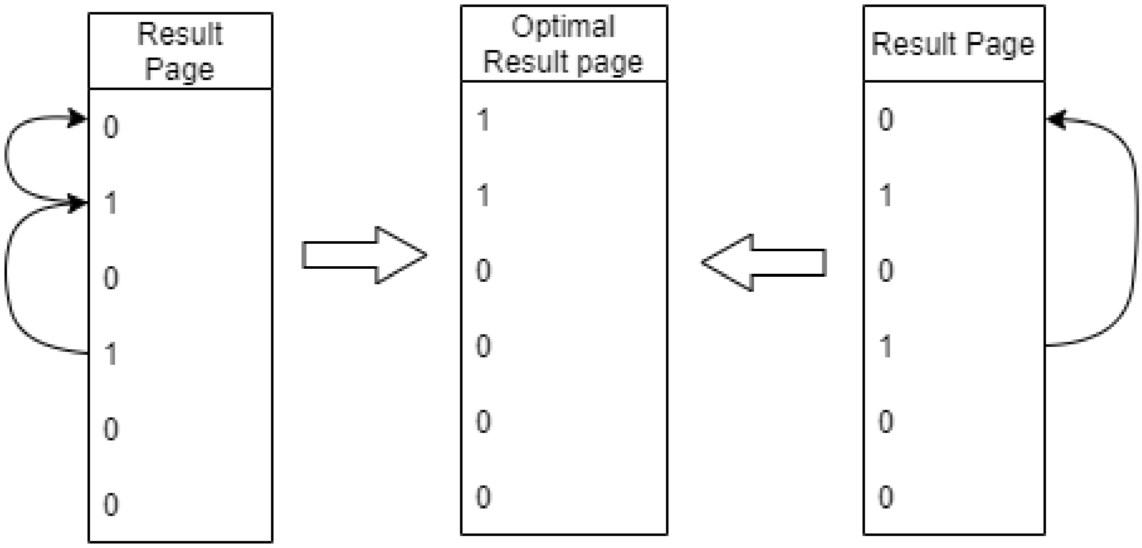


Figure 2.6: An example showing two different ways to create an optimal list

Relativity of Predicted Values

We are interested in the reranking of lists, more so than the actual prediction of whether a document is relevant or not. Hence, the predicted values of the model are important in a *relative* sense looking at the result page, not the value by itself. An example illustrating this is shown below in Fig. 2.7. In these two examples, the scores are different but the result is the same. It is, therefore, appropriate to evaluate on the result page, not single documents.

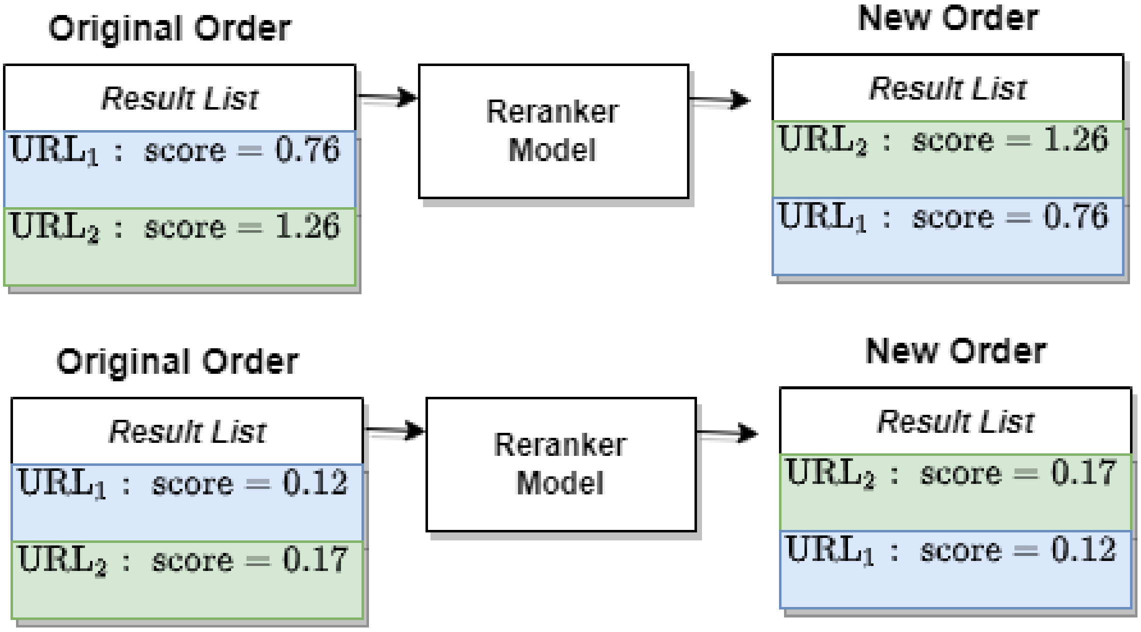


Figure 2.7: An example how reranking is relative to the result page. The two rerankings are equally good, but the scores produced are different.

2.3.1 Evaluation Metrics

NDCG and Other Metrics

To evaluate our models, we first introduce the notion of **discounted cumulative gain** (DCG) (Burges, 2010), which is an evaluation metric for lists of documents, in this case, result pages. Given a list of K documents, where the ith document d_i has relevance r_i, DCG is defined as:

$$DCG@K = \sum_{i=1}^{K} \frac{2^{r_i} - 1}{\log(1+i)}.$$

A large DCG indicates a well-ranked list, since documents with high relevance is placed at the top of the list. The main advantage with DCG is that an error towards the bottom of the list is less penalized than an error towards the top of the list due to the denominator i.e. we give more importance to documents towards the top of the list, which is advantageous (Burges, 2010). Another advantage is that DCG handles non-binary relevance levels i.e. $r_i \in \{0, 1, 2, ..., n\}$.

DCG is however not relative to the list. Two optimally ranked lists may get different DCG scores. Fig. 2.8 visualizes this. To correct this, we introduce the normalized version as

$$NDCG@K = \frac{DCG@K}{maxDCG@K},$$

where $maxDCG@K$ is the $DCG@K$ for the optimally ranked list.

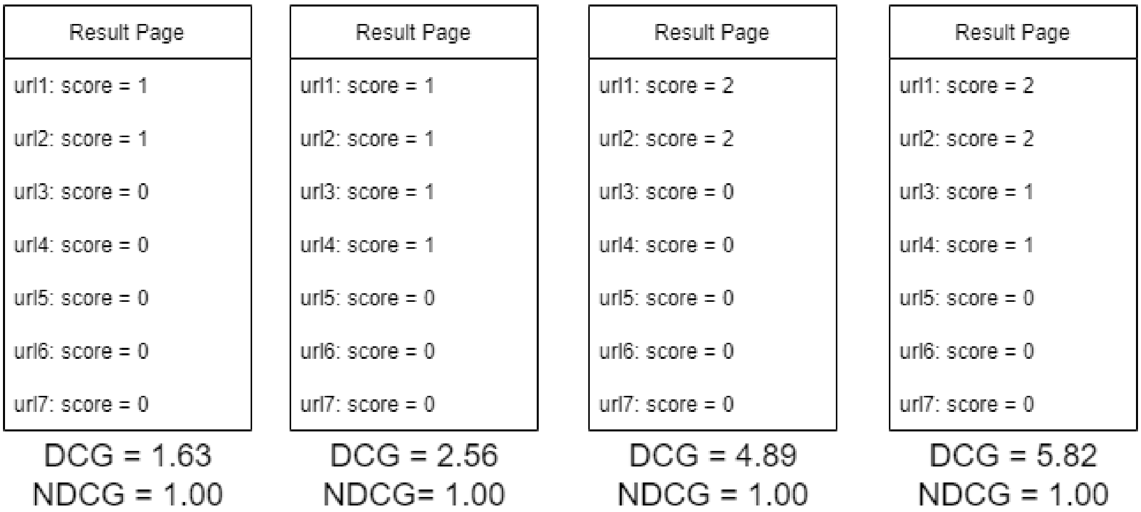


Figure 2.8: A few examples displaying the difference between DCG and NDCG of optimally ranked lists. All lists in the figure are optimally ranked.

It is reasonable to use NDCG as our main evaluation metric, since this will be the metric used when traing our model. This is further explained in Chapter 3.

In addition to NDCG, we also introduce *Average Error in the Rank of a Click* (AERC), *Mean Reciprocal Rank* (MRR) and *Change in Click-Through Rate by Position* (Δ CTR) as proposed by Yoganarasimhan (2020). These metrics are also important since they serve as neutral metrics which no model uses during training. This is particularly important when comparing to the neural network, which does not explicitly train on NDCG.

Mean Reciprocal Rank

Mean reciprocal rank (MRR) measures at what index the first click appears on average. For example, if the first click on average appears at index 0 the MRR would be $\frac{1}{0+1} = 1$, at index 1 the MRR would be $\frac{1}{1+1} = 0.5$, at index 2 the MRR would be $\frac{1}{2+1} = 0.33$ etc. Averaging over all lists gives a metric of how far up the list the user clicks. This metric was suggested by our supervisor, and is included since we consider it simple and intuitive.

Average Error in the Rank of a Click

In a well-ranked list, clicks will occur towards the top of the list. In a poorly ranked list, clicks will instead happen towards the bottom (if they happen at all). Thus, measurements that captures the position of clicks are often considered when evaluating "recommendation-systems" (Ricci et al., 2011). Presenting URLs is such a system since we recommend results in response to a query. In particular, we would like a metric that captures the difference between the optimal list and a reranked list. In other words, a metric that captures the error in the rank of clicked documents.

AERC is such a metric. First, we introduce ERC for a document d_i^k in a list D_i. This metric calculates the difference in position for a document d_i^k in D_i and a completely optimized list D_i^*. Mathematically:

$$ERC(d_i^k, D_i) = \begin{cases} \left|p(d_i^k, D_i) - p(d_i^k, D_i^*)\right|, & \text{if } d_i^k \text{ has a relevancy label different from 0} \\ 0, & \text{otherwise,} \end{cases}$$

where $p(d_i^k, D_i)$ is the position of document d_i^k in list D_i. $AERC$ is then the average ERC over the entire list of documents;

$$AERC(d_i^k, D_i) = \frac{\sum_{d_i^k \in D_i} ERC(d_i^k, D_i)}{\sum_{d_i^k \in D_i} \mathbb{1}(r(d_i^k) \neq 0)},$$

where $\mathbb{1}(r(d_i^k) \neq 0) = 1$ if and only if $r(d_i^k) \neq 0$ i.e. the relevancy of d_i^k is different from 0.

Considering Fig. 2.6, we notice that a document has to be moved three spaces to create the optimal list.

Change in Click-Through Rate by Position

Click-through rate (CTR) simply measures how often users click at certain positions in a list. The metric is often important for firms (Yoganarasimhan, 2020) since it measures how often users click at top positions i.e. how often relevant information appears towards the top of the list. A model that improves CTR towards the top of the list, at the expense of CTR towards the bottom of the list is a good algorithm for our problem. Thus, we would like to measure the change in CTR. Mathematically we introduce $\Delta CTR@p$ as the change in CTR at position p between a list D and a re-ranked list D^r. The metric can also be extended to only consider high definition clicks (relevancy $r > 1$) as $\Delta HDCTR@p$.

Note that for the entire list the CTR is always unchanged i.e. $\sum_{p=1}^{K} \Delta CTR@p = 0$, which is why, we need to consider $\Delta CTR@p$ for each p separately.

2.4 Implementation

In this section we discuss our implementation, what is supplied directly from IKEA and what is supplied by a third party. For the IKEA, all of the implementation are done by us except the data sampling. For Yandex, the implementation is done by us except data sampling and the initial parsing.

2.4.1 Dictionaries

As mentioned in Section 2.2.3, we are collecting a lot of information to make the model as good as possible. We, therefore, need an efficient data structure to support writing and reading, such that data can be efficiently stored, retrieved, and updated. *Dictionaries* in python are hash-tables and serve this purpose.

2.4.2 Implementation steps

Step 1: Parsing

As data is given in a somewhat different format for the `red_yandex_dataset` and the two IKEA datasets, the parsing is different but the resulting structure after parsing is the same. To collect data efficiently, we use *layered* `Python dictionaries`. The structure of collected data after parsing is illustrated in Fig. 2.9. As we are iterating through each row of the `DataFrame` and doing a fixed number of iterations, the parsing is done in linear time complexity.

For Yandex, a pre-written parser from another *Kaggle-competitor* on GitHub was used (Kaggle, 2014), such that we get a user-dictionary, mapping each userID to the corresponding sessions.

For IKEA, the initial parsing is done using `pandas DataFrame` with the function `read_csv()`. `DataFrame`s is convenient since it accesses a certain part of data and removes corrupted data. Since there is a column stating the environment in which the data was collected, we can remove data not relevant to the model. There are three different environments: production (EntryPoint), IntraNet, and testing. Since we are creating separate models for EntryPoint and IntraNet the relevant data can easily be extracted using `DataFrame.loc`. When the non-relevant information has been removed, it is converted into a user-dictionary similar to that of Yandex using our own implementation.

Step 2: Split

For Yandex we use the provided information about which day a session occurred to partition data into *past* (1-24) and *present* (25-27). For the *present* dataset, the unique userID's where split into *train* (60%), *validation* (20%) and *test* (20%) set using `train_test_split` from `sklearn`.

For the `entrypoint_dataset` and the `intranet_dataset`, we use 85% of samples as past data and the remaining 15% as present data, as discussed in Section 2.2.1. The partitioning of present data into train, validation, and test was done the same way as for `red_yandex_dataset` and `intranet_dataset`.

Step 3: Preprocessing

To make feature collection as efficient as possible the data was *preprocessed* before the actual feature extraction. Without preprocessing, it is possible for *all* documents to be involved in the feature extraction of a single document. Thus, the time complexity for the feature extraction of a single document would be $O(D_{\text{past}})$ where D_{past} are all documents in the *past* dataset. By introducing preprocessing, the feature extraction of a single document is done in *constant* time. This improves the efficiency of the program.

During preprocessing, every document in D_{past} is iterated. The features collected are those mentioned in Section 2.2.4. The time complexity of the preprocessing is $O(k_{\text{max}} \cdot Q_{\text{past}})$, where Q is the number of queries issued in the *past* data and k_{max} is the max number of results found on a result page. Since k_{max} is constrained and in general small (< 12) the time complexity is simply $O(Q)$. The reasoning for why a linear program is necessary, is that in our initial, quadratic implementation the program would run for around 30 min for 10, 000 documents while the linear implementation now runs around 10 seconds. This also improves the scalability of the program, as requested by IKEA.

Step 4: Creating Features

During preprocessing, features are saved in dictionaries such that the can be fetched in constant time. For a single document we extract M features and the total time complexity is $O(M \cdot D_{\text{present}})$ where D_{present} is the number of documents in the *present* data. The features for every document in train, validation and test are saved in the matrices `X_train`, `X_valid` and `X_test` respectively. The labels are saved as `y_train`, `y_valid` and `y_test`.

Step 5: Creating Models

For XGBoost, the models are trained using `xgboost.core.Booster.train` with (`y_train`, `X_train`) and (`y_valid`, `X_valid`). The parameters are the following:

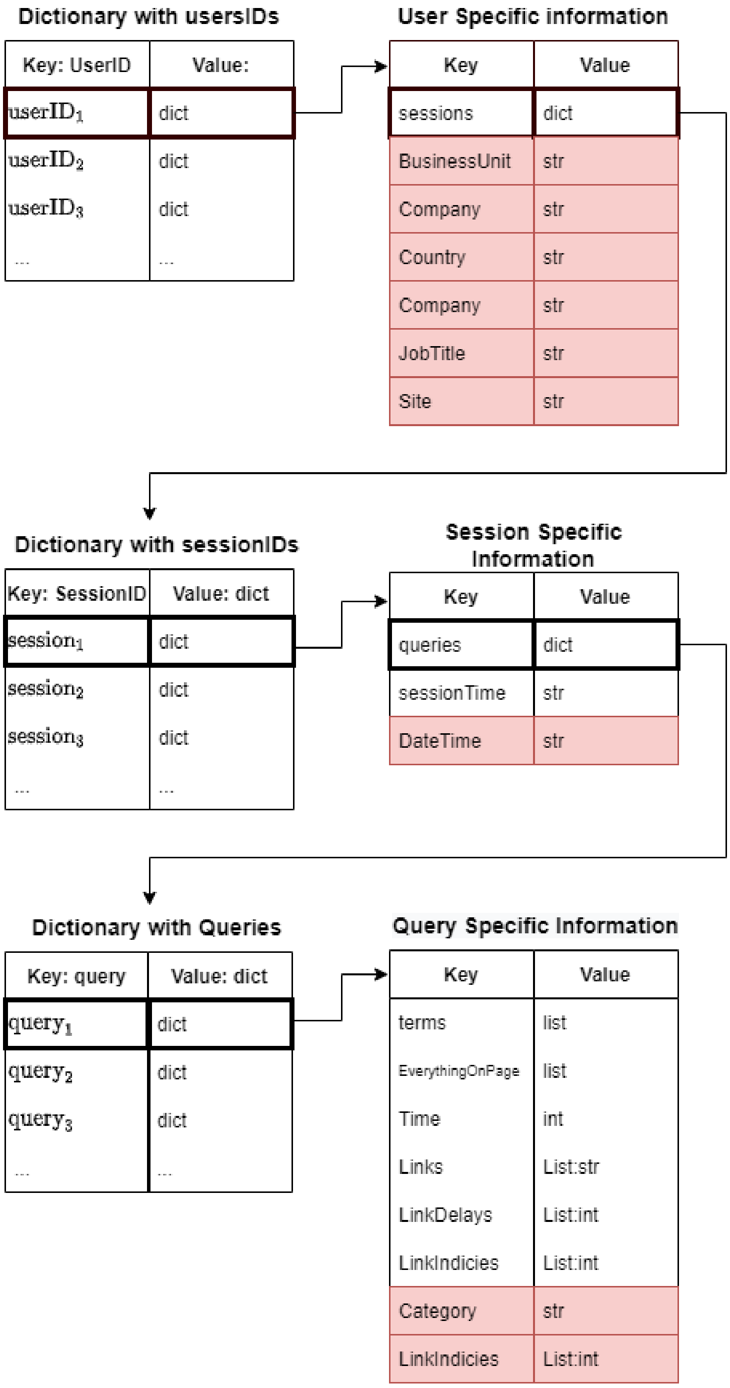
- `eta` (learning rate): 0.1,
- `objective`: '*rank:ndcg*' (uses LambdaMart introduced in Section 3)

We made a prediction using `xgboost.core.Booster.predict` and `X_test` and it was evaluated against `y_test`.

For LightGBM, the models are trained using `lightgbm.sklearn.LGBMRanker.fit` with (`y_train`, `X_train`) and (`y_valid`, `X_valid`). The parameters are the following:

- `eta` (learning rate): 0.1,
- `objective` ='*lambdarank*',
- `metric` = '*ndcg*',
- `boosting_type` ='*dart*',
- `importance_type` = '*gain*'

We made a prediction using `lightgbm.sklearn.LGBMRanker.predict` and `X_test` and it was evaluated against `y_test`.



Dictionary with usersIDs

Key: UserID	Value:
userID$_1$	dict
userID$_2$	dict
userID$_3$	dict
...	...

User Specific information

Key	Value
sessions	dict
BusinessUnit	str
Company	str
Country	str
Company	str
JobTitle	str
Site	str

Dictionary with sessionIDs

Key: SessionID	Value: dict
session$_1$	dict
session$_2$	dict
session$_3$	dict
...	...

Session Specific Information

Key	Value
queries	dict
sessionTime	str
DateTime	str

Dictionary with Queries

Key: query	Value: dict
query$_1$	dict
query$_2$	dict
query$_3$	dict
...	...

Query Specific Information

Key	Value
terms	list
EverythingOnPage	list
Time	int
Links	List:str
LinkDelays	List:int
LinkIndicies	List:int
Category	str
LinkIndicies	List:int

Figure 2.9: Diagram showing the layers of data. Red blocks means that the information is specific to IKEA.

Chapter 3

"Learning-to-Rank" Algorithms

In the previous chapter, we introduced the concept of features, labels and the goal of finding the optimal model i.e. the optimal model weights, ω_i^{opt}. As mentioned in Section 1.2, the family of algorithms specifically designed to create *ranking models* are called learning-to-rank-algorithms (LTR). In this thesis, we have used a state-of-the-art LTR called **LambdaMART**. Implementations using LambdaMART have won multiple learning-to-rank competitions such as the 'YAHOO! Learning to Rank Challenge (Track 1)' (Chapelle and Chang, 2011). To understand this algorithm, we first present its predecessors **RankNet** and **LambdaRank**. Most of the theory is based on the work of Burges (2010).

3.1 Introduction

For a reranker, we assume that we have a search engine that, given a query, will return a list of the K first documents, $D_i^K = (d_i^1, d_i^2, ..., d_i^K)$. In this thesis, choosing K as the number of documents displayed on the first page will suffice. As presented by Yoganarasimhan (2020), this is a reasonable assumption since most users do not click on a page beyond the first one. To simplify our notation, we will denote $D_i^K = D_i$.

3.2 RankNet

RankNet lays the groundwork for LambdaRANK and consequentially LambdaMART, in particular the cost function. As mentioned in Section 2.2.5, we want to find the optimal scoring function $f : \mathbb{R}^M \rightarrow \mathbb{R}$ that maps a feature vector to a score. For *RankNet*, f is an arbitrary C^1 function. Assume d_i and d_j are two documents in the same result list D. We denote $d_i \triangleright d_j$ as the case where document d_i is *more relevant* than d_j, i.e. that document d_i should be ranked higher than d_j. The probability is mapped via a logistic function as:

$$\begin{aligned} P_{ij} \quad \equiv \quad P(d_i \triangleright d_j | f, \mathbf{x}_i, \mathbf{x}_j, \sigma) \quad &= \quad \frac{1}{1 + \exp\Big(- \sigma(f(\mathbf{x}_i) - f(\mathbf{x}_j))\Big)}, \\ &= \quad \frac{1}{1 + \exp\big(- \sigma(s_i - s_j)\big)}, \end{aligned}$$

where σ is the shape parameter. We define $\overline{P}_{ij}$ as the observed probability and it is defined as follows:

$$\overline{P}_{ij} = \begin{cases} 1, & \text{if } d_i \text{ is more relevant than } d_j, \\ 0, & \text{if } d_j \text{ is more relevant than } d_i, \\ 0.5, & \text{if } d_i \text{ and } d_j \text{ are equally relevant.} \end{cases}$$

We hence can define the likelihood of our model, given d_i and d_j, as:

$$L(d_i, d_j) = P_{ij}^{\overline{P}_{ij}} \cdot (1 - P_{ij})^{1 - \overline{P}_{ij}},$$

Instead of *maximizing* the likelihood of our model given the data, we can instead *minimize* the negative log-likelihood of our model. We hence define the cross-entropy function (cost function) as:

$$C(d_i, d_j) = -\log L(d_i, d_j) = -\overline{P}_{ij} \log P_{ij} - (1 - \overline{P}_{ij}) \log(1 - P_{ij}) \tag{3.1}$$

hence we have that

$$C = \begin{cases} -\log P_{ij}, & \text{if } \overline{P}_{ij} = 1 \\ -\log(1 - P_{ij}), & \text{if } \overline{P}_{ij} = 0 \\ -\frac{1}{2}(\log P_{ij} + \log(1 - P_{ij})), & \text{if } \overline{P}_{ij} = 0.5. \end{cases}$$

The cost function penalizes deviation in the output probabilities P_{ij} from the observed probabilities $\overline{P}_{ij}$.

In a labeled training data set, $\overline{P}_{ij}$ is known. As such, we can introduce:

$$\overline{P}_{ij} \equiv \frac{1}{2}(1 + S_{ij}) \tag{3.2}$$

where

$$S_{ij} = \begin{cases} 1, & \text{if } d_i \text{ is labeled as more relevant than document } d_j, \\ -1, & \text{if } d_j \text{ is labeled as more relevant than } d_i, \\ 0, & \text{if the documents have the same label.} \end{cases}$$

Combining Eq. (3.1) and Eq. (3.2), we arrive at:

$$C(d_i, d_j) = C = \frac{1}{2}(1 - S_{ij})\sigma(s_i - s_j) + \log\Big(1 + \exp\big(- \sigma(s_i - s_j)\big)\Big)$$

where we notice the following symmetry.

$$S_{ij} = 1 \implies C = \log\Big(1 + \exp\big(-\sigma(s_i - s_j)\big)\Big)$$

but

$$S_{ij} = -1 \implies C = \log\Big(1 + \exp\big(-\sigma(s_j - s_i)\big)\Big).$$

Finally, the gradient of the cost function Eq. (3.1) is constructed as

$$\frac{\partial C}{\partial s_i} = \sigma\left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + \exp\left(\sigma(s_i - s_j)\right)}\right) = -\frac{\partial C}{\partial s_j} \tag{3.3}$$

The model weights $\{\omega_i\}_{i=1,\dots,N}$ are updated using *gradient descent* yielding:

$$\begin{aligned} \omega_k &\leftarrow \omega_k - \eta\frac{\partial C}{\partial \omega_k}, \\ &\leftarrow \omega_k - \eta\Big(\frac{\partial C}{\partial s_i}\frac{\partial s_i}{\partial \omega_k} + \frac{\partial C}{\partial s_j}\frac{\partial s_j}{\partial \omega_k}\Big), \end{aligned} \tag{3.4}$$

where $\eta > 0$ is the learning rate parameter. We can factorize Eq. (3.4) using Eq. (3.3) yielding:

$$\begin{aligned} \frac{\partial C}{\partial \omega_k} &= \frac{\partial C}{\partial s_i}\frac{\partial s_i}{\partial \omega_k} + \frac{\partial C}{\partial s_j}\frac{\partial s_j}{\partial \omega_k} \\ &= \sigma\left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + \exp(\sigma(s_i - s_j))}\right)\Big(\frac{\partial s_i}{\partial \omega_k} - \frac{\partial s_j}{\partial \omega_k}\Big) \\ &= \lambda_{ij}\Big(\frac{\partial s_i}{\partial \omega_k} - \frac{\partial s_j}{\partial \omega_k}\Big) \end{aligned}$$

where we in the third equality define:

$$\lambda_{ij} \equiv \sigma\left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + \exp(\sigma(s_i - s_j))}\right). \tag{3.5}$$

Here, λ_{ij} is the gradient of the cost with respect to the scores and is important for future improvements. These improvements are discussed in the upcoming sections.

3.3 LambdaRank

3.3.1 Introducing NDCG in RankNet

When considering two documents in RankNet where the relevancy ordering is wrong, they are updated equally in each step. This means that RankNet is designed to *minimize the number of inversions* between documents with different labels (relevancy) and is not designed to take user behavior into account. To improve ranking, we introduce **NDCG** when training our model.

The crucial observation to improve RankNet is that we do not need the value of the cost function, only the gradient. If we have computed the gradient, we can simply model the cost with respect to the model scores λ. *LambdaRank* includes NDCG as a metric, which makes it an improvement over RankNet. Imagine there is a utility function C where we modify λ_{ij} in Eq.3.5 by multiplying the size of the change in NDCG produced by swapping the place of the two documents d_i and d_j, denoted $\left|\Delta NDCG_{ij}\right|$:

$$\lambda_{ij} \equiv \sigma\left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + \exp(\sigma(s_i - s_j))}\right)\left|\Delta NDCG_{ij}\right|. \tag{3.6}$$

By multiplying with $\left|\Delta NDCG_{ij}\right|$ it proven empirically that this maximizes NDCG. For the proof the is encouraged to read Burges (2010, page 9).

3.4 Speeding up LambdaRank

In the previous sections, we introduced λ_{ij} which, given two documents, can be seen as forces moving the documents d_i, d_j up or down by increasing or decreasing their predicted relevance. Here, λ_{ij} is computed for all possible pairs in the list $d_i, d_j \in D$. Let I by the set of all such pairs (d_i, d_j) where $i \neq j$. Since each pair only needs to be regarded once, we constrain I to only include pairs where $d_i \rhd d_j$. In the case where d_i and d_j have the same relevancy, we have that $|\Delta NDCG_{ij}\right| = 0$ and therefore does not need to be included in I. With this restriction, for a pair $(i, j) \in I$ we have that $S_{ij} = 1$ and derive the following from Eq. (3.6):

$$\lambda_{ij} = \frac{-\sigma\left|\Delta NDCG_{ij}\right|}{1 + \exp\left(\sigma(s_i - s_j)\right)}. \tag{3.7}$$

For a given document d_i, we can accumulate the λ_{ij}'s to get how much the document should be "moved" in total:

$$\lambda_i = \sum_{j:\{i,j\}\in I} \lambda_{ij} - \sum_{j:\{j,i\}\in I} \lambda_{ij}. \tag{3.8}$$

The interpretation of Eq. (3.8) is thus the sum of Eq. (3.7) for swapping d_i with any document d_j that has a less relevant label minus the sum of Eq. (3.7) for every document d_j that has a more relevant label. As mentioned by Burges (2010), summing contributions from all pairs, then doing the update leads to significant speedup in training since weight update is expensive.

3.5 Multiple Additive Regression Trees (MART)

The gradient descent used so far can be interpreted as an arbitrary machine learning model. However, LambdaMART combines LambdaRank with *multiple additive regression trees* (MART) Burges (2010). It uses *gradient boosted regression trees* with a cost function derived from LambdaRank to order any ranking situation.

3.5.1 Regression Trees

A regression tree is a directed tree graph that processes a data-point, consisting of a feature vector $\mathbf{x}_i$ and a label y_i, until it is placed in a leaf node (a node without children) with the purpose of predicting the label y_i.

At each non-leaf node, the tree considers one feature $x_{ij} \in \mathbf{x}_i$ and compares it to a threshold t_j. If $t_j < x_{ij}$, the data-point is passed to the left child. Similarly, if $t_j \geq x_{ij}$ it is passed to the right child. The process is repeated until the child is a leaf node. The value corresponding to that leaf will be the predicted value of $\mathbf{x}_i$. What features and thresholds to use in the tree are decided during training, where every data point is eventually assigned to a leaf node. An example of a regression tree is provided in Figure 3.1. Note that trees, in general, have large depths, but are kept small in this example for the sake of clarity.

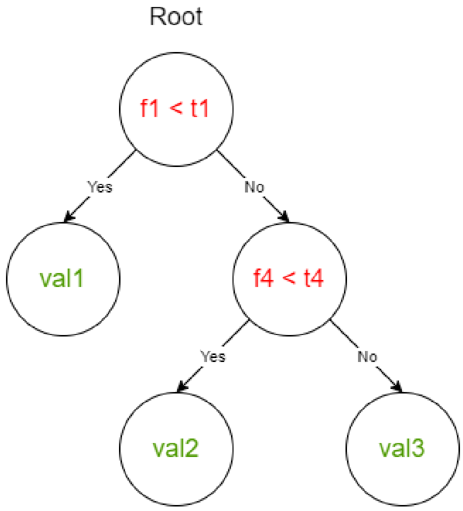


Figure 3.1: Example of a regression tree. Let $\mathbf{x}_i = (f1 = x_{i1}, f2 = x_{i2}, .., fM = x_{iM})$, be a data-point passed is passed through the tree from the root node. For example, let $x_{i1} \geq t1\ t1$ and $x_{i4} < t4$, then the predicted value would be $val2$.

The trees are built different for XGBoost and LightGBM and there reader is encouraged to read XGBoost (2021) and LightGBM (2021) for further detail. It is important that this is the second notable difference between XGBoost and LightGBM, the first being the support of categorical features. This further validate the need to investigate and compare the results models of the two frameworks.

3.5.2 MART

MART is a class of algorithms that can be trained for a general loss functions using regression trees. Similar to regular regression trees, it maps a feature vector $\mathbf{x}_i \in \mathbb{R}^M$ to a score $s_i \in \mathbb{R}$. During training, MART builds many trees, where each new tree built improves upon the previous ones. The goal is to minimize a differentiable loss-function L and with each tree, the output takes a small step towards an optimal output value for the given input. The general structure for the output can be written as

$$F_T(\mathbf{x}_i) = \sum_{n=1}^{T} \alpha_n f_n(\mathbf{x}_i). \tag{3.9}$$

Here $f_n(\mathbf{x}_i)$ and α_i corresponds to the value produced by the ith regression tree and the weight associated with it. But how does MART minimize L? First we initialize the model as

$$F_0(\mathbf{x}_i) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma) \tag{3.10}$$

to some constant value. When $F_0(x)$ has been initialized, we start the iterative process of building regression trees. This is done for $m = 1$ to T, where T is the number of trees to build. In this rapport the number of trees built chosen are 100, as default for LightGBM LightGBM (2021).

In each step, we first compute the *pseudo residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(x)=F_{m-1}(x)}, \quad \text{for } i = 1, \ldots, n. \tag{3.11}$$

Now the new tree is built using the pseudo-residuals i.e.the training set $\{(\mathbf{x}_i, r_{im})_{i=1}^{n}\}$. Worth noting is that the tree hence *models gradients of the cost-function with respect to the model scores*. In particular, the model scores is that of the "latest model" e.g. for the first iteration $F(x) = F_{m-1}(x) = F_0(x)$.

For each leaf l, in the new tree, we compute the multiplier γ_{lm} by solving:

$$\gamma_{lm} = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma). \tag{3.12}$$

The model is then updated as:

$$F_m(x) = F_{m-1}(x) + \eta \sum_{l} \gamma_{lm} I(\mathbf{x}_i \in R_{lm}) \tag{3.13}$$

where R_{lm} are all data-points attached to the lth leaf-node in the mth tree. Further $I(\mathbf{x}_i \in R_{lm})$ is an indication function defined as:

$$I(\mathbf{x}_i \in R_{lm}) = \begin{cases} 1, & \text{if } \mathbf{x}_i \text{ is assigned to leaf } R_{lm}, \\ 0, & \text{if } \mathbf{x}_i \text{ is not assigned to leaf } R_{lm}. \end{cases}$$

An example of the MART algorithm is shown in Fig. 3.2.

3.6 LambdaMART

The final piece of the puzzle is combining LambdaRank and MART to produce *LambdaMART* Burges (2010), the algorithm used by Yoganarasimhan (2020). The idea is to use MART, which

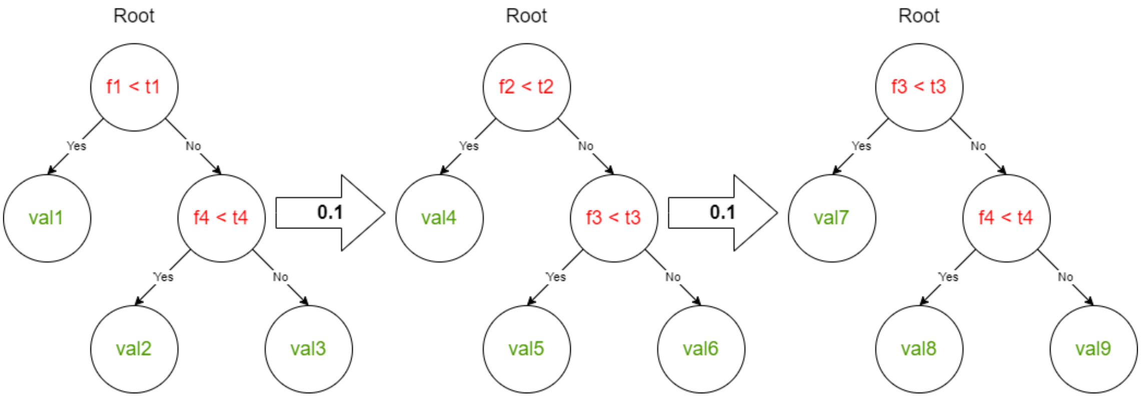


Figure 3.2: Example of three Regression Trees. In MART multiple trees are built, each trying to compensate for the mistake of the earlier ensemble to reach the optimal solution. Each contributing tree is scaled by the learning rate η (= 0.1), taking many small steps are taken towards an optimal solution.

uses boosted regression trees to model the gradient of the cost with respect to model scores. We first reconnect to Eq. (3.8). We simplify the notation and denote the operation as follows:

$$\sum_{\{i,j\} \rightleftharpoons I} \lambda_{ij} \equiv \sum_{j:\{i,j\} \in I} \lambda_{ij} - \sum_{j:\{j,i\} \in I} \lambda_{ij} \tag{3.14}$$

We can hence formulate the utility function for which λ_i is the derivative:

$$C = \sum_{\{i,j\} \rightleftharpoons I} \left|\Delta NDCG_{ij}\right| \log\left(1 + \exp\left(-\sigma(s_i - s_j)\right)\right) \tag{3.15}$$

such that

$$\frac{\partial C}{\partial s_i} = \sum_{\{i,j\} \rightleftharpoons I} \frac{-\sigma\left|\Delta NDCG_{ij}\right|}{1 + \exp\left(\sigma(s_i - s_j)\right)} = \sum_{\{i,j\} \rightleftharpoons I} -\sigma\left|\Delta NDCG_{ij}\right|\rho_{ij}, \tag{3.16}$$

where we have defined ρ_{ij} as:

$$\rho_{ij} \equiv \frac{1}{1 + \exp(\sigma(s_i - s_j))} = \frac{-\lambda_{ij}}{\sigma\left|\Delta NDCG_{ij}\right|}.$$

Now we have everything we need to build the regression trees. From RankNet we have a well defined cost (utility) function and from MART we have a way to model the gradient. Reconnecting to Eq. (3.12), we still need a way to find the γ's. This can be approximated by using the newton step as $x_{k+1} = x_k + t$ with $t = -\frac{f'(x)}{f''(x)}$.

and we hence get an expression for the second derivative by differentiating again:

$$\frac{\partial^2 C}{\partial s_i^2} = \sum_{\{i,j\} \rightleftharpoons I} \sigma^2 \left| \Delta NDCG_{ij} \right| \rho_{ij}(1 - \rho_{ij}). \tag{3.17}$$

Using Eq. (3.16) and Eq. (3.17) we finally arrive at the step size for the l:th leaf of the m:th tree as:

$$\gamma_{lm} = \frac{\sum_{x_i \in R_{lm}} \frac{\partial C}{\partial s_i}}{\sum_{x_i \in R_{lm}} \frac{\partial^2 C}{\partial s_i^2}} = \frac{\sum_{\{i,j\} \rightleftharpoons I} -\sigma \left| \Delta NDCG_{ij} \right| \rho_{ij}}{\sum_{\{i,j\} \rightleftharpoons I} \sigma^2 \left| \Delta NDCG_{ij} \right| \rho_{ij}(1 - \rho_{ij})}. \tag{3.18}$$

In conclusion, we have now gathered all parts needed for *LambdaMART*. For any model F_{k-1}, we build a new tree by calculating the pseudo-residuals r_{im} defined in Eq. 3.11, and assign new values to the leaves according to Eq. (3.18). The next iteration of the model is thus:

$$F_m(\mathbf{x}_i) = F_{m-1} + \eta \sum_l \gamma_{lm} I(x_i \in R_{lm})$$

where η once again is a user defined learning rate and $I(x_i \in R_{lm})$ is the indication function.

Chapter 4

Evaluation

In the evaluation chapter, we present and discuss our results. Firstly, we present an analysis of the datasets without using any found models to investigate how applicable personalization is. This is important, to understand how much we can expect each dataset to benefit from personalization. We proceed to discuss and present the results for the models on our chosen metrics.

4.1 Model Free Analysis

When considering search personalization, it is important to analyze whether it will improve the user experience or not. As discussed in Section 1.2, if for every query, users click the same URL, personalization will not bring any improvements since all users expect the same result. On the other hand, if users tend to click on many different URLs in response to the same query there is much to gain from reranking.

To evaluate the potential of personalization on our data we analyze the following metrics:

- How large search histories do users have i.e. how many queries have been issued in the past. This is used for 'across session personalization',
- If user behavior supports 'within-session personalization',
- If users have heterogeneous preferences i.e. do they click different URLs in response to the same search query,
- If users are tenacious. i.e. how many clicks do they perform for the same search query?

4.1.1 IKEA datasets

In Fig. 4.1 we present how many queries users have been issued in the past. Since information is collected on click, every user present in the dataset has at least one earlier search. Notably, for *EntryPoint* ~55% of users have issued two or fewer queries and ~70% for *IntraNet*, indicating that many users have a different amount of search history. This fact indicates less support for across-session personalization for especially IntraNet, but also EntryPoint, when compared to Yandex. As more users use the search engine performance might however improve.

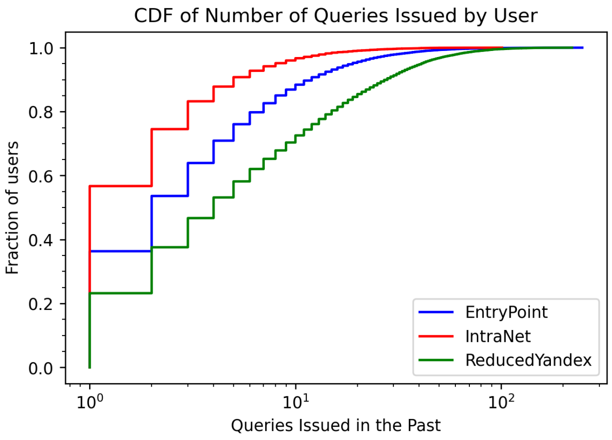


Figure 4.1: Number of queries issued by users.

Furthermore, in Fig. 4.2 we study the entropy of the datasets i.e. how many unique documents are clicked in response to a search query. For *EntryPoint* ~ 60% of queries leads to clicks on one link or less. Similarly, for *IntraNet*, the number is ~70%. These are called *navigational queries* and do not benefit from personalization since user preferences are heterogeneous. Since the datasets mainly consist of such queries, we might expect worse results if the single relevant document is reranked to a lower position.

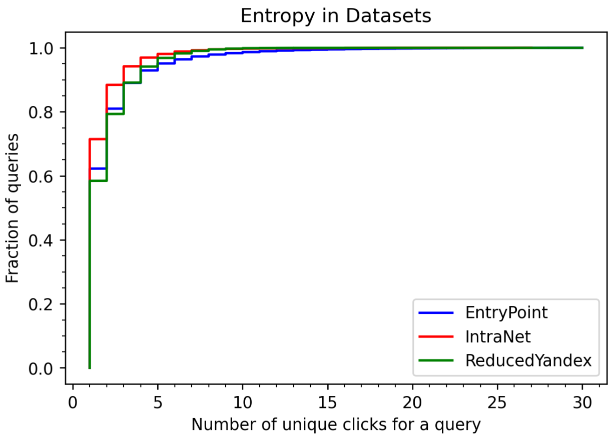


Figure 4.2: Entropy for the datasets i.e. how many unique documents are clicked given a query.

In Fig. 4.3 we study the support for within-session personalization i.e. how many queries users issue within a session. For *EntryPoint* most sessions only consist of one query. Users are either happy with their results, or they give up on the search. Only **11%** of users issue multiple queries in the same session. Due to how sessions are reported in *IntraNet*, every session consists of only one query. These findings motivate why within-session personalization, i.e. reranking depending on earlier searches in the same session, is not explored in this report.

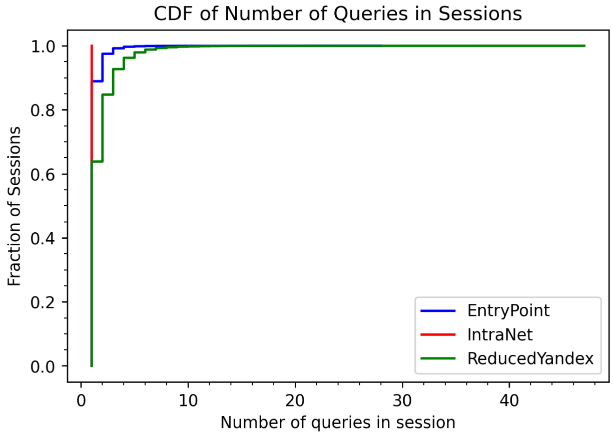


Figure 4.3: Number of search queries in sessions. Due to how data is reported, every session in IntraNet contains of only one query.

Finally in Fig. 4.4 we can see that for *EntryPoint* ~46% (~33% for *IntraNet*) of users click on multiple documents on average. This shows the tenacity among users and can be used to indicate which labeling technique should be used. The more tenacity showed among users, the more we can rely on labeling using dwell time. This is because dwell time assumes that the user takes the time to consider a document, and clicks a new one if the first one was deemed irrelevant. If users tend to just click one document, we can forego dwell time and just determine relevancy based on clicks. This was our chosen approach.

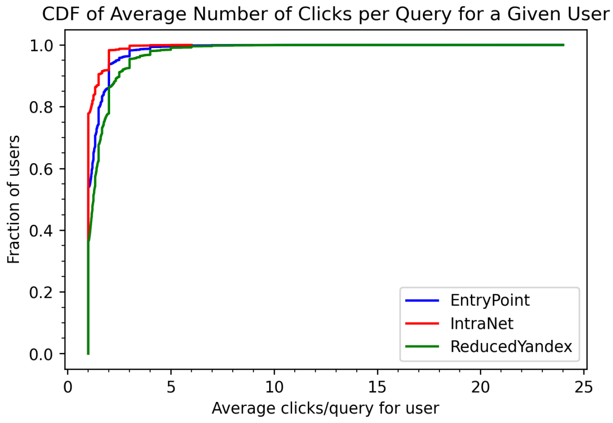


Figure 4.4: Average number of clicks a user performs in response to a search query.

4.1.2 Comparing to the Yandex Dataset

A similar evaluation for the *Yandex* dataset has already been done by others such as (Yoganarasimhan, 2020). The main findings are that the dataset supports both across-session and within-session personalization and the data has reasonably high entropy.

For completeness, we provide a discussion about our `red_yandex_dataset` still concerning Fig. 4.1, 4.2, 4.3 and 4.4.

In comparison to the two IKEA datasets, *Yandex* users have larger search histories. Users also perform more clicks per query on average. We also have more data about users in general. Due to this, we argue that we have more information about users, and as such we expect the `red_yandex_dataset` to benefit more from personalization.

Concerning the IKEA datasets, there is still similar entropy and some users have large search histories. As such we still expect a reranker to increase performance. Returning to Table 2.3, we further note that the `entrypoint_dataset` is much larger than `intranet_dataset`. It is thus reasonable to expect the *EntryPoint* model to perform better. As more data is collected, the result are likely to continuously improve for both datasets.

4.2 Results

In the following section, we will present the obtained results. The section will initially be partitioned by dataset. Finally we present results by user history for all datasets. Since we showed that the `entrypoint_dataset` is better suited for personalization than the `intranet_dataset` in Section 4.1, we disregard the `intranet_dataset` results which instead can be found in Appendix A. Feature importance for the different models are presented in Appendix B.

We remind the reader that there are five models created by three machine learning frameworks. The models are called:

XGBoost derived from the XGBoost framework,

Pruned XGBoost which is the pruned XGBoost model,

LightGBM derived from the LightGBM framework,

Pruned LightGBM which is the pruned LightGBM model,

Neural Network derived from the `sklearn` framework.

4.2.1 Dataset: EntryPoint

Table 4.1 shows the result when comparing model performance on the `entrypoint_dataset`. We allow ourselves to rerank the list *freely*, disregarding the 'Block Structure' of EntryPoint, explained in Section 2.1.3. Recall that $NDCG$ is the metric that the XGBoost- and LightGBM-models are trained to maximize, while MRR and $AERC$ are complementary metrics. The pruned models are the models where we have removed less important features according to cumulative gain. The column with the head 'original' provides a baseline since it evaluates the *original* ranking provided by the search engine. The column with the head 'optimal' instead shows the *optimal* results for the dataset discussed in Section 2.3. Lastly, we present the number of features used in each model.

Notably, all our models have merit since they improved NDCG after reranking. The full (pruned) XGBoost yields an increase of 5.0967% (5.1678%) while the full (pruned) LightGBM yields 4.7275% (4.4174%). Finally, the neural network got 4.4112%. This should be compared to the optimal increase of 15.6%.

It is important to note, that the models also have an improvement in the complementary metrics MRR, DCG, and $AERC$. This is also true for the CTR. The results of $\Delta CTR@k$, defined in Section 2.3.1, for $p = 0, ..., 9$ are shown in Table 4.2. For the EntryPoint dataset, we rarely have exactly 10 results. Sometimes the SERPs contains more results and sometimes fewer. To be able to compare lists, we had to decide on a cutoff. This was chosen as 10 since very few relevant documents are placed at position $k > 9$.

Similar to the other metrics, CTR at top positions are improved for all models at the expense of CTR at lower positions. This indicates a well-ranked list since relevant documents are placed at the top of the SERP.

	Original	Optimal	XGBoost	Pruned XGBoost	LightGBM	Pruned LightGBM	Neural Network
MRR	0.825	1.0	0.886	0.887	0.882	0.878	0.879
DCG	8405	9857	8859	8865	8826	8801	8796
NDCG	0.865	1.0	0.909	0.910	0.906	0.903	0.903
AERC	1.81	0.0	1.260	1.395	1.427	1.353	1.516
Num. features	–	–	589	206	596	26	589

Table 4.1: Model performance on EntryPoint dataset.

k	Optimal	XGBoost	Pruned XGBoost	LightGBM	Pruned LightGBM	Neural Network
0	19.365	6.428	6.602	5.785	5.433	6.178
1	-0.811	0.216	0.320	0.197	0.218	0.293
2	-1.801	0.882	0.812	1.249	1.314	1.094
3	-6.926	-3.942	-3.897	-3.985	-3.804	-4.266
4	-4.264	-2.143	-2.159	-1.94	-1.771	-2.123
5	-2.548	-1.031	-0.979	-0.984	-1.031	-0.83
6	-1.141	-0.216	-0.205	-0.09	-0.069	-0.158
7	-0.734	-0.056	-0.192	-0.086	0.013	-0.01
8	-0.556	-0.064	-0.151	-0.045	-0.083	-0.089
9	-0.584	-0.074	-0.151	-0.103	-0.22	-0.089

Table 4.2: $\Delta CTR@k$ (%) for the Entry Point dataset.

The results of the models are similar, even though the *full* models perform a bit better one should also consider the results in relation to the number of features. *Pruned LightGBM* uses the least number of features, even comparing to *Pruned XGBoost*.

4.2.2 Dataset: Reduced Yandex

We are now presenting results for the `red_yandex_dataset` to verify our choice of models.

Table 4.3 shows results for the different models. Reranking with the *XGBoost* or *LightGBM* models have improved $NDCG$ for the `red_yandex_dataset`. The Neural Network failed to converge and reduced the $NDCG$ compared to the baseline.

The full (pruned) *XGBoost* model improved $NDCG$ by 0.71% (0.95%) while the full (pruned) *LightGBM* model yielded an improvement of 1.19% (1.3%). The LambdaRANK models brought an improvement on MRR and $AERC$. Notably the pruned model got a better result, despite using fewer features.

This is also true for the CTR, which is shown in Table 4.4. While the neural network did not bring any improvement, *XGBoost* and *LightGBM* increased CTR at top positions, at the expense of lower positions. In the `red_yandex_dataset` every SERP has exactly 10 results. Thus no cutoff is needed.

	Original	Optimal	XGBoost	Pruned XGBoost	LightGBM	Pruned LightGBM	Neural Network
MRR	0.801	1.0	0.809	0.8113	0.813	0.814	0.287
DCG	3072.051	3649.851	3089.074	3094.113	3100.947	3101.694	3649.850
NDCG	0.840	1.0	0.846	0.848	0.850	0.851	0.445
AERC	1.161	0.0	1.144	1.147	1.123	1.126	5.399
Num. features	-	-	249	161	249	79	249

Table 4.3: Model performance on the reduced *Yandex* dataset.

k	Optimal	XGBoost	Pruned XGBoost	LightGBM	Pruned LightGBM	Neural Network
0	19.799	0.663	0.914	1.166	1.257	-35.025
1	2.629	0.229	0.274	0.114	-0.046	-10.837
2	-2.423	-0.526	-0.754	-0.709	-0.64	-6.104
3	-3.589	-0.206	-0.389	-0.503	-0.572	-2.743
4	-3.109	-0.046	-0.206	0.343	0.274	0.183
5	-3.292	-0.32	-0.137	-0.206	-0.091	2.012
6	-2.858	-0.023	-0.069	-0.114	-0.16	3.772
7	-2.424	0.389	0.183	0.137	0.274	6.838
8	-2.469	-0.297	0.183	-0.412	-0.457	11.637
9	-2.263	0.137	0.0	-2.263	0.16	30.27

Table 4.4: $\Delta CTR@k$ for the reduced *Yandex* dataset.

4.2.3 Result by user history

So far we have presented results for all users aggregated. Another approach is to divide users by the size of their search history and investigate if a longer search history improves personalization, or if other features prove more important. Figure 4.5 shows such a plot for the `entrypoint_dataset`. Due to the upper bound on the x-axis, we increase the number of users, including users with longer search history, as we move to the right. Figure 4.7 visualizes this. Thus we can see if the result improves when these users are added.

As we increase the upper bound, the average NDCG is initially increased for both models. However, the average NDCG increase stagnates and is eventually lowered. As such, we got a larger average NDCG increase when only considering users with smaller than average search histories, rather than the entire dataset. It should be noted that the change in increase is only ~0.8% for the XGBoost model and ~0.5% for the LightGBM model.

For the `red_yandex_dataset`, users in general issue more queries. Fig. 4.6 presents the average $NDCG$ improvement as before. The negative values on the y-axis mean that $NDCG$ is reduced. Further Fig. 4.7 shows the number of users for each level of search history. For this dataset, we instead find a larger average increase in NDCG-improvement when considering the entire dataset, rather than only considering users with smaller than average search histories.

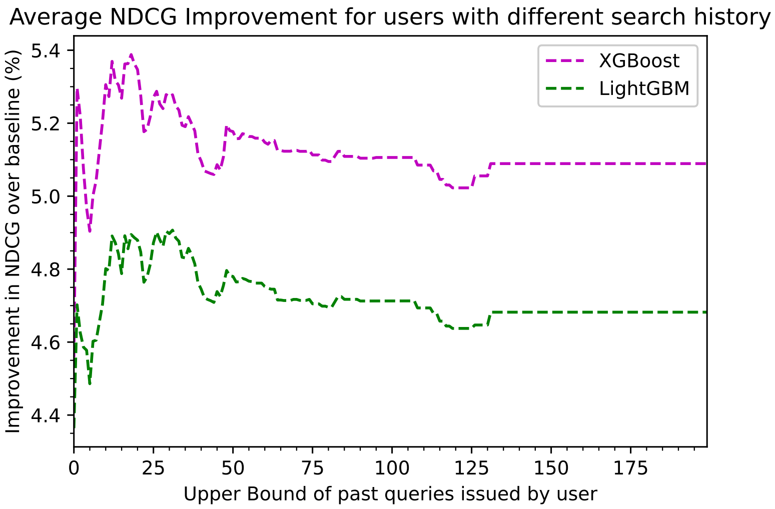


Figure 4.5: Average NDCG improvement on the EntryPoint dataset for users with search history if length up to the specified bound.

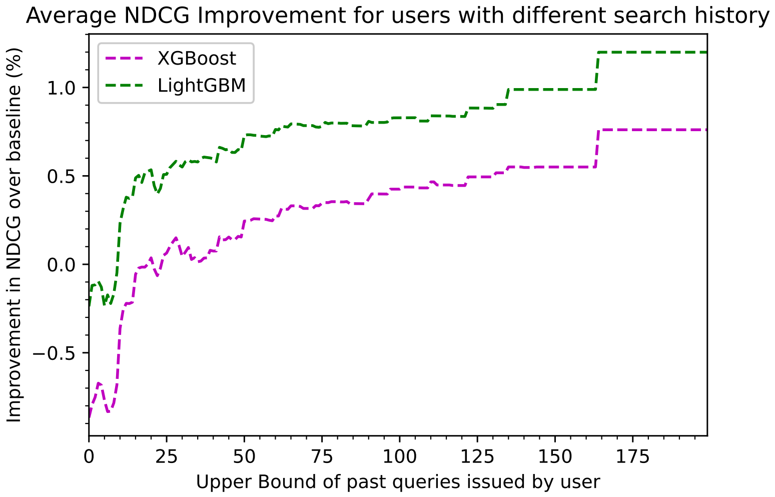


Figure 4.6: Average NDCG improvement on the reduced Yandex dataset for users with search history if length up to the specified bound.

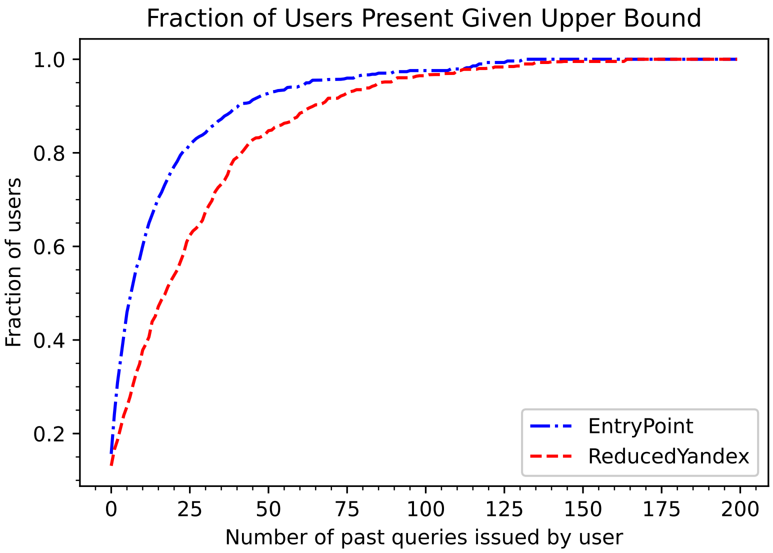


Figure 4.7: Fraction of users, of the entire dataset, for each value on the upper bound.

4.3 Discussion

In this section we discuss the results presented in Sections 4.1 and 4.2. We will compare the models using our different metrics and further analyze the long-term gain.

4.3.1 Performance and Features

Yandex

Table 4.3 shows that we managed to improve the $NDCG$ of the `red_yandex_dataset` when using *XGBoost* or *LightGBM* compared to the original list ordering. Further, we also improved the $AERC$, MRR, and $\Delta CTR@0$ metric. This is important since it validates our approach.

The `red_yandex_dataset` is seemingly more suitable for personalization than IKEA as was discussed in Section 4.1. Since this is the case, we cannot conclude anything on the potential improvements on the IKEA datasets from Yandex analysis alone.

We can compare our solution to the top *Kaggle* solutions. Yoganarasimhan (2020) achieved an improvement in $NDCG$ of 2.3%. Our best result was an improvement of 1.3% in $NDCG$. We speculate, that a larger search history results in more sophisticated personalization i.e. a larger increase in NDCG as motivated in 4.1. This is the case since the prediction, derived from the features, is based on past data, aside from the influence of the original ranking. Hence, we expect a higher accuracy in our prediction when considering more data.

Further, the gain in $NDCG$ is relatively small. This is however reasonable since many queries have resulted in a SERP with only one click at the top of the list. Such lists are already optimal and dilute the increase in $NDCG$ from the reranker. To further investigate the influence of

SERPs with single results, these could be removed and should be taken into consideration in further work.

IKEA

If we instead consider the IKEA datasets, Tables 4.1 and 4.2 show that reranking with our LambdaMART models yields an increase in $NDCG$ on the `entrypoint_dataset`. Since `entrypoint_dataset` is the larger dataset we will focus on its results. An increase in the optimization metric $NDCG$ indicates that our model works. With a reranked list, users tend to click higher in the list. This is further supported by the managerially relevant metrics $AERC$, MRR, and $\Delta CTR@0$ metrics upon which the models do not explicitly train. Notably, reranking with all three models yields an increase on the `entrypoint_dataset`.

When comparing the LambdaMART models to the *neural network*, the $NDCG$ gain is similar. This is surprising since the *neural network* does not explicitly train on this metric.

Features – number and importance

When removing features to create the pruned model, one might think that the features with low *feature importance* are bad, but this is not necessarily true. When creating the model, it is not necessary to have two features containing the same information. Assume that we have two features *country* and *company*. If there is only one company per country there is a 1-to-1 correspondence, and the information of one feature will be redundant when having the other. In essence, a feature need not be 'bad' if it has low feature importance, only redundant in combination with other features i.e. the feature has a high correlation with another feature with high importance. This is also true for linearly dependent features.

Another factor that influences the importance of features is the size of the dataset. Assume that document $d_1, d_2, ..., d_M$ are resulting documents in response to a query. Assume that d_1 is clicked almost every time and that d_2 are clicked slightly more often than $d_3, ..., d_M$. If that dataset is not large enough, it is likely that there *is not enough* history to support it is more relevant than the other document. When this is the case, the *variance* will be large and therefore the feature importance low. If the dataset is large, this feature might instead be well documented and hence highly relevant for personalization.

Performance (accuracy and efficiency)

All three models (*XGBoost*, *Neural Network* and *LightGBM*) have seemingly similar performance, when compared on the `entrypoint_dataset`. As the models are trained on relatively small datasets, the split (i.e. into past/training/validation/test-data) might have a great influence on the model performance which in turn makes it difficult to compare the accuracy of models.

To compare model accuracy we mainly study the $NDCG$ metric. For *EntryPoint*, there is an overall larger improvement of $NDCG$ when compared to *Yandex*. There could be multiple reasons for this:

1. **More information:** For the IKEA dataset, we have access to more information e.g. *country*, *company* etc. which are not disclosed in the `yandex_dataset`. Having more information translates to more features which in turn translates to a more detailed model.

2. **Performance of initial ranking:** The `red_yandex_dataset` is produced from a professional commercial search engine. It is reasonable to assume the initial ranking to be better in this dataset than in the IKEA datasets. Thus there is less to rerank so the overall improvement is less. It should be mentioned that the original $NDCG$ is higher in the `entrypoint_dataset` than the `red_yandex_dataset`. This does not contradict the assumption that *Yandex* has a better initial ranking, but is probably a result of more clicks being issued on SERPs.

4.3.2 Long Term Gain

Fig. 4.5 shows the average improvement in $NDCG$ depending on the size of the user history for the `entrypoint_dataset`. As mentioned, there is total difference in performance of $\sim 0.8\%$ (max = 5.4%, min = 4.6%) for XGBoost and $\sim 0.5\%$ (max = 4.9%, min = 4.4%). The plots are similar and it is interesting to study the overall structure. We get the largest average improvement when only including users with a search history of ~15 past queries or less. Including users with a larger search history does however reduce the overall improvement.

This is an interesting result since intuitively a user with a larger search history should benefit more from personalization since we have more information about that specific user. We thus expect the overall average increase to be larger when including users with larger search history, but this was not the case.

Now consider the feature importances in Sections B.1 and B.2, where we notice that the most important features are *global* and query-global. No user-related features are present among the top features. We thus suspect that most of the improvement is due to gain in *overall* data, rather than the search history of a single user. In contrast, studying the feature importances for Yandex in Sections B.3 and B.4, the second most important features for the Yandex models are user-related. We thus expect user search history to have a greater influence on the Yandex data influence than for IKEA.

Now consider figure 4.1, where we recollect that users, in general, have smaller search histories in the `entrypoint_dataset` compared to the `red_yandex_dataset`. This further supports that EntryPoint gains less from user-search history since there are not enough past queries to emphasize the user-related features. EntryPoint rather gains from features that aggregate users, either in the global dataset or a subset e.g. users from the same country. For Yandex, 4.6, we notice an increase in overall performance with user-search-history, which further supports our speculations.

4.3.3 Click Bias

In Section 2.2.3 we introduced the concept of click bias. Users may not know what information they are looking for, but rather click based on the position of the list. However, this presents an important difference between evaluating the model on a test dataset and in deployment. In a test dataset, user bias is introduced *before* reranking since we have gathered data on clicks where users have interacted with the original order of the SERP. In deployment, user bias would be introduced *after* reranking because the reranked SERP is presented to users. Since we were not allowed to deploy our model into production due to user integrity, it is difficult to say what the actual influence of index features will be. Considering Appendix B, we see that in our offline training index related features are deemed very important. Notably, SERP-rank is a top feature for all models.

4.3.4 Ethical Considerations

Since personalization models are derived from user data, we need to consider some ethical perspectives of our work to minimize the risk of harm to others.

First and foremost, to power our models we need to collect users' data. According to the European Convention on Human Rights, "Everyone has the right to respect for his private and family life, his home and his correspondence". This stands as background to the GDPR-data regulation law being passed in the European Union. It protects citizens when their data is being collected (Wolford, 2020). Should a personalization model be deployed, we need to be careful when collecting and handling data. We should not collect data on users that have not allowed us to do so, and we should be careful that this information is not leaked to any third party since only the specific user should be allowed to share their own data. It is not only against the law to not respect this, but also immoral since everyone has a right to their privacy at work.

It is also important to consider that the model itself contains private information. Since it learns at the user and group level, the model in combination with artificial (tailored to the user or group one wants information about) input can disclose information about users and groups. Providing input for a user from a specific country discloses what ranking is preferred by such a user, which is private information. It is therefore important that any user-related information is anonymized. Further, the model should not be public. Rather it should be regulated and protected by Non-Disclosure Agreements for everyone working with it.

Lastly, one should consider the consequences of personalization. Personalizing what news articles appear in a person's feed can close people into small boxes, only presenting them with conforming news. Similarly, news leading to many clicks can be prioritized over nuanced news that considers multiple aspects of an occurrence (Plattner, 2018). Since we mainly work on internal files, this is less applicable to our work. However *EntryPoint* does present (company-) news, so it is worth keeping in mind for future development.

A final comment is that we should recollect that we are working with search at work i.e. *internal* search. In other words, we are working with coworkers rather than private individuals. The information is kept within company boundaries which makes it less sensitive. Also, the reader is reminded that the users are anonymous.

Chapter 5

Conclusions

In the last chapter, we summarize our findings and discuss sources of errors as well as further work.

5.1 Summary of Findings

Returning to Section 1.1.2, we posed some research questions for our work.

Initially, we have shown that the search experience can be improved using personalization. We have created rerankers, using multiple machine learning frameworks, which has improved our chosen metrics implying that relevant documents are placed higher in a reranked result lists. We do however recognize that user bias is difficult to overcome, and that users may click more based on position of the document, rather than which information each document holds.

Further we have compared different models and frameworks. The LambdaMART implementations outperformed the Neural Network on the `entrypoint_dataset` and the `red_yandex_dataset`. For the `yandex_dataset` the pruned LightGBM model proved the best since it yielded the largest *NDCG* increase, while the pruned XGBoost model got the largest improvement on the `entrypoint_dataset`. The training was computationally-efficient.

Finally, we have provided a scaleable implementation for IKEA, as explained in Section 2.4. In addition, we provided an analysis of what information (which features) the models prioritized. The models for the `red_yandex_dataset` prioritized user specific features, while the models for the `entrypoint_dataset` instead emphasized aggregated user features.

5.2 Further work

5.2.1 Dwell-time

In Section 2.2.2, we introduced the notion of dwell time-based labeling. This can be an improvement on the click-based method since we introduce multiple levels of relevancy. To label the documents using the dwell time method we introduce *dwell time* as the time a user stays (dwells) on a document after clicking it before clicking another. We assume that if a user is not happy with the information, he/she would click on a new document. If a user dwells longer on a document, it is likely to be more relevant.

For `yandex_dataset` we can use the same relevancy labels as in the *Kaggle* competition. Assume a document d has relevancy r, then:

$r = 0$: A document is irrelevant if it has not been clicked or the click it receives has a dwell time strictly less than 50 time units and is not the last click in the session,

$r = 1$: A document is relevant if it receives a click with dwell time larger than 50 and less than 400 time units and it is not the last click in a session,

$r = 2$: A document is highly relevant if it is the last click in a session or the click has a dwell time of 400 time units or more.

Yandex assumes the last click a user performs to indicate high relevancy since the user has found a satisfying result.

For the `entrypoint_dataset` and `intranet_dataset` we use a similar definition. However, since sessions are reported differently we only consider result pages and different thresholds for dwell time. Specifically, the following relevancy labels are used:

$r = 0$: A document is irrelevant if it has not been clicked or the click it receives has a dwell time strictly less than 500 milliseconds and is not the last click on the result page,

$r = 1$: A document is relevant if it receives a click with a dwell time larger than 500 and less than 20000 milliseconds and assuming that it is not the last click on a result page,

$r = 2$: A document is highly relevant if it is the last click on a result page or the click has a dwell time of 20000 milliseconds or more.

While the dwell-based method was attempted in this thesis, we could not verify the assumption that users dwelling longer on documents found them more relevant for the IKEA datasets. We thus settled for the simpler metric. A further study of user behavior in addition to metrics with more relevancy labels could improve the performance of the models. In addition to Kaggle's Personalized search competition (which was based on the search engine *Yandex*) (Kaggle, 2013) providing three relevancy labels, Microsofts learning to rank dataset `MSLR-WEB30k` introduced 5 relevancy labels (Qin and Liu, 2010). In other words, large companies developing personalization models seem to favor non-binary relevancy labels.

Data

One of our limitations in this project was the data, or rather the lack thereof. Our computers could not process the full `red_yandex_dataset`. While the `entrypoint_dataset` was reasonably large, both IKEA datasets were tiny in comparison to the full `yandex_dataset`. To get a further understanding of the of the effect of personalization we suggest:

1. The model is to be deployed to a server (or a more powerful machine) that can handle large datasets,
2. More data from IKEA is to be collected, after which the models should be evaluated again.

With more data, the result should naturally improve. In addition, we implemented many features in this project. To avoid overfitting we had to scale down our model to fewer features. With a larger dataset, we can use more features without causing the model to overfit. In this case, we might find that some of the features we removed were in fact relevant.

5.2.2 Deployment to Evaluate Click Bias

We discussed earlier in Section 4.3.3 that user is introduced differently in offline training and in deployment. It is therefore difficult to fully grasp the impact of index related features. For future work, it would be interesiting to remove features related to the index and instead rerank only based on the document (URL, domain, category, etc.). We will likely see a reduction in the $NDCG$ score due to the user bias, but using a small number of users as "Guinea Pigs" (deploying the model only for them) and evaluating their behavior e.g. click position could provide a basis for proper evaluation.

5.2.3 Promoted links

Further analysis of the impact of *promoted* links could validate our model. We remind the reader that $\sim 96\%$ of SERPs containing a *promoted* link resulted in a click on a *promoted* link. These are navigational queries and do not benefit from personalization. Since *promoted* links are pinned to the top of a SERP they do not need reranking. One suggestion to battle these problems is to over-sample data with clicks further down the SERP i.e. clicks at positions $> k$, as introduced by Slack (2017). It would be interesting to study if this would improve the results since the model could better learn if should the result should be reranked. Another approach is to study the user behavior at IKEA. Understanding how users interact with the EntryPoint and IntraNet search engines gives insight into which queries should be reranked. From our understanding, there is a discrepancy between users' attitudes toward search. Some use search engines to find recently visited URLs and expect them to be ranked highly on the SERP. Others use tools such as bookmarks to find recent documents and see search engines as a 'last resort' when looking for information. Such users do not benefit from placing recent documents towards the top of a SERP. A better understanding of user behavior could help when selecting features and improving the model.

5.2.4 Neural Networks

In our thesis we have mainly focused on the *LambdaMART* algorithm and the two implementations *XGBoost* and *LightGBM*. We included a standard *neural network* as reference. However, the implementation of the *neural network* could be improved.

A *neural network* is not a poor choice for learning to rank problems. *RankNet*, the predecessor to *LambdaMART*, is based on a *neural network*. Further, *neural networks* generally perform well on many machine learning problems. However, since users expect results in milliseconds in real-time they may take too long. Ensembles of regression tree algorithms (e.g. *LambdaMART*) rerank much faster and also have the advantage of training much faster too.

As mentioned in Section 1.2, Nardini et al. (2022) present an implementation of a learning to rank *neural network*. On the dataset `MSN30K` (Qin and Liu, 2013) the *neural network* outperforms *LightGBM* and achieves $4.4x$ faster execution time when scoring than *LightGBM* without losing accuracy. This is done by levering multiple cutting edge approaches such as

> efficiency oriented pruning techniques and high-performance Dense and Sparse Matrix Multiplication techniques.

The report by Nardini et al. (2022) was published during our project (*2022-02-22*) and a further study of *neural networks* from this perspective would be an interesting addition to compare with the evaluation of *LambdaMART* implementations.

Appendix A
IntraNet Results

Tab. A.1 and A.2 shows the result for the `intranet_dataset`. As in Section 4.2, the columns with head ‘Original’ and ‘Optimal’ provide the results for the original result lists provided by the search engine, and the optimally reranked list. Notably, the full XGBoost model gets the best improvement on NDCG (0.93%). The model also improved the other metrics.

Curiously, the LightGBM model, in particular the pruned version, performs better than the XGBoost model on the $CTR@0$ metric.

	Original	Optimal	XGBoost	Pruned XGBoost	LightGBM	Pruned LightGBM	Neural Network
MRR	0.815	1.0	0.825	0.822	0.82	0.824	0.72
DCG	980.975	1150.359	990.407	988.094	988.540	988.430	980.975
NDCG	0.856	1.0	0.864	0.862	0.863	0.863	0.782
AERC	0.965	0.0	0.829	0.853	0.872	0.893	1.89
Num. features	–	–	589	226	595	90	589

Table A.1: Model performance on the IntraNet dataset.

k	Optimal	XGBoost	Pruned XGBoost	LightGBM	Pruned LightGBM	Neural Network
0	24.728	0.24	0.24	0.559	0.889	-7.807
1	-5.028	1.827	0.921	0.634	-0.109	-4.271
2	-5.718	-1.102	-1.267	-0.343	- 0.508	-2.137
3	-4.054	-0.43	0.641	-0.09	-0.09	1.42
4	-2.317	0.155	0.237	-0.252	0.491	0.421
5	-1.489	0.241	0.57	0.823	0.328	2.077
6	-1.737	0.899	0.158	-0.333	-0.333	1.331
7	-1.654	-0.419	-0.501	-0.663	-0.168	1.414
8	-1.323	-0.829	-0.5	0.328	-0.085	2.657
9	-1.406	-0.582	-0.5	-0.663	-0.415	4.896

Table A.2: $CTR@k$ for the IntraNet dataset.

Appendix B

Feature Importance

In this chapter, we provide the ~95% most important features in our full models. These are included to get an understanding of which features our models deemed important. Using the $EventListings(\theta_1, \theta_2, \theta_3)$ notation from Section 2.2.4, the features are in the form:

“θ_2->θ_3 : θ_1”

B.1 EntryPoint dataset *XGBoost*

Number	Feature	Importance	Cumulative
173	Term1->url: WClicks/Shows	52.379 %	52.379 %
0	SERP-rank	9.427 %	61.806 %
171	Term1->url: Clicks/Shows	3.513 %	65.319 %
47	Query->url: WClicks/Shows	2.749 %	68.068 %
186	Term1->category: ClicksTrue/Shows	1.97 %	70.038 %
52	Query->url: WShows	1.211 %	71.249 %
43	Query->url: Shows	0.878 %	72.127 %
180	Term1->category: Shows	0.809 %	72.936 %
381	Company->url: Clicks/Shows	0.766 %	73.703 %
184	Term1->category: WClicks/Shows	0.759 %	74.462 %
518	BusinessUnit->category: Clicks/Shows	0.736 %	75.198 %
182	Term1->category: Clicks/Shows	0.706 %	75.904 %
149	Query+UserID->domain: Shows	0.644 %	76.548 %
228	Term2->category: ClicksTrue/Shows	0.573 %	77.122 %
10	Global->url: WShows	0.519 %	77.641 %
65	Query->domain: Shows	0.45 %	78.091 %
425	Site->url: WClicks/Shows	0.414 %	78.505 %
51	Query->url: Skips/Shows	0.38 %	78.885 %
169	Term1->url: Shows	0.374 %	79.259 %
45	Query->url: Clicks/Shows	0.365 %	79.623 %
49	Query->url: ClicksTrue/Shows	0.354 %	79.977 %
54	Query->category: Shows	0.335 %	80.312 %
343	Country->url: ClicksTrue/Shows	0.327 %	80.639 %
339	Country->url: Clicks/Shows	0.257 %	80.896 %
74	Query->domain: WShows	0.237 %	81.133 %
383	Company->url: WClicks/Shows	0.2 %	81.333 %
175	Term1->url: ClicksTrue/Shows	0.198 %	81.531 %
559	Day->category: Clicks	0.197 %	81.727 %
435	Site->category: WClicks	0.191 %	81.919 %
46	Query->url: WClicks	0.191 %	82.11 %
476	JobTitle->category: Clicks/Shows	0.182 %	82.292 %
224	Term2->category: Clicks/Shows	0.165 %	82.457 %
1	Global->url: Shows	0.165 %	82.622 %
157	Query+UserID->domain: Skips/Shows	0.164 %	82.786 %
213	Term2->url: Clicks/Shows	0.162 %	82.948 %
22	Global->category: WShows/Shows	0.161 %	83.109 %
60	Query->category: ClicksTrue/Shows	0.16 %	83.269 %
385	Company->url: ClicksTrue/Shows	0.156 %	83.425 %
129	Query+UserID->url: Clicks/Shows	0.155 %	83.58 %
158	Query+UserID->domain: WShows	0.153 %	83.733 %
63	Query->category: WShows	0.15 %	83.883 %

Continued on next page

Table B.1 – continued from previous page

Number	Feature	Importance	Cumulative
522	BusinessUnit->category: ClicksTrue/Shows	0.15 %	84.032 %
58	Query->category: WClicks/Shows	0.145 %	84.178 %
138	Query+UserID->category: Shows	0.142 %	84.319 %
341	Country->url: WClicks/Shows	0.137 %	84.456 %
7	Global->url: ClicksTrue/Shows	0.137 %	84.593 %
148	Query+UserID->category: WShows/Shows	0.137 %	84.729 %
131	Query+UserID->url: WClicks/Shows	0.135 %	84.865 %
509	BusinessUnit->url: WClicks/Shows	0.135 %	85.0 %
469	JobTitle->url: ClicksTrue/Shows	0.133 %	85.133 %
566	Day->category: Skips/Shows	0.132 %	85.265 %
571	Day->domain: Clicks/Shows	0.128 %	85.392 %
64	Query->category: WShows/Shows	0.127 %	85.519 %
423	Site->url: Clicks/Shows	0.121 %	85.639 %
89	UserID->url: WClicks/Shows	0.119 %	85.758 %
345	Country->url: Skips/Shows	0.116 %	85.875 %
142	Query+UserID->category: WClicks/Shows	0.116 %	85.991 %
24	Global->domain: Clicks	0.111 %	86.102 %
44	Query->url: Clicks	0.109 %	86.211 %
12	Global->category: Shows	0.107 %	86.318 %
511	BusinessUnit->url: ClicksTrue/Shows	0.104 %	86.422 %
436	Site->category: WClicks/Shows	0.102 %	86.524 %
492	JobTitle->domain: Skips	0.102 %	86.626 %
216	Term2->url: ClicksTrue	0.101 %	86.727 %
133	Query+UserID->url: ClicksTrue/Shows	0.101 %	86.828 %
480	JobTitle->category: ClicksTrue/Shows	0.095 %	86.923 %
468	JobTitle->url: ClicksTrue	0.095 %	87.018 %
530	BusinessUnit->domain: WClicks	0.095 %	87.113 %
130	Query+UserID->url: WClicks	0.092 %	87.205 %
59	Query->category: ClicksTrue	0.089 %	87.294 %
88	UserID->url: WClicks	0.086 %	87.38 %
3	Global->url: Clicks/Shows	0.084 %	87.464 %
9	Global->url: Skips/Shows	0.083 %	87.547 %
99	UserID->category: WClicks	0.082 %	87.629 %
494	JobTitle->domain: WShows	0.082 %	87.712 %
66	Query->domain: Clicks	0.082 %	87.793 %
53	Query->url: WShows/Shows	0.081 %	87.874 %
188	Term1->category: Skips/Shows	0.081 %	87.955 %
117	UserID->domain: WShows/Shows	0.081 %	88.036 %
100	UserID->category: WClicks/Shows	0.08 %	88.116 %
197	Term1->domain: ClicksTrue/Shows	0.08 %	88.196 %
177	Term1->url: Skips/Shows	0.08 %	88.276 %
102	UserID->category: ClicksTrue/Shows	0.079 %	88.355 %
170	Term1->url: Clicks	0.079 %	88.434 %

Continued on next page

Table B.1 – continued from previous page

Number	Feature	Importance	Cumulative
257	Term3->url: WClicks/Shows	0.079 %	88.513 %
520	BusinessUnit->category: WClicks/Shows	0.078 %	88.591 %
478	JobTitle->category: WClicks/Shows	0.078 %	88.669 %
199	Term1->domain: Skips/Shows	0.077 %	88.745 %
253	Term3->url: Shows	0.077 %	88.822 %
179	Term1->url: WShows/Shows	0.076 %	88.898 %
379	Company->url: Shows	0.076 %	88.974 %
135	Query+UserID->url: Skips/Shows	0.075 %	89.049 %
397	Company->category: Skips	0.075 %	89.124 %
134	Query+UserID->url: Skips	0.075 %	89.198 %
465	JobTitle->url: Clicks/Shows	0.074 %	89.273 %
198	Term1->domain: Skips	0.074 %	89.346 %
190	Term1->category: WShows/Shows	0.074 %	89.42 %
62	Query->category: Skips/Shows	0.073 %	89.493 %
387	Company->url: Skips/Shows	0.073 %	89.566 %
405	Company->domain: WClicks/Shows	0.072 %	89.638 %
30	Global->domain: Skips	0.072 %	89.71 %
172	Term1->url: WClicks	0.072 %	89.782 %
340	Country->url: WClicks	0.071 %	89.853 %
5	Global->url: WClicks/Shows	0.07 %	89.924 %
214	Term2->url: WClicks	0.07 %	89.993 %
346	Country->url: WShows	0.07 %	90.063 %
146	Query+UserID->category: Skips/Shows	0.068 %	90.131 %
481	JobTitle->category: Skips	0.067 %	90.199 %
178	Term1->url: WShows	0.064 %	90.262 %
388	Company->url: WShows	0.064 %	90.326 %
555	Day->url: Skips/Shows	0.063 %	90.389 %
75	Query->domain: WShows/Shows	0.063 %	90.452 %
11	Global->url: WShows/Shows	0.063 %	90.515 %
70	Query->domain: ClicksTrue	0.063 %	90.577 %
255	Term3->url: Clicks/Shows	0.062 %	90.639 %
411	Company->domain: WShows/Shows	0.062 %	90.701 %
429	Site->url: Skips/Shows	0.061 %	90.762 %
217	Term2->url: ClicksTrue/Shows	0.059 %	90.821 %
183	Term1->category: WClicks	0.059 %	90.88 %
356	Country->category: Skips/Shows	0.059 %	90.939 %
445	Site->domain: Clicks/Shows	0.058 %	90.997 %
260	Term3->url: Skips	0.058 %	91.055 %
191	Term1->domain: Shows	0.058 %	91.113 %
347	Country->url: WShows/Shows	0.057 %	91.17 %
153	Query+UserID->domain: WClicks/Shows	0.056 %	91.226 %
230	Term2->category: Skips/Shows	0.056 %	91.281 %
73	Query->domain: Skips/Shows	0.056 %	91.337 %

Continued on next page

Table B.1 – continued from previous page

Number	Feature	Importance	Cumulative
557	Day->url: WShows/Shows	0.055 %	91.392 %
513	BusinessUnit->url: Skips/Shows	0.055 %	91.447 %
97	UserID->category: Clicks	0.054 %	91.502 %
485	JobTitle->domain: Shows	0.054 %	91.556 %
8	Global->url: Skips	0.054 %	91.61 %
71	Query->domain: ClicksTrue/Shows	0.054 %	91.663 %
358	Country->category: WShows/Shows	0.053 %	91.717 %
380	Company->url: Clicks	0.053 %	91.77 %
181	Term1->category: Clicks	0.053 %	91.823 %
409	Company->domain: Skips/Shows	0.053 %	91.876 %
321	Term4->domain: WClicks/Shows	0.053 %	91.929 %
185	Term1->category: ClicksTrue	0.053 %	91.981 %
193	Term1->domain: Clicks/Shows	0.052 %	92.033 %
441	Site->category: WShows	0.052 %	92.085 %
349	Country->category: Clicks	0.051 %	92.136 %
219	Term2->url: Skips/Shows	0.051 %	92.187 %
221	Term2->url: WShows/Shows	0.05 %	92.237 %
72	Query->domain: Skips	0.05 %	92.288 %
239	Term2->domain: ClicksTrue/Shows	0.05 %	92.337 %
150	Query+UserID->domain: Clicks	0.05 %	92.387 %
91	UserID->url: ClicksTrue/Shows	0.049 %	92.436 %
232	Term2->category: WShows/Shows	0.049 %	92.485 %
526	BusinessUnit->category: WShows/Shows	0.049 %	92.534 %
495	JobTitle->domain: WShows/Shows	0.049 %	92.582 %
67	Query->domain: Clicks/Shows	0.049 %	92.631 %
337	Country->url: Shows	0.048 %	92.679 %
4	Global->url: WClicks	0.048 %	92.727 %
556	Day->url: WShows	0.048 %	92.775 %
115	UserID->domain: Skips/Shows	0.048 %	92.823 %
442	Site->category: WShows/Shows	0.048 %	92.871 %
515	BusinessUnit->url: WShows/Shows	0.047 %	92.918 %
189	Term1->category: WShows	0.047 %	92.965 %
533	BusinessUnit->domain: ClicksTrue/Shows	0.047 %	93.013 %
112	UserID->domain: ClicksTrue	0.047 %	93.06 %
443	Site->domain: Shows	0.047 %	93.107 %
98	UserID->category: Clicks/Shows	0.047 %	93.154 %
261	Term3->url: Skips/Shows	0.047 %	93.201 %
431	Site->url: WShows/Shows	0.046 %	93.246 %
519	BusinessUnit->category: WClicks	0.046 %	93.292 %
482	JobTitle->category: Skips/Shows	0.046 %	93.338 %
360	Country->domain: Clicks	0.046 %	93.383 %
367	Country->domain: Skips/Shows	0.045 %	93.429 %
144	Query+UserID->category: ClicksTrue/Shows	0.045 %	93.474 %

Continued on next page

Table B.1 – continued from previous page

Number	Feature	Importance	Cumulative
233	Term2->domain: Shows	0.045 %	93.518 %
242	Term2->domain: WShows	0.045 %	93.563 %
471	JobTitle->url: Skips/Shows	0.045 %	93.608 %
561	Day->category: WClicks	0.045 %	93.652 %
472	JobTitle->url: WShows	0.044 %	93.696 %
525	BusinessUnit->category: WShows	0.044 %	93.741 %
449	Site->domain: ClicksTrue/Shows	0.044 %	93.785 %
487	JobTitle->domain: Clicks/Shows	0.044 %	93.829 %
467	JobTitle->url: WClicks/Shows	0.044 %	93.873 %
222	Term2->category: Shows	0.044 %	93.917 %
432	Site->category: Shows	0.044 %	93.961 %
537	BusinessUnit->domain: WShows/Shows	0.044 %	94.005 %
241	Term2->domain: Skips/Shows	0.044 %	94.048 %
240	Term2->domain: Skips	0.044 %	94.092 %
399	Company->category: WShows	0.044 %	94.136 %
489	JobTitle->domain: WClicks/Shows	0.044 %	94.179 %
235	Term2->domain: Clicks/Shows	0.044 %	94.223 %
390	Company->category: Shows	0.044 %	94.266 %
55	Query->category: Clicks	0.043 %	94.31 %
355	Country->category: Skips	0.043 %	94.353 %
187	Term1->category: Skips	0.043 %	94.397 %
488	JobTitle->domain: WClicks	0.043 %	94.44 %
2	Global->url: Clicks	0.043 %	94.483 %
484	JobTitle->category: WShows/Shows	0.042 %	94.525 %
516	BusinessUnit->category: Shows	0.042 %	94.567 %
439	Site->category: Skips	0.042 %	94.609 %
510	BusinessUnit->url: ClicksTrue	0.042 %	94.651 %
192	Term1->domain: Clicks	0.042 %	94.693 %
92	UserID->url: Skips	0.042 %	94.734 %
493	JobTitle->domain: Skips/Shows	0.042 %	94.776 %
201	Term1->domain: WShows/Shows	0.042 %	94.817 %
475	JobTitle->category: Clicks	0.042 %	94.859 %
434	Site->category: Clicks/Shows	0.041 %	94.9 %
111	UserID->domain: WClicks/Shows	0.041 %	94.942 %
338	Country->url: Clicks	0.041 %	94.983 %
344	**Country->url: Skips**	**0.041** %	**95.024** %

B.2 EntryPoint dataset *LightGBM*

Number	Feature	Importance	Cumulative
173	Term1->url: WClicks/Shows	27.86 %	27.86 %
0	SERP-rank	18.415 %	46.275 %
175	Term1->url: ClicksTrue/Shows	8.75 %	55.025 %
171	Term1->url: Clicks/Shows	8.313 %	63.338 %
594	JobTitle	4.173 %	67.511 %
47	Query->url: WClicks/Shows	4.093 %	71.604 %
595	BusinessUnit	3.584 %	75.188 %
54	Query->category: Shows	3.348 %	78.537 %
593	Site	3.081 %	81.618 %
182	Term1->category: Clicks/Shows	2.395 %	84.013 %
49	Query->url: ClicksTrue/Shows	2.24 %	86.253 %
45	Query->url: Clicks/Shows	1.81 %	88.063 %
341	Country->url: WClicks/Shows	0.885 %	88.948 %
65	Query->domain: Shows	0.804 %	89.752 %
343	Country->url: ClicksTrue/Shows	0.725 %	90.477 %
339	Country->url: Clicks/Shows	0.605 %	91.083 %
186	Term1->category: ClicksTrue/Shows	0.522 %	91.605 %
184	Term1->category: WClicks/Shows	0.507 %	92.112 %
58	Query->category: WClicks/Shows	0.455 %	92.567 %
9	Global->url: Skips/Shows	0.44 %	93.008 %
180	Term1->category: Shows	0.432 %	93.439 %
385	Company->url: ClicksTrue/Shows	0.396 %	93.835 %
56	Query->category: Clicks/Shows	0.367 %	94.202 %
7	Global->url: ClicksTrue/Shows	0.272 %	94.475 %
590	Category	0.272 %	94.747 %
43	Query->url: Shows	0.258 %	95.005 %

B.3 *Yandex* dataset *XGBoost*

Number	Feature	Importance	Cumulative
0	SERP-rank	44.18 %	44.18 %
67	UserID->url: WClicks/Shows	4.204 %	48.384 %
97	Query+UserID->url: WClicks	2.905 %	51.289 %
98	Query+UserID->url: WClicks/Shows	2.383 %	53.672 %
115	Query+UserID->domain: WShows/Shows	2.297 %	55.97 %
129	Term1->url: WClicks/Shows	1.653 %	57.623 %

Continued on next page

Table B.3 – continued from previous page

Number	Feature	Importance	Cumulative
100	Query+UserID->url: ClicksTrue/Shows	1.499 %	59.122 %
54	Query->index: Shows	1.187 %	60.309 %
36	Query->url: WClicks/Shows	1.05 %	61.36 %
101	Query+UserID->url: Skips	0.994 %	62.353 %
114	Query+UserID->domain: WShows	0.885 %	63.238 %
103	Query+UserID->url: WShows	0.863 %	64.101 %
52	Query->domain: WShows	0.703 %	64.804 %
80	UserID->domain: ClicksTrue/Shows	0.606 %	65.411 %
160	Term2->url: WClicks/Shows	0.543 %	65.953 %
105	Query+UserID->domain: Shows	0.528 %	66.481 %
32	Query->url: Shows	0.508 %	66.989 %
70	UserID->url: Skips	0.503 %	67.492 %
78	UserID->domain: WClicks/Shows	0.485 %	67.977 %
5	Global->url: WClicks/Shows	0.451 %	68.428 %
162	Term2->url: ClicksTrue/Shows	0.427 %	68.855 %
153	Term1->index: ClicksTrue/Shows	0.419 %	69.274 %
112	Query+UserID->domain: Skips	0.399 %	69.673 %
193	Term3->url: ClicksTrue/Shows	0.383 %	70.056 %
96	Query+UserID->url: Clicks/Shows	0.379 %	70.435 %
79	UserID->domain: ClicksTrue	0.375 %	70.81 %
7	Global->url: ClicksTrue/Shows	0.355 %	71.165 %
131	Term1->url: ClicksTrue/Shows	0.352 %	71.516 %
127	Term1->url: Clicks/Shows	0.319 %	71.835 %
68	UserID->url: ClicksTrue	0.296 %	72.131 %
66	UserID->url: WClicks	0.296 %	72.427 %
157	Term2->url: Clicks	0.294 %	72.721 %
120	Query+UserID->index: WClicks/Shows	0.291 %	73.012 %
77	UserID->domain: WClicks	0.271 %	73.283 %
233	Term4->domain: WClicks/Shows	0.268 %	73.551 %
113	Query+UserID->domain: Skips/Shows	0.266 %	73.817 %
158	Term2->url: Clicks/Shows	0.259 %	74.076 %
3	Global->url: Clicks/Shows	0.254 %	74.33 %
72	UserID->url: WShows	0.253 %	74.583 %
202	Term3->domain: WClicks/Shows	0.251 %	74.835 %
163	Term2->url: Skips	0.244 %	75.079 %
16	Global->domain: WClicks/Shows	0.244 %	75.323 %
38	Query->url: ClicksTrue/Shows	0.241 %	75.563 %
234	Term4->domain: ClicksTrue	0.235 %	75.799 %
130	Term1->url: ClicksTrue	0.235 %	76.034 %
238	Term4->domain: WShows	0.231 %	76.265 %
35	Query->url: WClicks	0.229 %	76.495 %
53	Query->domain: WShows/Shows	0.226 %	76.721 %
186	Term2->index: Skips/Shows	0.224 %	76.945 %

Continued on next page

Table B.3 – continued from previous page

Number	Feature	Importance	Cumulative
116	Query+UserID->index: Shows	0.22 %	77.164 %
14	Global->domain: Clicks/Shows	0.213 %	77.377 %
215	Term3->index: ClicksTrue/Shows	0.212 %	77.588 %
63	UserID->url: Shows	0.212 %	77.8 %
190	Term3->url: WClicks	0.211 %	78.011 %
168	Term2->domain: Clicks	0.211 %	78.222 %
203	Term3->domain: ClicksTrue	0.209 %	78.431 %
144	Term1->domain: Skips/Shows	0.206 %	78.637 %
149	Term1->index: Clicks/Shows	0.205 %	78.842 %
216	Term3->index: Skips	0.203 %	79.046 %
17	Global->domain: ClicksTrue	0.203 %	79.249 %
232	Term4->domain: WClicks	0.2 %	79.449 %
242	Term4->index: Clicks/Shows	0.199 %	79.648 %
21	Global->domain: WShows	0.199 %	79.847 %
155	Term1->index: Skips/Shows	0.196 %	80.042 %
156	Term2->url: Shows	0.195 %	80.237 %
244	Term4->index: WClicks/Shows	0.194 %	80.431 %
18	Global->domain: ClicksTrue/Shows	0.193 %	80.624 %
191	Term3->url: WClicks/Shows	0.192 %	80.815 %
61	Query->index: Skips	0.188 %	81.003 %
125	Term1->url: Shows	0.187 %	81.19 %
137	Term1->domain: Clicks	0.185 %	81.375 %
93	UserID->index: Skips/Shows	0.184 %	81.559 %
11	Global->url: WShows/Shows	0.184 %	81.743 %
159	Term2->url: WClicks	0.183 %	81.926 %
71	UserID->url: Skips/Shows	0.182 %	82.109 %
142	Term1->domain: ClicksTrue/Shows	0.182 %	82.291 %
12	Global->domain: Shows	0.182 %	82.473 %
172	Term2->domain: ClicksTrue	0.181 %	82.653 %
10	Global->url: WShows	0.18 %	82.833 %
184	Term2->index: ClicksTrue/Shows	0.178 %	83.012 %
212	Term3->index: WClicks	0.176 %	83.188 %
171	Term2->domain: WClicks/Shows	0.175 %	83.362 %
82	UserID->domain: Skips/Shows	0.174 %	83.536 %
239	Term4->domain: WShows/Shows	0.173 %	83.709 %
151	Term1->index: WClicks/Shows	0.172 %	83.881 %
196	Term3->url: WShows	0.17 %	84.052 %
147	Term1->index: Shows	0.168 %	84.219 %
76	UserID->domain: Clicks/Shows	0.167 %	84.386 %
152	Term1->index: ClicksTrue	0.167 %	84.553 %
211	Term3->index: Clicks/Shows	0.166 %	84.72 %
75	UserID->domain: Clicks	0.163 %	84.883 %
200	Term3->domain: Clicks/Shows	0.163 %	85.046 %

Continued on next page

Table B.3 – continued from previous page

Number	Feature	Importance	Cumulative
181	Term2->index: WClicks	0.163 %	85.209 %
207	Term3->domain: WShows	0.161 %	85.37 %
43	Query->domain: Shows	0.16 %	85.53 %
91	UserID->index: ClicksTrue/Shows	0.16 %	85.69 %
213	Term3->index: WClicks/Shows	0.16 %	85.85 %
42	Query->url: WShows/Shows	0.16 %	86.01 %
22	Global->domain: WShows/Shows	0.159 %	86.168 %
4	Global->url: WClicks	0.156 %	86.325 %
164	Term2->url: Skips/Shows	0.156 %	86.481 %
146	Term1->domain: WShows/Shows	0.156 %	86.637 %
85	UserID->index: Shows	0.153 %	86.79 %
140	Term1->domain: WClicks/Shows	0.153 %	86.944 %
210	Term3->index: Clicks	0.153 %	87.097 %
170	Term2->domain: WClicks	0.153 %	87.25 %
217	Term3->index: Skips/Shows	0.152 %	87.401 %
51	Query->domain: Skips/Shows	0.152 %	87.553 %
182	Term2->index: WClicks/Shows	0.152 %	87.705 %
177	Term2->domain: WShows/Shows	0.151 %	87.856 %
19	Global->domain: Skips	0.151 %	88.007 %
180	Term2->index: Clicks/Shows	0.151 %	88.157 %
55	Query->index: Clicks	0.15 %	88.307 %
58	Query->index: WClicks/Shows	0.149 %	88.456 %
237	Term4->domain: Skips/Shows	0.149 %	88.605 %
94	Query+UserID->url: Shows	0.149 %	88.754 %
206	Term3->domain: Skips/Shows	0.149 %	88.903 %
176	Term2->domain: WShows	0.149 %	89.052 %
135	Term1->url: WShows/Shows	0.148 %	89.2 %
87	UserID->index: Clicks/Shows	0.148 %	89.348 %
174	Term2->domain: Skips	0.148 %	89.496 %
208	Term3->domain: WShows/Shows	0.147 %	89.643 %
245	Term4->index: ClicksTrue	0.147 %	89.79 %
178	Term2->index: Shows	0.146 %	89.936 %
49	Query->domain: ClicksTrue/Shows	0.146 %	90.083 %
247	Term4->index: Skips	0.146 %	90.229 %
204	Term3->domain: ClicksTrue/Shows	0.145 %	90.373 %
167	Term2->domain: Shows	0.144 %	90.518 %
145	Term1->domain: WShows	0.144 %	90.662 %
83	UserID->domain: WShows	0.144 %	90.806 %
13	Global->domain: Clicks	0.142 %	90.948 %
86	UserID->index: Clicks	0.141 %	91.089 %
89	UserID->index: WClicks/Shows	0.141 %	91.23 %
74	UserID->domain: Shows	0.141 %	91.371 %
121	Query+UserID->index: ClicksTrue	0.14 %	91.511 %

Continued on next page

Table B.3 – continued from previous page

Number	Feature	Importance	Cumulative
24	Global->index: Clicks	0.139 %	91.65 %
209	Term3->index: Shows	0.139 %	91.789 %
138	Term1->domain: Clicks/Shows	0.138 %	91.927 %
195	Term3->url: Skips/Shows	0.138 %	92.066 %
56	Query->index: Clicks/Shows	0.138 %	92.204 %
9	Global->url: Skips/Shows	0.138 %	92.342 %
1	Global->url: Shows	0.138 %	92.479 %
185	Term2->index: Skips	0.137 %	92.617 %
240	Term4->index: Shows	0.136 %	92.753 %
148	Term1->index: Clicks	0.136 %	92.889 %
154	Term1->index: Skips	0.136 %	93.024 %
34	Query->url: Clicks/Shows	0.135 %	93.159 %
15	Global->domain: WClicks	0.134 %	93.294 %
62	Query->index: Skips/Shows	0.134 %	93.428 %
46	Query->domain: WClicks	0.134 %	93.562 %
109	Query+UserID->domain: WClicks/Shows	0.133 %	93.696 %
236	Term4->domain: Skips	0.133 %	93.829 %
248	Term4->index: Skips/Shows	0.133 %	93.962 %
246	Term4->index: ClicksTrue/Shows	0.132 %	94.095 %
92	UserID->index: Skips	0.132 %	94.227 %
235	Term4->domain: ClicksTrue/Shows	0.132 %	94.359 %
124	Query+UserID->index: Skips/Shows	0.131 %	94.49 %
221	Term4->url: WClicks	0.131 %	94.621 %
20	Global->domain: Skips/Shows	0.129 %	94.75 %
136	Term1->domain: Shows	0.129 %	94.879 %
6	Global->url: ClicksTrue	0.129 %	95.008 %

B.4 *Yandex* dataset *LightGBM*

Number	Feature	Importance	Cumulative
0	SERP-rank	77.736 %	77.736 %
78	UserID->domain: WClicks/Shows	0.937 %	78.674 %
222	Term4->url: WClicks/Shows	0.779 %	79.453 %
54	Query->index: Shows	0.629 %	80.082 %
80	UserID->domain: ClicksTrue/Shows	0.595 %	80.677 %
96	Query+UserID->url: Clicks/Shows	0.566 %	81.243 %
93	UserID->index: Skips/Shows	0.466 %	81.709 %
114	Query+UserID->domain: WShows	0.454 %	82.162 %
155	Term1->index: Skips/Shows	0.441 %	82.603 %
98	Query+UserID->url: WClicks/Shows	0.421 %	83.025 %
186	Term2->index: Skips/Shows	0.375 %	83.4 %
14	Global->domain: Clicks/Shows	0.371 %	83.77 %
82	UserID->domain: Skips/Shows	0.36 %	84.13 %
184	Term2->index: ClicksTrue/Shows	0.351 %	84.482 %
67	UserID->url: WClicks/Shows	0.347 %	84.829 %
153	Term1->index: ClicksTrue/Shows	0.316 %	85.145 %
160	Term2->url: WClicks/Shows	0.308 %	85.454 %
220	Term4->url: Clicks/Shows	0.302 %	85.756 %
77	UserID->domain: WClicks	0.299 %	86.054 %
16	Global->domain: WClicks/Shows	0.267 %	86.322 %
158	Term2->url: Clicks/Shows	0.236 %	86.557 %
213	Term3->index: WClicks/Shows	0.234 %	86.792 %
76	UserID->domain: Clicks/Shows	0.234 %	87.025 %
129	Term1->url: WClicks/Shows	0.219 %	87.244 %
22	Global->domain: WShows/Shows	0.214 %	87.458 %
3	Global->url: Clicks/Shows	0.208 %	87.666 %
36	Query->url: WClicks/Shows	0.198 %	87.864 %
32	Query->url: Shows	0.198 %	88.062 %
5	Global->url: WClicks/Shows	0.196 %	88.258 %
87	UserID->index: Clicks/Shows	0.196 %	88.454 %
92	UserID->index: Skips	0.191 %	88.645 %
178	Term2->index: Shows	0.183 %	88.828 %
11	Global->url: WShows/Shows	0.182 %	89.01 %
18	Global->domain: ClicksTrue/Shows	0.18 %	89.19 %
115	Query+UserID->domain: WShows/Shows	0.174 %	89.364 %
7	Global->url: ClicksTrue/Shows	0.171 %	89.535 %
85	UserID->index: Shows	0.17 %	89.705 %
149	Term1->index: Clicks/Shows	0.167 %	89.872 %
182	Term2->index: WClicks/Shows	0.167 %	90.039 %
151	Term1->index: WClicks/Shows	0.166 %	90.204 %
237	Term4->domain: Skips/Shows	0.162 %	90.366 %

Continued on next page

Table B.4 – continued from previous page

Number	Feature	Importance	Cumulative
229	Term4->domain: Shows	0.161 %	90.528 %
239	Term4->domain: WShows/Shows	0.159 %	90.686 %
89	UserID->index: WClicks/Shows	0.159 %	90.845 %
51	Query->domain: Skips/Shows	0.158 %	91.003 %
162	Term2->url: ClicksTrue/Shows	0.154 %	91.157 %
187	Term3->url: Shows	0.151 %	91.307 %
38	Query->url: ClicksTrue/Shows	0.148 %	91.455 %
52	Query->domain: WShows	0.147 %	91.603 %
166	Term2->url: WShows/Shows	0.146 %	91.748 %
175	Term2->domain: Skips/Shows	0.145 %	91.893 %
206	Term3->domain: Skips/Shows	0.144 %	92.037 %
20	Global->domain: Skips/Shows	0.138 %	92.176 %
217	Term3->index: Skips/Shows	0.138 %	92.314 %
72	UserID->url: WShows	0.135 %	92.449 %
71	UserID->url: Skips/Shows	0.13 %	92.578 %
112	Query+UserID->domain: Skips	0.128 %	92.706 %
248	Term4->index: Skips/Shows	0.122 %	92.828 %
116	Query+UserID->index: Shows	0.122 %	92.95 %
191	Term3->url: WClicks/Shows	0.12 %	93.069 %
180	Term2->index: Clicks/Shows	0.118 %	93.188 %
83	UserID->domain: WShows	0.117 %	93.305 %
198	Term3->domain: Shows	0.116 %	93.42 %
233	Term4->domain: WClicks/Shows	0.116 %	93.536 %
102	Query+UserID->url: Skips/Shows	0.111 %	93.646 %
91	UserID->index: ClicksTrue/Shows	0.11 %	93.756 %
146	Term1->domain: WShows/Shows	0.109 %	93.865 %
15	Global->domain: WClicks	0.101 %	93.966 %
144	Term1->domain: Skips/Shows	0.1 %	94.066 %
244	Term4->index: WClicks/Shows	0.099 %	94.165 %
228	Term4->url: WShows/Shows	0.099 %	94.264 %
70	UserID->url: Skips	0.098 %	94.362 %
236	Term4->domain: Skips	0.097 %	94.458 %
88	UserID->index: WClicks	0.096 %	94.554 %
231	Term4->domain: Clicks/Shows	0.096 %	94.65 %
179	Term2->index: Clicks	0.095 %	94.745 %
205	Term3->domain: Skips	0.094 %	94.839 %
13	Global->domain: Clicks	0.094 %	94.933 %
210	Term3->index: Clicks	0.089 %	95.022 %

References

Burges, C. J. (2010). From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81.

Chapelle, O. and Chang, Y. (2011). Yahoo! learning to rank challenge overview. In *JMLR: Workshop and Conference Proceedings 14*, pages 1–24.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA. Association for Computing Machinery.

de Vrieze, P. T. (2006). *Fundaments of Adaptive Personalisation*. PhD thesis, Radboud Universiteit, Nijmegen.

Dou, Z., Song, R., and Wen, J.-R. (2007). A large-scale evaluation and analysis of personalized search strategies. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 581–590, New York, NY, USA. Association for Computing Machinery.

Kaggle (2013). Personalized web search challenge. `https://www.kaggle.com/c/yandex-personalized-web-search-challenge`. Accessed: 2022-03-25.

Kaggle (2014). Personalized web search challenge. `https://www.kaggle.com/competitions/yandex-personalized-web-search-challenge/discussion/6489`. Acessed: 2022-06-09.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3149–3157, Red Hook, NY, USA. Curran Associates Inc.

LightGBM (2021). Lightgbm read the docs. `https://lightgbm.readthedocs.io/en/latest/`. Acessed: 2022-05-13.

Nardini, F. M., Rulli, C., Trani, S., and Venturini, R. (2022). Distilled neural networks for efficient learning to rank. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1.

Plattner, T. (2018). Five risks of news personalization. `https://medium.com/jsk-class-of-2018/five-risks-of-news-personalizations-5bdc97fdbdcc`. Acessed: 2022-06-09.

Qin, T. and Liu, T. (2010). Microsoft learning to rank datasets. `https://www.microsoft.com/en-us/research/project/mslr/#:~:text=%20Microsoft%20Learning%20to%20Rank%20Datasets%20%201,the%20datasets%2C%20you%20must%20read%20and...%20More%20`. Acessed: 2022-05-17.

Qin, T. and Liu, T. (2013). Introducing LETOR 4.0 datasets. *CoRR*, abs/1306.2597.

Ricci, F., Rokach, L., and Shapira, B. (2011). *Introduction to Recommender Systems Handbook*. Springer US, Boston, MA.

Russell, S. and Norvig, P. (2016). *Artificial Intelligence: A Modern Apporach, 3rd ed.* Pearson Education Limited, Edinburgh Gate, Harlow, Essex CM20 2JE, England.

Slack (2017). Search at slack. `https://slack.engineering/search-at-slack/`. Accessed: 2022-05-04.

Ursu, R. M. (2018). The Power of Rankings: Quantifying the Effect of Rankings on Online Consumer Search and Purchase Decisions. *Marketing Science*, 37(4):530–552.

Wolford, B. (2020). What is gdpr, the eu's new data protection law? `https://gdpr.eu/what-is-gdpr/`. Acessed: 2022-06-09.

XGBoost (2021). Xgboost documentation. `https://xgboost.readthedocs.io/en/stable/`. Acessed: 2022-05-05.

Ye, A. (2020). Xgboost, lightgbm, and other kaggle competition favorites. `https://medium.com/analytics-vidhya/xgboost-lightgbm-and-other-kaggle-competition-favorites-6212e8b0e835`. Acessed: 2022-05-13.

Yoganarasimhan, H. (2020). Search personalization using machine learning. *Management Science*, 66(3):1045–1070.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2022-06-03

EXAMENSARBETE Personalizing the Order of Search Results Using Machine Learning
STUDENTER Liam Fahlstad, Max Gustafson
HANDLEDARE Emelie Lundh (INGKA), Pierre Nugues (LTH)
EXAMINATOR Flavius Gruian (LTH)

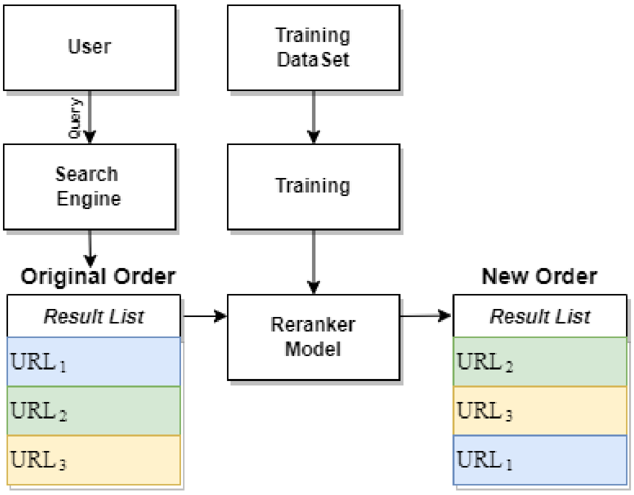
Personaliserad företagssök genom maskininlärning

POPULÄRVETENSKAPLIG SAMMANFATTNING **Liam Fahlstad, Max Gustafson**

Dagens sökmotorer, inte minst 'Google', använder sig ofta av personalisering för att producera relevanta resultat för användaren. Detta examensarbete har tagit fram en 'omrankingsmodell' för IKEAs interna sökplatformar som placerar dokument relevanta för användaren högre upp i den returnerade listan med träffar.

När en användare brukar en kommersiell sökmotor beror den resulterande listan med träffar i stor utsträckning på vem som är användare. En användare som befinner sig i Lund och söker på 'restauranger' förväntar sig resultat på restauranger i Lund, inte i Malmö. På stora företag, t.ex. med många olika avdelningar, kan samma princip appliceras på deras interna företagssök. En användare från Sverige som jobbar på lager förväntar sig annorlunda resultat på 'Monthly Report' än en användare som jobbar med IT i Kananda. Tidigare studier har visat att om relevanta resultat hamnar för långt ner på resultatlistan är det sannolikt att användaren ger upp sin söksession.

När en användare utfärdar en sökning, indexerar och returnerar sökmotorn en lista med sökresultat. I detta examensarbete, har sökhistoriken på ett stort företag, IKEA, använts för att bygga en *omrankningsmodell*. Innan en lista med sökresultat presenteras, bearbetas den av omrankingsmodell som tar hänsyn till användarens sökhistorik, samt sökhistorik från liknande användare. Modellen ger därefter en ny rankning till varje sökresultat och presenterar listan i ny ordning där relevanta resultat för användaren befinner sig högre upp. Omrankningsmodellen är baserad på maskininlärning vilket betyder att den lär sig att lösa problemet med hjälp av tidigare data, i vårt fall sökhistorik.



Konceptuell beskrivning av en omrankingsmodell.

Flera olika modeller byggdes där samtliga lyckades förflytta relevanta sökresultat högre upp i listan dvs. att användare i större utsträckning klickar högre upp i listan.