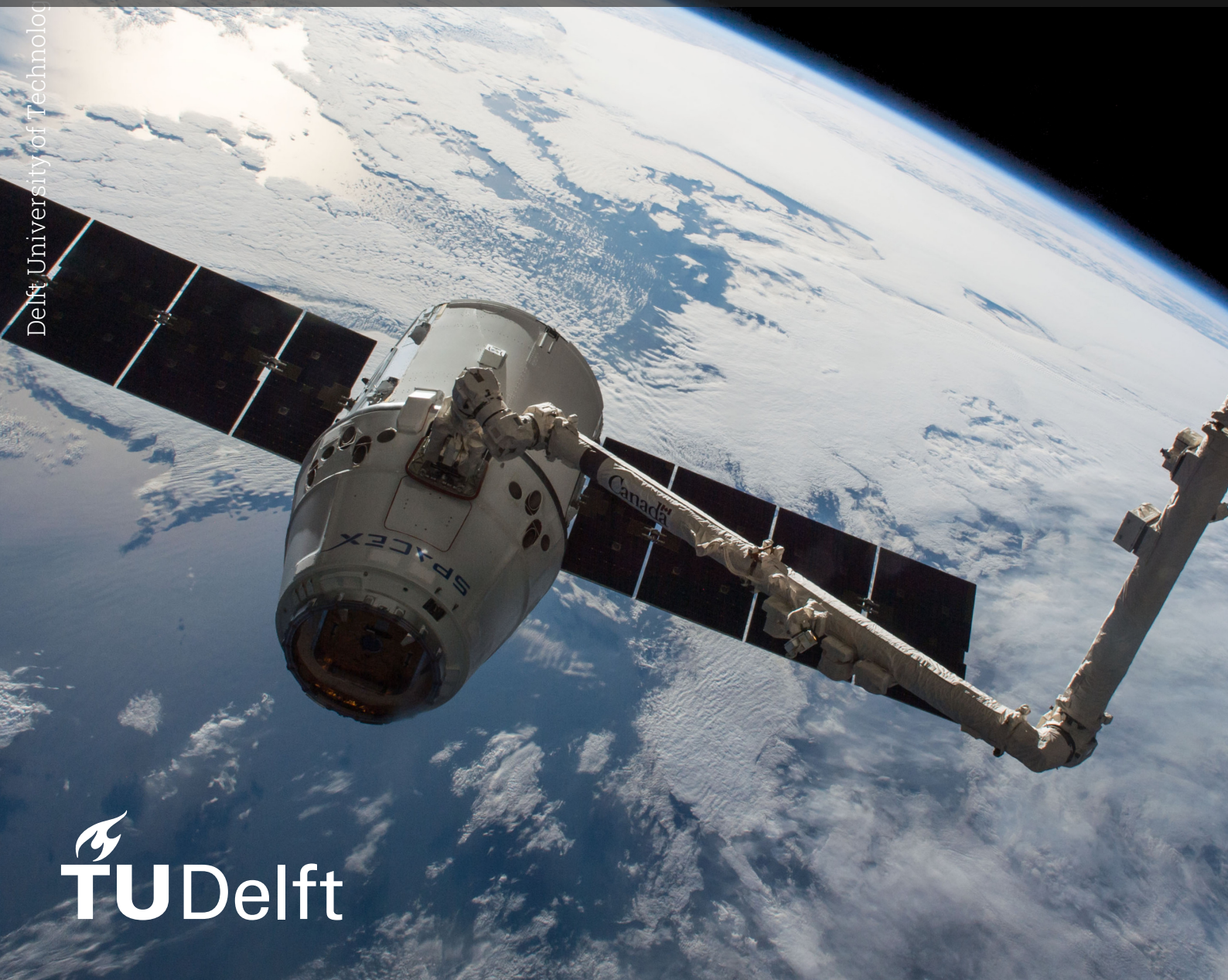
Sequential Feature Selection for Neural Ranking Models in Learning-to-Rank

Master Thesis

Alex Xingjian He

Delft University of Technology



TU Delft

Sequential Feature Selection for Neural Ranking Models in Learning-to-Rank

by

Alex Xingjian He

5773911

Thesis Supervisor:	A.Anand
Daily Supervisor:	L.Lyu
Project Duration:	Mar, 2024 - Apr, 2024
Faculty:	Faculty of Computer Science, Delft

TU Delft

I would like to express my gratitude to my thesis advisor, Avishek Anand, for his unwavering support, patience, and understanding throughout the course of my thesis. Even during the difficult times when I doubted myself and felt overwhelmed, his encouragement helped me get back on track.

I am also grateful to Lijun Lyu for her insightful ideas and inspirational discussions that laid the foundation for this thesis. Her input gave me the direction and motivation I needed to begin and I am truly thankful.

I would like to thank the members of the reading group for organizing weekly sessions and fostering an engaging academic environment. Although I did not attend most of the meetings, knowing that such a community existed was a quiet source of reassurance and motivation.

Lastly, I want to acknowledge myself for persevering and embracing the challenges, and for continuing to push forward despite the difficulties. This journey has not been easy, but I am proud of the growth it has brought me.

Abstract

Feature selection plays a crucial role in enhancing the efficiency, interpretability, and generalization of Learning-to-Rank (LTR) models. While recent advances in neural ranking have shown competitive performance, they often lack explicit mechanisms for selecting relevant input features, resulting in increased computational costs and reduced transparency. Moreover, existing feature selection methods frequently suffer problems from feature and label leakage, which undermines interpretability and inflates performance estimates.

In this thesis, we adapt and extend the Sequential Unmasking without Reversion (SUWR) method in the context of neural Learning-to-Rank. Our approach introduces a leakage-free, sequential feature selection method that iteratively reveals features through a selector network, while maintaining a separate neural ranker for relevance prediction. Additionally, we integrate NeuralNDCG, a differentiable approximation for standard ranking metrics, to directly optimize ranking performance during training.

We evaluate our method on three benchmark LTR datasets: MQ2008, Web10K, and Yahoo!. Experimental results demonstrate that our approach not only maintains ranking effectiveness with significantly fewer features but also provides consistent and interpretable feature selections across queries. This work provides a feasible solution to feature selection in neural LTR models, with potential benefits for interpretable and resource-efficient information retrieval systems.

Contents

Summary ii

1 Introduction 1
- 1.1 Context . . . 1
  - 1.1.1 Information Retrieval and Learning to Rank . . . 1
  - 1.1.2 Feature Selection in Machine Learning . . . 2
  - 1.1.3 Feature Selection in Learning-to-Rank . . . 2
- 1.2 Research Gaps and Motivation . . . 3
- 1.3 Research Questions . . . 4
- 1.4 Contributions . . . 5
- 1.5 Thesis Organization . . . 5

2 Background 7
- 2.1 Learning-to-Rank . . . 7
  - 2.1.1 Definition . . . 7
  - 2.1.2 Learning Objectives and Optimization . . . 8
  - 2.1.3 Ranking Metrics . . . 9
  - 2.1.4 Traditional Ranking Methods in Information Retrieval . . . 10
  - 2.1.5 Learning-to-Rank Models . . . 11
- 2.2 Feature Selection . . . 11

3 Related Works 13
- 3.1 Rankers in Learning to Rank . . . 13
  - 3.1.1 Gradient Boosted Decision Trees-based Rankers . . . 14
  - 3.1.2 MART . . . 15
  - 3.1.3 LambdaRank . . . 15
  - 3.1.4 LambdaMART . . . 16
  - 3.1.5 Neural Rankers . . . 16
  - 3.1.6 Deep Neural Networks for Learning-to-Rank . . . 16
  - 3.1.7 Transformer-Based Rankers . . . 17
  - 3.1.8 Motivation for Feature Selection in Neural Rankers . . . 17
- 3.2 Feature Selection Methods for LTR . . . 18
  - 3.2.1 Filter Methods . . . 18
  - 3.2.2 Wrapper Methods . . . 18
  - 3.2.3 Embedded Methods . . . 19
- 3.3 Feature Selection Methods for Nerual LTR . . . 19
  - 3.3.1 Sampling-Based Feature Selection Methods . . . 19
  - 3.3.2 Regularization-Based Feature Selection Methods . . . 20
  - 3.3.3 Feature and Label Leakage in Local Feature Selection . . . 20
  - 3.3.4 Challenges . . . 21
  - 3.3.5 Motivation for a Novel Feature Selection Approach . . . 21

3.4 Summary . . . 21

4 Methodology 22
4.1 Overview of Sequential Feature Selection Method . . . 22
4.2 Formal Definition of Sequential Unmasking . . . 23
4.2.1 Leakage Prevention . . . 24
4.3 Sequential Feature Selection for Neural Rankers . . . 25
4.3.1 Selector Network . . . 26
4.3.2 Ranker Network . . . 27
4.3.3 Integrating NDCG in training . . . 27
4.3.4 Implementation Details. . . . 29
4.3.5 Summary of Sequential Feature Selection for Neural Rankers . . . 29
4.4 Training Strategies . . . 29
4.4.1 Joint Training . . . 30
4.4.2 Two-Phase Training . . . 31
4.5 Summary of Methodology . . . 31

5 Experiment and Evaluation 32
5.1 Experiment Overview . . . 32
5.2 Datasets . . . 33
5.2.1 MQ2008 . . . 33
5.2.2 Web10k . . . 33
5.2.3 Yahoo! . . . 33
5.3 Experiment Setup . . . 34
5.4 Experiment Results . . . 35
5.4.1 RQ1: Ranking Performance of SUWR Compared to Full-Feature Baselines . . . 35
5.4.2 RQ2: Comparing SUWR with Existing Feature Selection Methods in Neural Rankers . . . 36
5.4.3 RQ3: Impact of Different Loss Functions on SUWR Performance 40
5.4.4 RQ4: Consistency and Interpretability of Selected Features . . . 42
5.5 Summary . . . 44

6 Conclusion 47
6.1 Summary of the Thesis . . . 47
6.2 Limitations . . . 48
6.3 Future Directions . . . 48

References 50

1

Introduction

1.1. Context

The rapid expansion of the internet has led to an unprecedented growth in data, making efficient and effective information retrieval systems essential. As of 2024, there are over 1.2 billion websites, each hosting vast amounts of digital information. The sheer volume of available content makes traditional filtering impractical, requiring the development of more sophisticated search engines to retrieve and rank relevant information efficiently. The increasing dependence on search engines underscores the importance of ranking methodologies that optimize document retrieval and enhance user experience.

1.1.1. Information Retrieval and Learning to Rank

Information retrieval (IR) is a fundamental area of study concerned with identifying and retrieving relevant documents from large, unstructured datasets [24]. The core objective of IR systems is to deliver the most relevant information in response to a user's query. Many IR tasks inherently involve ranking, including document retrieval, collaborative filtering, key term extraction, sentiment analysis, product recommendation, and spam detection. This thesis primarily focuses on document retrieval, where ranking plays a crucial role in determining the relevance of retrieved documents.

Learning to Rank (LTR) is a subfield of information retrieval that focuses on optimizing the ranking order of retrieved items within IR systems. Unlike traditional ranking techniques based on heuristic rules, LTR models learn ranking functions from labeled training data. These models can adapt to different ranking contexts, handle high-dimensional feature spaces, and exploit complex relationships between features. The primary advantage of LTR methods is their ability to improve retrieval accuracy by leveraging large-scale data and optimizing for task-specific ranking objectives.

Traditional IR systems have long relied on heuristic models such as TF-IDF and BM25, which apply predefined formulas based on term frequency and inverse document frequency. While effective in many cases, these models do not fully exploit the vast and

diverse datasets available today. Advances in machine learning and deep learning have facilitated the development of more sophisticated ranking models. Specifically, learning-to-rank (LTR) methods [22] leverage machine learning techniques to optimize document ranking by automatically identifying complex relationships between features and relevance scores. Unlike heuristic-based approaches, LTR models can learn from large-scale data and adapt to different ranking contexts, leading to improved retrieval performance and more relevant search results.

A defining characteristic of LTR methods is their focus on predicting the optimal ranking order of documents rather than assigning absolute relevance scores [23]. Traditional LTR models often utilize gradient-boosted decision trees (GBDTs) such as LambdaMART [3]. These models have demonstrated strong performance due to their ability to model complex feature interactions while maintaining robustness against overfitting.

More recently, neural ranking models have emerged as a promising alternative. These models leverage deep learning architectures to capture intricate feature representations and directly learn ranking functions. Neural rankers can model complex, non-linear relationships within data, which traditional models may fail to capture. Techniques such as attention mechanisms have been introduced to enhance ranking performance further. Recent studies indicate that with appropriate feature transformations and data augmentations, neural ranking models can achieve performance on par with, or even surpass, GBDT-based methods [29]. While GBDT-based models like LambdaMART remain strong contenders due to their efficiency and interpretability, neural models have shown competitive performance when trained with large-scale data and optimized architectures. The trade-offs between interpretability, computational efficiency, and ranking effectiveness remain an active area of research in learning-to-rank methodologies.

1.1.2. Feature Selection in Machine Learning

Feature selection is a critical process in machine learning that involves identifying and retaining the most relevant features while discarding redundant or irrelevant ones. This approach improves model interpretability, reduces computational complexity, and mitigates overfitting [15].

While feature selection methods have proven effective in tasks such as classification and regression, adapting them to more complex problems like learning-to-rank poses additional challenges. In ranking scenarios, the relative importance of features is crucial for ordering items correctly, and traditional feature selection techniques may not fully capture the intricate dependencies required for optimal ranking performance. This realization motivates the exploration of feature selection strategies specifically tailored for learning-to-rank models, which is the focus of the next section.

1.1.3. Feature Selection in Learning-to-Rank

Feature selection plays an essentialial role in enhancing the interpretability and efficiency of learning-to-rank models. As highlighted in [23], while interpretable machine

learning techniques have advanced significantly in various domains, their effectiveness for feature selection in learning-to-rank remains an open challenge. In traditional machine learning tasks, the objective of feature selection is to identify and retain a subset of relevant features that can improve model performance while reducing computational complexity. However, when applied to learning-to-rank, the task becomes more intricate because ranking models must determine the optimal ordering of documents rather than simply predicting discrete labels or continuous outputs.

Neural ranking models must capture complex, nonlinear relationships between features. However, to enhance efficiency and interpretability, the feature selection process must limit the input to only the most relevant features, ensuring that no additional information from unselected features is inadvertently encoded. Such unintended encoding can lead to feature or label leakage [25], undermining the model's interpretability. Moreover, many existing feature selection methods suffer from this leakage problem, as they sometimes incorporate extra information that should not be present in the selected features. This challenge is compounded by the high degree of feature redundancy found in LETOR datasets, making it difficult to isolate only those features that truly belong in the final selection.

Existing feature selection techniques for neural ranking models include L2X [9], INVASE [33], and TabNet [2]. These methods have indeed been applied to neural rankers with the aim of enhancing interpretability by identifying and retaining the most relevant features while preserving ranking performance. However, it is important to note that these techniques were originally developed for classification and regression tasks. As a result, although they are used in neural ranking contexts, their direct applicability to Learning-to-Rank scenarios is not automatically guaranteed.

In Learning-to-Rank, the objective is not to predict absolute labels but to determine the optimal ordering of documents, a task that demands capturing subtle relative differences in feature importance and interactions among features. This means that while methods like L2X, INVASE, and TabNet provide useful starting points, they may require adaptation or further refinement to fully address the unique challenges of ranking tasks. Recent research has therefore focused on developing ranking-specific feature selection techniques, such as G-L2X, which employs a global selector layer to produce a single, shared probability distribution that is applied uniformly across all items, ensuring consistency in feature selection. However, there is still a critical gap in current methods and motivates our research to explore and develop a feature selection approach that explicitly mitigates feature leakage while enhancing interpretability and efficiency in neural LTR models.

1.2. Research Gaps and Motivation

A critical limitation of current feature selection methods is their susceptibility to both feature leakage and label leakage [25]. Feature leakage occurs when the feature selection mask encodes information about non-selected features, resulting in an unfaithful representation of those features that genuinely drive the model's predictions. Similarly, label leakage arises when the selected features unintentionally reveal infor-

mation about labels, thereby influencing the feature selection process. These issues are particularly detrimental in neural LTR models, where leakage can create hidden dependencies between selected and non-selected features or labels, ultimately leading to misleading interpretations that compromise model interpretability.

Prior work, such as the Sequential Unmasking Without Recursion (SUWR) method presented in [25], has made significant progress in mitigating both feature and label leakage in general supervised learning tasks. However, its application to Learning-to-Rank remains largely unexplored. The unique challenges of ranking, where the goal is to predict the relative ordering of documents rather than absolute labels, demand more tailored adaptations.

To address this problem, this thesis aims to adapt the SUWR strategy from [25] to the context of neural LTR models and introduces modifications by training a dedicated selector and ranker network. The sequential feature selection method effectively limits the input to only the most relevant features, ensuring that no extra information from unselected features or labels is encoded, while preserving the critical features necessary for accurate ranking. Additionally, we incorporate NeuralNDCG, a differentiable approximation of the standard NDCG metric, into the training objective to better align the optimization process with the ranking goals of LTR tasks. In doing so, our approach should both interpretability and computational efficiency while maintaining a reasonable level of ranking performance. We then conduct experiments across standard LTR datasets to demonstrate the effectiveness of the proposed method in selecting compact subsets of informative features without sacrificing ranking quality. In addition, we evaluate the consistency of the selected features across queries and compare the results with existing feature selection methods to assess interpretability and stability.

1.3. Research Questions

To evaluate the effectiveness of SUWR in Learning-to-Rank contexts, we formulate the following research questions:

RQ1: *Can SUWR achieve comparable ranking performance while selecting only a subset of features, compared to baseline neural rankers that use the full feature set?*
This question investigates whether the sequential feature selection method can identify informative features that effectively contribute to generating accurate ranking results. By comparing SUWR to a baseline neural ranker that uses the full feature set, we evaluate whether selecting a reduced subset of features can preserve performance. This comparison helps determine the effectiveness of the selection process and whether it meaningfully contributes to model efficiency without sacrificing ranking quality.

RQ2: *How does SUWR compare with existing feature selection methods in terms of ranking accuracy and feature sparsity?*
This question evaluates SUWR with other feature selection methods by constraining all approaches to select the same number of features and comparing their resulting ranking performance. The objective is to determine which method

is more effective at identifying relevant features that lead to higher ranking quality. In addition, we investigate the trade-off between ranking accuracy and feature sparsity. Specifically, how reducing the proportion of selected features impacts performance across different selection methods, and whether a smaller subset can still produce satisfactory ranking results.

RQ3: *How do different loss functions affect SUWR's performance in feature selection and ranking?*
In Learning-to-Rank scenarios, training objectives typically fall into three categories: pointwise, pairwise, and listwise loss functions. This question investigates how the choice of LTR-specific loss function influences SUWR's ability to select more informative features in order to achieve higher ranking accuracy. By comparing performance under different loss formulations, we aim to understand the trade-offs involved and determine which loss functions are most effective for guiding the sequential feature selection process in neural rankers.

RQ4: *Are the features selected by SUWR consistent across queries, thereby enhancing model interpretability?*
This question explores whether SUWR consistently selects the same or similar features across different queries, which would indicate that certain features are generally more important for effective ranking. By analyzing the frequency of selected features, we aim to gain insights into which features the neural ranker relies on most. Such consistency enhances the interpretability of the model by revealing patterns in feature usage, helping users understand which features drive the ranking decisions.

1.4. Contributions

This thesis addresses the identified research gaps by making the following contributions:

- Adapting the feature selection strategy from [25] to neural LTR models through modifications that involve training a dedicated selector and ranker network.
- Conducting a comprehensive evaluation of the adapted feature selection approach against established techniques, demonstrating its impact on ranking effectiveness and computational efficiency in LTR tasks.
- Providing insights into the trade-offs between feature sparsity, model interpretability, and computational cost, thereby contributing to a broader understanding of effective feature selection in neural LTR models.

1.5. Thesis Organization

The thesis is organized as follows: Chapter 2 provides background on Learning to Rank and feature selection techniques. Chapter 3 reviews related work on feature selection in LTR and neural ranking models. Chapter 4 introduces the proposed feature selection method, along with details of the experimental setup. Chapter 5 presents

the experimental results and analysis. Finally, Chapter 6 concludes the thesis and discusses potential future research directions.

The implementation codes for this thesis is available at [1] this GitHub repository.

[1] https://github.com/DizzyMizLizzy/thesis_feature_selection

2

Background

This chapter provides an overview of the fundamental concepts that serve as the foundation for this research. The discussion begins with Learning-to-Rank, introducing its significance and different modeling approaches. Traditional ranking models, including gradient-boosted decision trees, are explored, followed by an examination of neural ranking models and their characteristics. The latter sections focus on feature selection, emphasizing its role in improving efficiency and interpretability in ranking models. The challenges associated with feature selection in Learning-to-Rank tasks and existing methodologies are also discussed, establishing the foundation for the research contributions of this thesis.

2.1. Learning-to-Rank

2.1.1. Definition

Learning-to-Rank(LTR) is a supervised machine learning approach that constructs ranking functions to order a list of items based on their relevance to a query or task-specific objective [22]. The objective is to optimize ranking quality by learning an optimal sorting function from labeled data. LTR models typically rely on a dataset of query-document pairs $\Psi = \{(x, y) \in \chi^n \times \mathbb{R}^n\}$, where:

- x represents a list of n items, each described by a feature vector $x_i \in \mathbb{R}^k$.
- y denotes the associated relevance labels, where $y_i \in \mathbb{R}$ represents the degree of relevance of x_i to the query.
- The ranking function $s(x)$ predicts an ordering of items, optimizing ranking metrics such as Normalized Discounted Cumulative Gain (NDCG) or Mean Average Precision (MAP).

2.1.2. Learning Objectives and Optimization

From the formulation above, it is evident that the ranking function plays a central role in the effectiveness of Learning-to-Rank models. Since the core objective of LTR is to produce an optimal ordering of items, optimizing the ranking function becomes a fundamental challenge. An ideal ranking function should not only correctly differentiate between relevant and irrelevant documents but also generalize well across diverse datasets and retrieval tasks. Therefore, research in LTR focuses extensively on learning robust ranking functions that enhance retrieval performance while minimizing computational complexity and overfitting risks.

A ranking function is a mathematical model that assigns a score to each item in a set based on its relevance to a query. The primary objective of Learning-to-Rank methods is to learn an effective ranking function that optimally sorts items according to their importance. The choice of ranking function varies across different models; linear models such as logistic regression, tree-based models like LambdaMART, and neural ranking models such as transformers each define their ranking function differently.

The training process of an LTR model is guided by a loss function, which measures how well the ranking function's predictions align with the ground truth. The loss function plays a critical role in optimizing the ranking function by minimizing discrepancies between predicted and ideal rankings. Unlike traditional supervised learning, where loss functions minimize classification or regression errors, LTR loss functions focus on optimizing the ranking order. The loss functions used in LTR models can be categorized as follows:

- Pointwise Loss Functions: Treat each query-document pair independently, optimizing mean squared error (L2 loss) or logistic regression-based loss [21]. One commonly used loss is the L1 loss, which minimizes the absolute difference between predicted and actual relevance scores and is defined as:

$$L_{L1} = \sum_{i} |y_i - s(x_i)|$$

  where y_i is the ground truth relevance and $s(x_i)$ is the predicted relevance score.

- Pairwise Loss Functions: Optimize ranking quality by reducing the number of incorrectly ranked pairs, commonly using hinge loss or logistic loss [5]. A well-known pairwise loss function is the hinge loss, which is used to enforce a margin between correctly and incorrectly ordered pairs:

$$L_{hinge} = \sum_{(i,j)\in P} \max(0, 1 - (s(x_i) - s(x_j)))$$

  where (i, j) represents a pair where x_i should be ranked higher than x_j, and $s(x)$ is the ranking function.

- Listwise Loss Functions: Directly optimize ranking metrics such as NDCG through methods like softmax cross-entropy or surrogate ranking losses [6]. A commonly

used loss function in this category is the softmax cross-entropy loss, which encourages higher relevance scores for documents ranked at the top:

$$L_{softmax} = -\sum_{i} P_i \log Q_i$$

where P_i is the ground truth probability distribution over ranked documents and Q_i is the predicted probability distribution obtained through softmax normalization of scores.

The effectiveness of LTR models depends not only on the optimization of ranking functions but also on how ranking quality is assessed. Evaluating a ranking model requires appropriate ranking metrics that measure the alignment between predicted and ideal rankings. These metrics guide model selection and optimization, ensuring that ranking functions produce results that align with user expectations. The next section introduces key ranking metrics commonly used in Learning-to-Rank research.

2.1.3. Ranking Metrics

LTR models are trained to optimize specific ranking evaluation measures that quantify the effectiveness of ranked lists. One of the most widely used metrics is Normalized Discounted Cumulative Gain (NDCG), which accounts for both the position and the relevance of retrieved documents. It is defined as:

$$\text{NDCG@}k = \frac{DCG@k}{IDCG@k},$$

where Discounted Cumulative Gain (DCG) is computed as:

$$DCG@k = \sum_{i=1}^{k} \frac{2^{y_{\pi(i)}} - 1}{\log_2(\pi(i) + 1)}.$$

In this formulation:

- $\pi(i)$ represents the position of the document ranked at index i in the predicted ranking order.
- $y_{\pi(i)}$ denotes the relevance score of the document at position $\pi(i)$, indicating how relevant the document is to the query.

The Ideal DCG (IDCG) is the DCG computed using the optimal ranking order of the documents based on their true relevance scores. NDCG is commonly used because it captures two key aspects of ranking quality: relevance and position. Higher relevance scores contribute more to the ranking, ensuring that the most relevant documents receive greater weight. Additionally, the logarithmic discounting factor penalizes lower-ranked documents, reflecting the intuition that users are more likely to focus on top-ranked results. By normalizing DCG with IDCG, NDCG ensures comparability across different queries, making it a robust metric for ranking evaluation.

Ranking metrics serve as critical benchmarks for assessing the performance of Learning-to-Rank models. The choice of metric depends on the specific application and ranking objective. For instance, web search engines prioritize metrics like NDCG, whereas recommendation systems may focus on MAP or MRR. Given the diversity of ranking tasks, selecting an appropriate evaluation metric is essential for ensuring that models optimize for relevant ranking objectives.

The effectiveness of a ranking model is largely determined by its ability to optimize these metrics. To achieve this, various Learning-to-Rank models have been developed, ranging from traditional tree-based methods to modern neural ranking architectures. The following section explores these approaches in detail, discussing their fundamental principles, strengths, and limitations in different ranking scenarios.

2.1.4. Traditional Ranking Methods in Information Retrieval

Before the emergence of supervised learning-based ranking methods, information retrieval systems largely used traditional ranking models. These approaches often rely on well-established theoretical frameworks such as the probabilistic model to score and rank documents. For instance, term frequency, inverse document frequency, and document length are some of the core statistical properties used in models like BM25.

These methods are typically not considered LTR because they do not train a ranking function directly from labeled data in a supervised manner. Instead, their scoring formulas are usually defined by a closed-form equation or probabilistic framework, with limited adaptation to new domains or specific ranking tasks. Two widely recognized classical approaches are:

- Term Frequency-Inverse Document Frequency (TF-IDF): Measures the importance of a term within a document relative to its occurrence in the entire corpus. It is defined as:
$$TF - IDF(t, d) = TF(t, d) \cdot IDF(t),$$
where $TF(t, d)$ represents the frequency of term t in document d, and $IDF(t)$ is the inverse document frequency, computed as:
$$IDF(t) = \log\left(\frac{N}{DF(t) + 1}\right) + 1.$$
Here, N is the total number of documents, and $DF(t)$ is the number of documents containing term t. The smoothing term $+1$ in the denominator ensures that terms appearing in all documents do not receive an IDF score of zero.
- BM25: A probabilistic ranking function that extends TF-IDF by incorporating term saturation and document length normalization. It is given by:
$$BM25(d, q) = \sum_{t \in q} IDF_{BM25}(t) \cdot \frac{TF(t, d) \cdot (k_1 + 1)}{TF(t, d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\text{avgdl}}\right)},$$
where:

- k_1 and b are tunable hyperparameters.
- $|d|$ is the length of document d.
- avgdl is the average document length in the corpus.
- The inverse document frequency in BM25 is calculated as:

$$IDF_{BM25}(t) = \log\left(\frac{N - DF(t) + 0.5}{DF(t) + 0.5} + 1\right).$$

The BM25 formulation accounts for the diminishing returns of term frequency, preventing excessively frequent terms from dominating the ranking while also adjusting for varying document lengths.

TF-IDF and BM25 are widely used in search engines due to their efficiency and simplicity. However, they serve different purposes: TF-IDF is effective in basic keyword matching, while BM25 enhances ranking by considering term saturation and document length normalization, making it more adaptive to varying query-document relationships. Despite their success, these methods rely on manually defined weighting functions and do not leverage large-scale relevance feedback, limiting their ability to model complex feature interactions and optimize ranking dynamically. As a result, more sophisticated ranking approaches have been developed, leading to the rise of machine-learning-based LTR models. These models leverage labeled data to learn optimal ranking functions that better generalize across different queries and documents.

2.1.5. Learning-to-Rank Models

The core objective of Learning-to-Rank models is to learn an optimal ranking function that orders documents based on their relevance to a given query. LTR approaches differ from traditional methods in that they use supervised learning techniques to optimize ranking functions. Among these models, Gradient-Boosted Decision Trees-based models, such as LambdaMART, have been widely adopted due to their robustness and interpretability in handling structured data, while Neural Network-based models excel at capturing complex non-linear interactions from high-dimensional data, making them well-suited for intricate ranking tasks despite often being less transparent. A more detailed discussion on recent advancements and challenges in LTR models is provided in the related works section.

2.2. Feature Selection

Feature selection is a crucial step in many machine learning tasks, aimed at identifying the most informative subset of input features while discarding those that are irrelevant or redundant. By reducing the dimensionality of the input space, feature selection can improve model generalization, reduce training and inference costs, and enhance interpretability. This is particularly important in applications where understanding the influence of features on a model's predictions is essential, such as in decision-critical or user-facing systems.

In the context of Learning-to-Rank, feature selection presents unique challenges. Unlike classification or regression tasks that predict absolute labels, LTR models must capture relative orderings among items, which often depend on relationships between features. As a result, feature selection methods for LTR must preserve not just individual feature relevance, but also feature dependencies that influence ranking quality.

Feature selection methods can be categorized into three main classes in the context of machine learning: filter, wrapper, and embedded methods [7]. Filter methods evaluate features based on statistical properties independent of any learning algorithm, offering computational efficiency. Wrapper methods assess subsets of features using a predictive model's performance, typically yielding better results but at higher computational cost. Embedded methods incorporate feature selection within the model training process itself, striking a balance between performance. A more detailed discussion of existing feature selection approaches can be found in the related work section.

3

Related Works

This chapter provides a comprehensive review of the literature on ranking methods in Learning-to-Rank and feature selection techniques. The discussion begins with an analysis of widely used ranking models in LTR, with a focus on Gradient Boosted Decision Trees-based rankers and neural rankers. Following this, various feature selection methodologies and their applications in machine learning are examined, where their potential to enhance model performance, reduce complexity, and improve interpretability in the context of LTR remains an area of ongoing investigations.

3.1. Rankers in Learning to Rank

Learning-to-Rank is a fundamental task in Information Retrieval, where the objective is to learn a model that can predict the optimal ordering of items based on their relevance to a query. Over the years, various ranking models have been developed, ranging from traditional approaches to more advanced neural network-based methods. Among the various approaches to LTR, the most prominent rankers are Gradient Boosted Decision Trees-based rankers, particularly LambdaMART. GBDT models are widely used due to their strong generalization ability across different ranking tasks, their efficiency, and their inherent interpretability through decision tree structures. LambdaMART, in particular, optimizes ranking-specific metrics like NDCG, making it one of the most widely adopted ranking algorithms. The effectiveness of GBDT models in ranking tasks can be attributed to their ability to model complex feature interactions while maintaining interpretability, as well as their robustness in learning from structured tabular data, which is common in ranking scenarios.

Recent research has explored the potential of neural rankers, leveraging deep learning architectures to model intricate ranking patterns. Unlike GBDT models, neural rankers can directly learn feature representations from raw data. They have demonstrated strong performance in capturing complex query-document relationships, particularly in text-based ranking tasks. However, neural rankers often require significantly larger amounts of labeled training data and high computational resources. Furthermore, their black-box nature presents challenges in interpretability, making it difficult

to understand how ranking decisions are made.

While our study primarily focuses on GBDT-based and neural rankers, several other ranking models have been influential in LTR, which have laid the groundwork for modern retrieval systems. Some of the most influential rankers include:

- RankSVM: RankSVM [18] is a pairwise ranking model that leverages support vector machines (SVMs) to learn a ranking function. By optimizing a hinge loss over pairs of documents, RankSVM captures relative ordering preferences effectively, making it a popular choice for various ranking applications.
- ListNet: ListNet [6] is a listwise LTR approach that directly optimizes a permutation-based loss function over entire ranking lists. Unlike pairwise methods, ListNet considers the whole order of documents, allowing it to capture complex dependencies within the list, which can lead to improved ranking performance.
- AdaRank: AdaRank [32] is a boosting-based ranking algorithm that iteratively adjusts the weights of training instances according to their ranking errors. By combining multiple weak rankers into a strong ensemble, AdaRank is capable of directly optimizing ranking metrics, making it particularly effective for information retrieval tasks.

The following sections provide an in-depth examination of both GBDT-based rankers and neural ranking models. While GBDT-based models have been widely used in LTR due to their strong performance, recent advancements in deep learning have introduced neural rankers as a competitive alternative. Their ability to learn feature representations directly from raw data makes them particularly suitable for complex ranking tasks. However, feature selection remains a crucial aspect of LTR, and GBDT-based models offer valuable insights into this process through their intrinsic ability to rank feature importance. This study explores these insights and investigates how they can be leveraged to enhance the feature selection process for neural rankers in Learning to Rank.

3.1.1. Gradient Boosted Decision Trees-based Rankers

Gradient Boosted Decision Trees have been widely adopted in LTR tasks due to their strong performance and ability to handle complex feature interactions [10]. GBDT-based rankers are ensemble models that combine the predictions of multiple decision trees, built sequentially to correct the errors of preceding trees. These models iteratively refine the decision boundary, allowing them to model non-linear relationships between features and ranking outcomes.

Among these models, LambdaMART [3] stands out as one of the most widely used ranking models. LambdaMART builds upon two key predecessors: Multiple Additive Regression Trees (MART) [10], which applies boosting techniques to decision trees, and LambdaRank [4], which introduces ranking-specific gradient adjustments to optimize information retrieval metrics. By integrating the strengths of both models, LambdaMART refines ranking functions with enhanced effectiveness. In the following sections, we present a detailed introduction of MART, LambdaRank, and their

combination in LambdaMART, highlighting their progression and contributions to LTR models.

3.1.2. MART

MART is the foundation of many GBDT-based ranking models. It is a variant of GBDT that sequentially builds regression trees to minimize loss functions, making it highly effective for ranking tasks. MART trains an ensemble of decision trees sequentially, where each new tree corrects the errors of the previous ensemble. It optimizes for squared error loss by iteratively fitting decision trees to the residuals of previous trees. The model updates follow:

$$F_m(x) = F_{m-1}(x) + \eta \sum_{t=1}^{T} w_t h_t(x),$$

where $F_m(x)$ is the ensemble model at iteration m, η is the learning rate, w_t is the weight of the t-th tree, $h_t(x)$ represents the individual regression trees.

MART's strength lies in its ability to capture non-linear relationships through decision trees, making it particularly useful for structured data. However, its primary limitation in ranking tasks is its use of squared error loss, which does not directly optimize ranking metrics, making it less suited for LTR tasks.

3.1.3. LambdaRank

LambdaRank is an extension of RankNet [5], introducing lambda gradients to optimize ranking-specific objectives more effectively. Unlike traditional gradient boosting, LambdaRank modifies gradient updates to prioritize changes that improve ranking order based on a chosen ranking metric, such as NDCG.

LambdaRank modifies the gradient of RankNet by incorporating a weighting factor, *lambda*, which reflects the change in ranking quality due to a swap of two documents i and j. The gradient is adjusted as:

$$\lambda_{ij} = |\Delta NDCG| \cdot \sigma \cdot (1 - P_{ij})$$

where:

- $\Delta NDCG$ is the change in *NDCG* if documents i and j swap positions.
- P_{ij} is the predicted probability from RankNet.
- σ is a scaling factor.

This adjustment ensures that errors that have a higher impact on ranking metrics, receive stronger gradient updates, aligning the training process more closely with ranking performance. This approach makes LambdaRank metric-aware, differentiating it

from RankNet by improving the correlation between model training and ranking effectiveness.

3.1.4. LambdaMART

LambdaMART [3] combines the boosting mechanism of MART with the ranking optimization of LambdaRank. It refines traditional boosting approaches by employing lambda gradients, ensuring that ranking models directly optimize for ranking-specific measures, and dynamically adjusts gradient updates to maximize ranking performance.

By building decision trees and adjusting their predictions using lambda gradients iteratively, LambdaMART efficiently refines ranking scores in a way that aligns with ranking objectives. This iterative refinement allows the model to adapt dynamically, making it one of the most widely used LTR algorithms. LambdaMART's ability to capture feature interactions while directly optimizing ranking metrics contributes to its strong performance across various benchmark datasets, such as the Yahoo! Learning to Rank Challenge [8] and Microsoft's LETOR datasets [28].

3.1.5. Neural Rankers

Neural LTR models have gained significant attention in recent years due to their ability to model complex interactions between ranking features. Unlike GBDT-based rankers, which rely on structured feature engineering, neural rankers can learn feature representations directly from raw data. However, recent studies [29] have shown that GBDT-based models, particularly LambdaMART, still outperform neural rankers on many benchmark datasets. To address these shortcomings, [29] has focused on improving neural rankers through enhanced feature representations, data augmentation, and feature selection and regularization. Despite these improvements, neural rankers still exhibit challenges in training stability, computational efficiency, and interpretability. In the following sections, we introduce two of the most widely adopted neural ranking architectures: Deep Neural Network-based rankers and Transformer-based rankers.

3.1.6. Deep Neural Networks for Learning-to-Rank

Deep Neural Networks (DNNs) were among the first neural architectures explored for Learning-to-Rank tasks [16, 14]. These models utilize fully connected layers to learn non-linear transformations of input features, enabling them to model complex ranking relationships.

A typical DNN-based ranking function is formulated as:

$$s(x) = f(W_L \cdot \sigma(W_{L-1} \cdot \ldots \sigma(W_1 x + b_1) + b_{L-1}) + b_L)$$

where:

- x represents the input feature vector,

- W_l and b_l are the weights and biases for the l-th layer,
- σ is a non-linear activation function (e.g., ReLU),
- $s(x)$ is the final ranking score.

DNN-based rankers operate by mapping input feature vectors to ranking scores, which are then used to order query-document pairs. Compared to traditional ranking models, DNNs offer greater flexibility in learning complex patterns but often require substantial training data and regularization techniques to prevent overfitting [11].

3.1.7. Transformer-Based Rankers

Transformers [31] have revolutionized neural ranking by introducing self-attention mechanisms, which enable models to capture both local and global dependencies within ranking inputs. Unlike DNNs, which rely on fixed-length feature vectors, Transformer-based models process sequential data in a context-aware manner, making them particularly effective for text-based ranking tasks.

A typical self-attention operation in Transformer-based ranking models is given by:

$$s(x) = W \cdot \text{softmax}(QK^T/\sqrt{d})V$$

where:

- Q, K, V are the query, key, and value matrices, computed from input embeddings,
- d is the dimensionality of the embeddings,
- W is a learnable weight matrix.

By leveraging pre-trained language models, Transformer-based rankers can encode rich contextual representations to improve ranking accuracy. However, they also require substantial computational resources and large-scale pretraining to achieve competitive performance. Additionally, their black-box nature poses challenges in interpretability, making them less practical for ranking applications where feature importance analysis is crucial.

3.1.8. Motivation for Feature Selection in Neural Rankers

This thesis focuses on neural rankers due to their ability to automatically learn feature representations and adapt to diverse ranking tasks. Despite their increasing adoption, one major challenge remains: neural ranking models often lack explicit feature selection mechanisms, making their decision-making process less interpretable and potentially inefficient. While deep learning models excel at capturing complex, high-dimensional feature interactions, they do not inherently prioritize the most relevant features, which can lead to redundancy, increased computational costs, and reduced interpretability.

In contrast, GBDT-based models, particularly LambdaMART, have long been considered the state-of-the-art in Learning-to-Rank due to their structured decision-making and inherent feature selection capabilities. The success of GBDT models stems from their ability to selectively split on the most informative features, effectively reducing noise and improving model efficiency. This structured decision process offers valuable insights into how feature selection can enhance ranking models.

This thesis investigates how feature selection can be incorporated into neural ranking models to achieve better performance and interpretability. By drawing inspiration from the structured decision processes of GBDTs, we aim to develop novel feature selection strategies that enable neural rankers to focus on the most informative features, reduce computational overhead, and produce more explainable ranking decisions.

3.2. Feature Selection Methods for LTR

Feature selection methods for LTR can be broadly classified into three categories: filter methods, wrapper methods, and embedded methods.

3.2.1. Filter Methods

Filter methods assess each feature independently from any LTR model training, typically relying on statistical methods that estimate how well a single feature (or a simple combination of features) correlates with a relevance signal. Unlike wrapper or embedded techniques, these methods do not require iterative training of a ranking model, making them computationally efficient and easy to implement.

Examples of filter-based approaches in LTR often adapt established metrics from classification or regression tasks. Geng et al. [12] discuss how correlation-based filters, which quantify the degree to which a feature's values align with relevance scores across documents, can be effective baselines in ranking settings. Mutual Information has also been explored as a way to measure the reduction of uncertainty in relevance when a feature is observed; higher mutual information typically indicates a more informative feature. Similar logic underpins the use of Chi-square tests, which evaluate the dependence between feature values and relevance labels to identify statistically salient dimensions. Although these metrics are straightforward to compute, they may overlook complex interactions among features.

3.2.2. Wrapper Methods

Wrapper methods perform feature selection by iteratively training and evaluating a ranking model, using performance metrics such as NDCG or MAP to guide the search for informative features. This process can be computationally expensive, as each feature subset under consideration must be assessed with a full training cycle of the LTR model. Geng et al. [12] illustrate how wrapper methods can adapt traditional feature selection strategies to ranking by directly optimizing ranking-based objectives rather than classification accuracy or regression error. This direct coupling often yields

more reliable insights into which features genuinely enhance a model's ability to order documents by relevance.

However, the iterative nature of wrapper methods presents scalability challenges when the feature space is large or training data is extensive. Gigli et al. [13] address these concerns by introducing accelerated strategies, such as parallelization and approximate ranking objectives, that reduce the number of full model evaluations needed. By doing so, they maintain much of the performance benefit inherent in wrapper approaches while mitigating the associated computational overhead. As a result, wrapper methods remain an attractive option in scenarios where model accuracy and fine-grained feature evaluation outweigh runtime constraints.

3.2.3. Embedded Methods

Embedded methods integrate feature selection into the training of the ranking model itself, using mechanisms such as regularization [30] or attention-based gating to quantify and enforce feature importance. In these methods, feature weights or masks are updated together with model parameters, allowing the selection process to be directly informed by the optimization of ranking metrics. One key advantage of embedded methods is their ability to capture intricate feature interactions without the need for repeated retraining on multiple subsets, as in wrapper approaches. They are also more flexible than filter methods in addressing the relative nature of relevance by tailoring regularization or gating criteria to ranking losses. However, the success of an embedded method can hinge on the quality of the inductive biases or architectural choices within the model, meaning that performance may vary significantly across different LTR datasets and tasks. Despite these caveats, embedded methods continue to gain popularity as they often strike a favorable balance between computational efficiency, interpretability, and ranking effectiveness.

3.3. Feature Selection Methods for Nerual LTR

Feature selection is an essential technique for improving model interpretability, reducing computational complexity, and enhancing generalization performance. While extensively studied in classification and regression tasks, its application in neural LTR remains an emerging area of research. Recent studies have evaluated whether interpretable machine learning methods can effectively perform feature selection in neural ranking models [23]. These methods can be broadly categorized into sampling-based and regularization-based approaches.

3.3.1. Sampling-Based Feature Selection Methods

Sampling-based methods aim to select a subset of informative features by training supplementary models to learn feature importance. These approaches typically rely on learning an explicit selection mechanism that identifies the most relevant features based on predictive performance. In sampling-based methods, L2X [9] employs an encoder-decoder to sample feature subsets that maximize predictive information but

remains unadapted for ranking losses; G-L2X [23] extends L2X by focusing on pairwise feature importance for ranking, yet does not mitigate feature leakage; INVASE [33] leverages a reinforcement learning framework for data-driven selection but lacks direct integration with ranking metrics; CAE [1] uses a differentiable concrete distribution for feature selection but is untested in LTR settings; IFG [20] adds a secondary model for importance estimation, which may limit scalability in ranking tasks.

3.3.2. Regularization-Based Feature Selection Methods

Regularization-based methods introduce sparsity constraints during model training, encouraging the selection of relevant features while penalizing redundant or irrelevant ones. These methods integrate feature selection directly into the optimization process, making them computationally efficient and well-suited for large-scale ranking tasks. Among regularization-based methods, TabNet [2] integrates sparse attention masks to dynamically select features but does not explicitly account for ranking dependencies, and LassoNet [19] applies hierarchical L1 constraints within deep networks, though it remains unexplored for ranking-specific objectives.

3.3.3. Feature and Label Leakage in Local Feature Selection

Local feature selection methods produce instance-specific masks, revealing only a subset of input features for each sample, whereas global selection methods employ a single, fixed mask for an entire dataset [25]. Although local approaches can offer more flexible and fine-grained explanations, they are prone to two forms of leakage that fundamentally undermine interpretability.

Label Leakage. According to Oosterhuis et al. [25], label leakage occurs when a feature selector's output encodes information about the label itself rather than strictly reflecting the relevant features. In other words, the act of selecting features should not alter the conditional distribution of the label given the selected features. However, if the selection mask allows a predictor to infer information about the true label that is not contained in the chosen features alone, the method exhibits label leakage. This situation defeats the intended purpose of local feature selection, because the resulting explanations no longer indicate which input features genuinely determine the prediction.

Feature Leakage. Oosterhuis et al. [25] similarly define feature leakage as the unintended encoding of non-selected (masked) feature values into the selection mask. Even though certain features appear "excluded" from the prediction, the selection mechanism can inadvertently embed information about them, allowing the model to effectively use hidden variables it is not supposed to access. This behavior compromises the interpretability of local selection, because it implies the chosen subset is not the sole information source used by the predictor.

As Oosterhuis et al. [25] emphasize, preventing both forms of leakage is critical for ensuring faithful local feature selections. In contrast, global feature selection methods

use a uniform mask for all instances, reducing opportunities for leakage but forfeiting the instance-specific explanations that make local approaches attractive. Nonetheless, leakage is primarily a concern in local feature selection, where the flexible, per-instance nature of the masking process can unintentionally encode extra information and undermine the trustworthiness of the selected features.

3.3.4. Challenges

Despite advancements in feature selection methods for neural models, their application in LTR presents several challenges:

- Ranking-Specific Optimization: Most existing methods were originally designed for classification or regression, making it unclear how well they adapt to ranking-specific objectives.
- Feature Leakage: Many feature selection techniques inadvertently introduce biases by conditioning feature importance on the model's predictions, reducing interpretability.
- Scalability: Feature selection in ranking requires evaluating feature importance across entire lists, making computation significantly more expensive than in standard supervised learning tasks.

3.3.5. Motivation for a Novel Feature Selection Approach

Given these challenges, there is a need for dedicated feature selection approaches designed specifically for neural LTR models. Existing methods do not fully address ranking-specific constraints, and feature leakage remains an unsolved problem. This thesis aims to develop a novel feature selection approach that aligns with ranking objectives while maintaining interpretability and computational efficiency. Inspired by the structured decision paths of GBDT models, we explore how feature selection can be effectively integrated into neural ranking architectures.

3.4. Summary

In this chapter, we reviewed the existing literature on ranking methods in Learning to Rank and feature selection techniques. We discussed the strengths and limitations of GBDT-based rankers and neural rankers, highlighting the evolution of ranking models. We also explored various feature selection methods, emphasizing their importance in improving model performance and interpretability. The gap identified in applying feature selection to neural rankers in LTR sets the foundation for the research presented in this thesis.

4

Methodology

This chapter outlines our approach to integrating feature selection into neural LTR. Neural rankers have not fully capitalized on explicit feature selection, consequently, neural models often require large feature sets, leading to reduced interpretability and higher computational cost.

However, many feature selection methods face a critical challenge: feature leakage and label leakage, wherein the selected features unintentionally contain information about hidden features and labels that should not be available at training time [25]. Oosterhuis et al. [25] have shown that such leakage can compromise both the interpretability and reliability of ranking models. They further propose a Sequential Unmasking without Reversion(SUWR) method with formal guarantees against leakage, demonstrating its effectiveness on tasks like MNIST digits recognition. Despite its promise, the application of this leakage-free approach in LTR scenarios remains unexplored.

Motivated by the need for robust feature selection in neural rankers and inspired by GBDT's iterative, split-based feature reduction, we adapt and extend the SUWR method to address ranking-specific objectives by incorporating loss functions that explicitly optimize towards NDCG. Our proposed *sequential feature selection* framework is designed to maintain ranking performance while enhancing interpretability and computational efficiency. By integrating SUWR's leakage-free principles with an NDCG-focused objective, our method is better aligned with the unique demands of LTR and seeks to provide a more principled path toward robust feature selection for neural rankers.

4.1. Overview of Sequential Feature Selection Method

Our framework for integrating feature selection into neural rankers in LTR builds upon the principle of sequential feature unmasking. The core idea is that, at any point in time, the model only observes a subset of available features. The decision to unmask additional features is given by a Selector, which uses only the currently revealed subset to predict whether new features would substantially improve ranking performance.

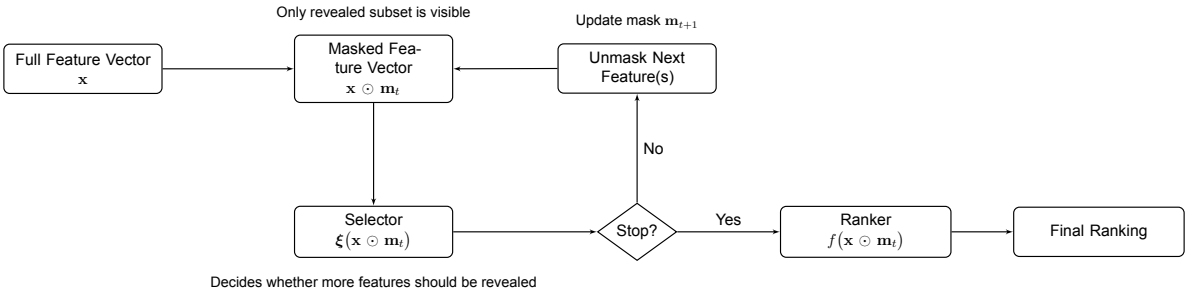


Figure 4.1: Illustration of the sequential feature selection procedure. At each time step t, the Selector decides whether to *stop* or *unmask* additional features. If it chooses to unmask more features, the mask is updated and the process repeats. If the Selector stops, the Ranker computes the final relevance score using the current masked input, outputting both the final ranking and mask.

Once a feature is unmasked, it remains visible for subsequent steps—the model never reverts a previously unmasked feature. This constraint ensures that hidden features remain inaccessible, preventing unintended signals about masked features from influencing future decisions.

In contrast to other traditional feature selection methods, which typically decide on a single subset of features all at once, our approach unfolds in multiple steps. At each step:

1. A partially observed feature vector $\mathbf{x} \odot \mathbf{m}$ is fed to the *Selector* network.
2. The *Selector* decides whether to stop (i.e., proceed with the current subset of features) or to unmask additional features.
3. If additional features are to be unmasked, the Selector generates a probability distribution over the currently masked features and then samples from this distribution to determine which feature(s) to reveal next.
4. The updated masked input is then passed to the *Ranker*, which computes the ranking and estimates performance metrics (e.g., NDCG) based on the currently revealed subset.

The process repeats until either the user-defined maximum number of features or training steps is reached or until the Selector decides that revealing more features would yield marginal utility. By only accessing the subset of features that have already been revealed, the Selector is guaranteed not to rely on or encode any hidden-feature information, thus mitigating the risk of feature leakage.

4.2. Formal Definition of Sequential Unmasking

Formally, let $\mathbf{x} \in \mathbb{R}^d$ be the full feature vector for a query–document pair, and let $\mathbf{m} \in \{0,1\}^d$ be a binary mask denoting which features are currently revealed ($m_i = 1$) or hidden ($m_i = 0$). Our feature selection method involves a sequence of T steps. At step t:

1. The Selector applies a learned function $\zeta_t^{\text{stop}}(\cdot)$ to the partially observed feature

vector $\mathbf{x} \odot \mathbf{m}_t$ to produce a scalar value

$$p^t_{\text{stop}}(\mathbf{x} \odot \mathbf{m}_t) \in [0, 1].$$

This value represents the probability of stoping further feature unmasking at this step. A Bernoulli trial (a random experiment with two outcomes, where a value of 1 indicates that the process should stop and 0 indicates that it should continue) is then performed using p^t_{stop}. If the trial yields 1, the unmasking process stops and the Ranker will later compute the final ranking based on the features revealed so far.

2. If the Bernoulli trial indicates that the process should continue, the Selector outputs a probability distribution

$$\pi^t(\mathbf{x} \odot \mathbf{m}_t) \in \Delta_{\{\text{masked}\}},$$

over the set of features that remain hidden. This distribution, defined over the remaining hidden features, represents how likely it is that revealing each one would improve ranking performance.

3. Based on the probability distribution $\pi^t(\mathbf{x} \odot \mathbf{m}_t)$, the Selector samples more features to reveal. Let newMask_t denote the sampled one-hot vector indicating the newly unmasked features. The mask is updated as

$$\mathbf{m}_{t+1} \;=\; \mathbf{m}_t \;+\; \text{newMask}_t.$$

Since $\text{newMask}_t \geq 0$, the mask only ever expands—once a feature is unmasked, it remains visible in all subsequent steps.

After step T (or earlier if the Bernoulli trial signals to stop), the Ranker f processes the fully updated masked input $\mathbf{x} \odot \mathbf{m}_T$ to compute the final ranking and estimate performance metrics like NDCG. And we record both the final prediction and the final mask $\mathbf{m}_T$. This sequential unmasking procedure ensures that each decision is based solely on the features that have already been revealed, thereby minimizing the risk of feature leakage.

(a) Traditional Feature Selection **(b)** Sequential Feature Selection

Figure 4.2: Comparison of Traditional vs. Sequential Feature Selection. In **(a)**, all features are observed in a single step, yielding one final subset. In **(b)**, features are revealed gradually across multiple steps, with each decision based on the partial set of unmasked features.

4.2.1. Leakage Prevention

In local feature selection, the Selector can leak information about currently hidden features or the relevance label by embedding it in the selection mask. Oosterhuis et al. [25] formally characterize this phenomenon, demonstrating that mitigating leakage

requires any selection policy to depend solely on features already revealed in previous steps. To that end, they propose a sequential unmasking without reversion algorithm, wherein features, once exposed, cannot be reverted back to hidden. By forbidding reversion, the selection process ensures each unmasking decision is based purely on the observed features, i.e., on $\mathbf{x} \odot \mathbf{m}_t$ at step t.

In this thesis, we adapt the general sequential unmasking framework of Oosterhuis et al. [25] to Learning-to-Rank scenarios by retaining its stepwise exposure of features and the no-reversion property, while incorporating ranking-specific loss functions during training phases. This modification preserves the leakage-free guarantees of the original algorithm, yet better aligns the selection process with ranking metrics critical to many real-world search and recommendation tasks.

4.3. Sequential Feature Selection for Neural Rankers

Algorithm 1 (adapted from [25]) illustrates how the inference proceeds. Initially, all features are masked ($\mathbf{m}_0 = \mathbf{0}$). At each iteration:

Algorithm 1 Sequential Feature Selection in Rankers

Require: Features $\mathbf{x} \in \mathbb{R}^d$; Initial Mask $\mathbf{m} \leftarrow \mathbf{0}$; Selector $\boldsymbol{\xi}$; Ranker f; Max step T; Max features $F_{\max}$.

for $t \leftarrow 1$ to T **do**
  $p_{\text{stop}} \leftarrow \xi^t_{\text{stop}}(\mathbf{x} \odot \mathbf{m})$
  if BernoulliTrial(p_{stop}) = `True` **or** $\|\mathbf{m}\|_1 \geq F_{\max}$ **then**
    return $\Big(f(\mathbf{x} \odot \mathbf{m}), \mathbf{m}\Big)$
  end if
  $\pi \leftarrow \xi^t_{\text{select}}(\mathbf{x} \odot \mathbf{m})$ ▷ Prob. over still-masked features
  newMask $\leftarrow$ SampleMask(π)
  $\mathbf{m} \leftarrow \mathbf{m} +$ newMask
end for
return $\Big(f(\mathbf{x} \odot \mathbf{m}), \mathbf{m}\Big)$

- At each step t, ξ^t_{stop} outputs $p_{\text{stop}} \in [0, 1]$. A Bernoulli trial with this probability determines whether to terminate, preventing unnecessary feature selection.
- If termination does not occur, the next features to reveal are sampled from $\pi \in \Delta_{\{\text{masked}\}}$. The updated mask $\mathbf{m}$ is strictly greater than the previous one, ensuring no-reversion in feature selection.
- The process ends if any of the following conditions are met: (i) the maximum number of steps T is reached, (ii) p_{stop} signals termination, or (iii) the predefined maximum number of revealed features has been reached. Once terminated, the ranker f evaluates the partially observed input $\mathbf{x} \odot \mathbf{m}$ to generate the final scoring.

4.3.1. Selector Network

The Selector network, denoted as $\boldsymbol{\xi}$, is implemented in our code as a deep neural network that takes the partially observed feature vector $\mathbf{x} \odot \mathbf{m}$ as input. The network comprises several layers:

- Encoder: A sub-network, denoted as FFenc, encodes the input into a hidden representation. In our experiments, this encoder is configured with 3 hidden layers.
- Stop Head: From the encoded representation, a stop head outputs a single probability value in $[0, 1]$, denoted as

  $$p_{\text{stop}}(\mathbf{x} \odot \mathbf{m}) = \zeta_{\text{stop}}(\mathbf{x} \odot \mathbf{m}).$$

  A sigmoid activation is applied so that this output can be interpreted as a probability for a Bernoulli trial used to decide whether to stop unmasking.
- Selection Head: A separate sub-network outputs raw scores over the remaining masked features. These scores are passed through a softmax layer to yield a discrete probability distribution

  $$\pi(\mathbf{x} \odot \mathbf{m}) \in \Delta_{\{\text{masked features}\}},$$

  ensuring that each probability $\pi_i \geq 0$ and that $\sum_i \pi_i = 1$. This distribution is interpreted as a discrete probability distribution from which one or more features are sampled to be unmasked next.

In our implementation, these two heads share initial hidden layers but branch out into separate output layers for the stop probability and the selection probability distribution, respectively. Missing features (where $\mathbf{m}_i = 0$) are zeroed out before being passed to the network, ensuring that the Selector never sees values of masked features. This design aligns with our goal of avoiding feature leakage.

Training. During training, we backpropagate through the Selector by sampling partial masks in an online fashion (see Section 4.4). For the stop head, we treat the output p_{stop} as the probability parameter of a Bernoulli trial; for the selection head, we interpret the softmax output as a discrete probability distribution from which we sample one or more features to reveal. Both heads are trained end-to-end. However, because sampling discrete features is a non-differentiable operation, we need to use special techniques to propagate gradients through the Selector. One common approach is to use the Gumbel-Softmax relaxation [17], which approximates discrete sampling with a continuous, differentiable function by adding Gumbel noise to the logits and applying a softmax with a temperature parameter; as the temperature decreases, the output more closely approximates a one-hot vector. This method allow gradients to flow through the Selector's decision-making process, enabling end-to-end training of the model despite the discrete nature of the sampling step.

4.3.2. Ranker Network

Let $f(\mathbf{x} \odot \mathbf{m}; \theta)$ denote our neural ranker. Our neural ranker is built using a deep neural network (DNN) architecture, and produces a relevance score for each document. These scores are then used to generate the final ranking by sorting the documents in order of decreasing relevance. In our implementation, the ranker is designed as follows:

- Input Processing: It receives the masked input $\mathbf{x} \odot \mathbf{m}$, where masked entries are replaced by zero.
- Hidden Layers: The input is processed through several fully-connected layers. In our experiments, the ranker typically consists of 3 hidden layers with non-linear activations such as ReLU, enabling the network to capture complex feature interactions.
- Output Layer: Finally, the network outputs a single relevance score for each document. These scores are then used to derive the final ranking by sorting the documents and to compute performance metrics such as NDCG.

Because the ranker only operates on the unmasked portion of the input, it is agnostic to the features that remain hidden. This ensures that the final ranking is based solely on the information revealed through the sequential feature selection process.

4.3.3. Integrating NDCG in training

In the leakage-free local selection framework of Oosterhuis et al. [25], the selector ϕ and predictor θ are jointly optimized so that the selector only conditions on the already revealed features, thereby avoiding feature or label leakage, and the predictor accurately models the label distribution from the selected inputs. Originally, the training signal combined a loss (e.g. mean squared error) with a sparsity regularization term. To adapt this approach to ranking scenarios, we replace the loss with NeuralNDCG [26] while preserving the leakage-free guarantees through sequential unmasking and no reversion.

NeuralNDCG is a differentiable metric that approximates NDCG used in ranking models to improve their performance on ranking tasks. It addresses the discrepancy between the optimization objective and the evaluation metric in LTR models by replacing the non-differentiable sorting operator in NDCG with NeuralSort, a differentiable approximation of sorting. NeuralNDCG consists of a multi-layer perceptron for learning latent document scores and an innovative NDCG layer for computing the NDCG score based on the predicted document scores. The NDCG layer has a temperature parameter that allows control over the trade-off between exploration and exploitation during training. It is formulated as below:

$$\mathrm{NeuralNDCG}_k^{(\tau)}(s, y) \;=\; N_k^{-1} \sum_{j=1}^{k} \Big(\mathrm{scale}\big(P^{(\tau)}\big) \cdot g(y) \Big)_j \cdot d(j)$$

where N^{-1} is the maxDCG at the kth rank, τ is the temperature parameter which allows to control the trade-off between the accuracy of the approximation and the variance of the gradients, $\mathrm{scale}(\cdot)$ is the Sinkhorn scaling, and $g(\cdot)$ and $d(\cdot)$ are the gain and discount functions, respectively.

By minimizing the negative NeuralNDCG, which serves as a ranking objective, our framework directly optimizes ranking quality and is better adpated to LTR tasks while preserving leakage-free feature selection.

Let $\{(x_i,\, y_i)\}_{i=1}^{N}$ be the training set, where $x_i \in \mathbb{R}^d$ represents the feature vector for instance i and y_i denotes its associated relevance labels. We denote by m_i the random mask sampled from the selector ϕ for x_i. Inspired by Oosterhuis et al. [25], we define the combined loss over the dataset as:

$$\mathcal{L}(\phi, \theta) \;=\; \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{m_i \sim \xi(x_i)} \Big[- \,\mathrm{NeuralNDCG}\big(f\big(x_i \odot m_i\big),\, y_i\big) \;+\; \lambda\, \Omega(m_i) \Big],$$

where:

- $\mathrm{NeuralNDCG}\big(f(x_i \odot h_i),\, y_i\big)$ denotes the differentiable NDCG reward computed from the subset of features revealed by h_i [26].
- $\Omega(h_i)$ is a regularization term (e.g., L1 or cardinality penalty on the number of selected features) that discourages revealing too many features, thus enhancing interpretability.
- λ controls the trade-off between maximizing ranking quality (via NeuralNDCG) and minimizing the number of unmasked features.
- The expectation $\mathbb{E}_{h_i \sim \zeta(x_i)}$ arises because the selector ζ samples masks stochastically, in line with the reinforcement learning perspective of sequential unmasking.

As in [25], the no reversion rule ensures once a feature is revealed, it remains unmasked for all subsequent steps, and the selector never has access to hidden feature values or labels. This construction guarantees no label leakage as the probability of selecting a particular mask h_i does not depend on y_i when ζ never observes the label directly. Similarly, it also guarantees no feature leakage as the probability of selecting a feature does not depend on features that remain hidden; each selection step relies only on the partially revealed input $\mathbf{x}_i \odot \mathbf{m}_t$. Hence, even though NeuralNDCG internally uses label information to compute a reward signal, the selector itself only sees previously unmasked features, preserving the leakage-free guarantee.

Altogether, by combining NeuralNDCG within the leakage-free sequential unmasking framework, we obtain a training objective that directly aligns local feature selection with ranking performance, without inadvertently exposing the selector to hidden features or true labels.

4.3.4. Implementation Details.

In our experiments, the ranker network is implemented as a standard DNN, typically using 3 hidden layers, with additional techniques such as dropout or batch normalization employed as needed to enhance generalization. The network is trained using a ranking loss, softmax cross-entropy, over the documents of a query, following standard LTR practices. This means that for each query, we collect all document vectors and their corresponding mask $\mathbf{x} \odot \mathbf{m}$ and compute the loss relative to the ground-truth relevance labels. Because f is agnostic to masked features, it can easily work with partial inputs. When only a few features are unmasked, the ranker simply makes predictions based on the information that is available.

Overall, the Selector and Ranker form a cohesive system: the Selector dynamically decides which features to reveal, while the Ranker predicts the relevance score for each document based on the unmasked features. This setup ensures that the final ranking reflects only the information actually revealed by the sequential feature selection procedure, mitigating feature leakage and improving interpretability.

4.3.5. Summary of Sequential Feature Selection for Neural Rankers

In summary, our sequential unmasking framework achieves the following:

1. It ensures that each decision step is based exclusively on the features that have already been revealed, thereby preventing any hidden information from influencing subsequent selections.
2. It maintains a strictly expanding mask by enforcing the no reversion rule, which prevents re-masking and thus minimizes the risk of feature leakage.
3. It integrates seamlessly with the neural ranker by incorporating a ranking loss that is designed to operate on partially observed feature inputs.
4. It balances ranking performance with the cost of feature acquisition by including a regularization term (e.g., $\lambda\|\mathbf{m}\|_1$) that encourages sparsity in the unmasking process.
5. It is adapted to LTR tasks by employing a differentiable approximation of the NDCG metric, NeuralNDCG. By optimizing the negative NeuralNDCG as a surrogate ranking objective, our framework directly enhances ranking quality while maintaining leakage-free feature selection.

This framework provides a principled feature selection method for neural Learning-to-Rank to enhance both interpretability and efficiency.

4.4. Training Strategies

Training our sequential feature selection framework requires jointly optimizing two components:

- **Selector** ξ: A network that determines, at each step, which features to reveal

and when to stop, based solely on the currently visible features.

- **Ranker** $f(\cdot;\theta)$: A deep neural network that produces relevance scores (which are then used to generate the final ranking) based on a partially masked input.

Our training framework supports two approaches: joint (end-to-end) training and a two-phase pipeline.

4.4.1. Joint Training

In joint training, both the Ranker and Selector are optimized simultaneously. The procedure is as follows:

1. Parameter Initialization: Initialize the Ranker's parameters θ and the Selector's parameters ϕ randomly, with a fixed random seed for reproducibility.
2. Forward Pass with Partial Masks: For each query–document pair in a mini-batch, a partial unmasking sequence is sampled from the Selector until a maximum steps/number of features or a stop decision is reached. The Ranker then computes relevance scores for the partially revealed feature vector $\mathbf{x} \odot \mathbf{m}$.
3. Loss and Regularization: The overall loss is defined as:

$$\mathcal{L}(\phi,\theta) \;=\; \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{m_i\sim\xi(x_i)}\Big[-\,\mathrm{NeuralNDCG}\big(f\big(x_i\odot m_i\big),\, y_i\big) \;+\; \lambda\,\Omega(m_i)\Big],$$

   where $\Omega(h_i)$ is a regularization term that discourages revealing too many features, thus enhancing interpretability, and λ controls the trade-off between maximizing ranking quality via NeuralNDCG and minimizing the number of unmasked features.
4. Gradient Estimation with Discrete Sampling: Since the unmasking decisions are discrete, we require special techniques to propagate gradients through Gumbel-Softmax Relaxation [17]: given logits z_i for each masked feature, we add Gumbel noise g_i and compute:

$$y_i = \frac{\exp\big((z_i+g_i)/\tau\big)}{\sum_j \exp\big((z_j+g_j)/\tau\big)},$$

   where τ is the temperature parameter. As τ decreases, the output y becomes closer to a one-hot vector, approximating the discrete sampling while still allowing gradients to flow.
5. Parameter Updates: Finally, we update both θ and ϕ using an optimizer such as Adam.

This integrated approach enables the Ranker to adapt to partial feature inputs while the Selector refines its policy for revealing features, thereby improving the final ranking performance and controlling the feature cost.

4.4.2. Two-Phase Training

Alternatively, training can be divided into two phases:

1. Phase 1 – Train Ranker: Train a conventional neural ranker $f(\mathbf{x};\theta)$ using the full feature set to establish a strong baseline.
2. Phase 2 – Train Selector: With θ fixed, train the Selector $\boldsymbol{\xi}$ to simulate partial unmasking so that the ranker scores masked inputs $f(\mathbf{x}\odot\mathbf{m})$. The Selector's parameters ϕ are optimized to minimize:

$$\min_{\phi}\ \mathbb{E}_{\mathbf{m}\sim\boldsymbol{\xi}}\Big[-\,\mathrm{NeuralNDCG}\big(f(x_i\odot m_i),\,y_i\big)\ +\ \lambda\,\Omega(m_i)\Big].$$

While this two-phase approach is simpler to implement, it is generally less optimal than joint training since the ranker does not adapt to partial feature inputs during its own training.

We will introduce further training implementation details in the next experiment chapter.

4.5. Summary of Methodology

In this chapter, we presented a comprehensive methodology for integrating our proposed sequential feature selection method into neural Learning-to-Rank systems. We began by examining the challenges in neural LTR—particularly its dependence on large, unrefined feature sets that can undermine interpretability and increase computational costs—and then explored the strengths of GBDT, drawing valuable insights from its iterative decision tree structure.

Motivated by these observations, we proposed a sequential feature selection framework that progressively unmasks features. At each step, the Selector operates solely on the currently revealed subset of features to decide whether to continue or stop further unmasking, thereby mitigating the risk of feature leakage. We provided a formal definition of this process, detailing the use of a learned stop function and a probability distribution via a softmax layer) for selecting the next features to reveal.

We then compared our approach with both traditional static methods and other dynamic feature selection techniques, highlighting its advantages in terms of adaptivity, leakage prevention, interpretability, and efficiency. Finally, we outlined our training strategies, describing both joint (end-to-end) and two-phase optimization pipelines, and discussed the use of gradient estimation techniques such as REINFORCE and Gumbel-Softmax relaxation to handle the discrete nature of the unmasking decisions.

Overall, our methodology offers a principled framework that not only improves computation efficiency by revealing only the most informative features, but also enhances interpretability in neural LTR systems. We will introduce further experiment details and results in the next chapter.

5

Experiment and Evaluation

5.1. Experiment Overview

To evaluate the effectiveness of the proposed sequential feature selection method in neural Learning-to-Rank settings, we design a set of experiments that directly correspond to the research questions outlined in Chapter 1. Our primary goal is to assess whether SUWR can maintain ranking performance while reducing input dimensionality, and whether it offers advantages in interpretability, efficiency, and feature selection quality.

Experiments are conducted on three widely used LTR benchmark datasets: MQ2008, Web10K, and Yahoo!. These datasets provide a diverse set of query-document pairs with rich feature representations, making them well-suited for evaluating feature selection strategies in neural ranking models.

To address RQ1, we compare the ranking performance of SUWR against a baseline neural ranker that uses the full feature set. This allows us to assess whether SUWR can achieve comparable results with significantly fewer features.

To address RQ2, we evaluate SUWR against other existing feature selection methods, such as L2X and CAE, under a fixed feature budget. This setup enables a fair comparison of how effectively each method selects relevant features that contribute to ranking performance, while also analyzing the trade-off between feature sparsity and accuracy.

To investigate RQ3, we evaluate the impact of different LTR-specific loss functions (pointwise, pairwise, and listwise) on SUWR's ability to select informative features and maintain high ranking quality. This helps determine which loss formulations best support effective sequential feature selection.

Finally, to address RQ4, we analyze the consistency of SUWR's selected features across queries. We measure how frequently individual features are selected and examine whether the most frequently selected features are consistent across different ranking contexts, offering insights into the interpretability and generalization of the

model.

The remainder of this chapter introduces the datasets, details the experimental setups for each research question, describes the evaluation metrics and baseline methods, and presents a comprehensive analysis of the results.

5.2. Datasets

To evaluate the proposed feature selection framework for neural LTR, we conduct experiments on three widely used benchmark datasets: MQ2008, Web10k, and Yahoo!. These datasets provide different ranking environments in terms of feature dimensionality, query-document distributions, and relevance annotations, making them suitable for assessing the generalizability and effectiveness of feature selection in neural rankers.

Each dataset consists of query-document pairs, where each document is represented by a feature vector and assigned a graded relevance label. The datasets also incorporate multiple numerical ranking features derived from search engine signals, document-query interactions, and retrieval heuristics such as term frequency and BM25 scores. A summary of dataset statistics is presented in Table 5.1.

5.2.1. MQ2008

MQ2008 is a dataset from the LETOR 4.0 collection [28], consisting of 784 queries sampled from real-world search logs, each associated with a set of retrieved documents. Each document-query pair is represented by 46 numerical features, derived from retrieval heuristics such as BM25, query-document similarity, and PageRank. Relevance labels range from 0 to 2, with higher values indicating stronger relevance.

5.2.2. Web10k

The Microsoft LTR Web10k dataset [27] contains 10,000 queries and approximately 1.2 million query–document pairs. Each document is represented by 136 numerical features, such as term frequency statistics, BM25 scores, and user behavior signals. Relevance labels range from 0 to 4, providing a more fine-grained benchmark compared to smaller datasets like MQ2008. Given its larger size and richer label space, Web10k offers a challenging yet realistic environment for evaluating Learning-to-Rank algorithms.

5.2.3. Yahoo!

Yahoo! LTR Challenge [8] contains 36,000 queries and 709,877 query-document pairs, with each document represented by 700 features extracted from Yahoo's search engine logs. Features include content-based signals, click-through rates, and link-based metrics. This dataset represents a high-dimensional real-world ranking scenario.

Table 5.1: Summary of datasets used for evaluation.

Dataset	# Queries	# Documents	# Features	Relevance Levels
MQ2008	784	∼15k	46	{0,1,2}
Web10k	10,000	∼1.2M	136	{0,1,2,3,4}
Yahoo!	36,000	∼710k	700	{0,1,2,3,4}

5.3. Experiment Setup

This section outlines the experimental setup used to evaluate the proposed sequential feature selection framework in neural Learning-to-Rank models. Our experiments are designed to answer the research questions defined in Chapter 1, focusing on evaluating SUWR's performance in terms of ranking effectiveness, feature sparsity, interpretability, and sensitivity to different loss functions.

For each dataset, we preprocess and split the data into training, validation, and test sets following standard LTR practices.

For **RQ1**, which investigates whether SUWR can achieve comparable ranking performance to baseline neural rankers trained on the complete feature set, we use a neural ranker trained with all available features as our primary baseline. Additionally, we include LambdaMART, a strong tree-based LTR model widely recognized for its competitive ranking performance, as a reference point to contextualize SUWR's results against established ranking approaches. We evaluate and compare the ranking quality of each method using NDCG@1, NDCG@5, and NDCG@10.

For **RQ2**, which explores how SUWR compares with existing feature selection methods, we design experiments in two scenarios: unconstrained and fixed-budget. In the unconstrained scenario, feature selectors dynamically determine the number of features to select based on their contribution to ranking performance, without an imposed selection limit. Here, we compare SUWR against TabNet and LassoNet, methods that inherently operate without explicit constraints on numbers of selections. In contrast, in the fixed-budget scenario, we explicitly limit all feature selection methods to choose a predetermined number of features (e.g., top-5). Under this scenario, SUWR is compared with composable methods such as L2X and CAE, which naturally support fixed feature constraints. Ranking effectiveness across these comparisons is assessed using the same metrics: NDCG@1, NDCG@5, and NDCG@10. Additionally, to analyze the relationship between feature sparsity and ranking accuracy, we conduct experiments by incrementing the proportion of selected features and evaluating the corresponding ranking performance using NDCG@10. This analysis helps identify whether selecting more features consistently leads to improved ranking effectiveness, or if diminishing returns occur beyond a certain selection threshold.

For **RQ3**, we aim to investigate how different loss functions affect SUWR's performance in terms of feature selection and resulting ranking accuracy. By default, our approach employs a listwise loss function, NeuralNDCG, which leverages ranking information to guide the training process, potentially leading to more relevant features.

However, the influence of alternative loss functions remains an open question. To explore this, we design experiments comparing the effectiveness of three different types of loss functions: a pointwise loss (Mean Squared Error), a pairwise loss (logistic loss, as used in RankNet [3]), and our default listwise loss (NeuralNDCG). In these experiments, we standardize the number of selected features across all loss functions to ensure fairness. The performance of each loss function is then assessed by comparing their ranking effectiveness using metrics NDCG@1, NDCG@5, and NDCG@10, enabling us to identify which loss most effectively guides SUWR towards selecting features that enhance ranking quality.

For **RQ4**, we investigate whether the features selected by SUWR are consistent across queries, thus providing enhanced interpretability and insights into the feature selecting process. Since datasets such as MQ2008 and Web10K explicitly define and label their features (e.g., BM25 scores, IDF values, PageRank), we analyze feature selection frequencies across all queries to identify patterns in feature importance. By identifying features that are consistently chosen by SUWR, we aim to determine whether these frequently selected features align with known, critical metrics in LTR context. Given that SUWR is designed to be leakage-free, any consistently selected feature is more likely to genuinely contribute to ranking performance, thereby providing reliable insights into feature importance and confirming that the selected features faithfully reflect the ranking task.

5.4. Experiment Results

In this section, we present experimental results to answer the research questions defined in Chapter 1. Specifically, we evaluate whether our sequential feature selection method (SUWR) can achieve comparable ranking performance to full-feature baseline neural rankers, how it compares against existing feature selection methods under both unconstrained and fixed-budget scenarios, the impact of different loss functions on SUWR's ranking effectiveness, and the consistency and interpretability of the selected features across queries. We compare our proposed method to established feature selection techniques (L2X, CAE, TabNet, and LassoNet) as well as to strong baselines, including a neural ranker trained on all features and the widely-used LambdaMART model.

5.4.1. RQ1: Ranking Performance of SUWR Compared to Full-Feature Baselines

RQ1 investigates whether our sequential feature selection method, SUWR, can achieve ranking performance comparable to neural rankers trained on the complete feature set. Tables 5.2, 5.3, and 5.4 show experimental results across the three benchmark datasets: MQ2008, Web10K, and Yahoo!.

From these results, we observe that both the full-feature neural ranker and LambdaMART establish strong baseline performances across all datasets. SUWR, however, achieves substantial reductions in the number of selected features, demonstrat-

ing a significant strength in computational efficiency. For instance, on the MQ2008 dataset, SUWR selects only around 6 features (an approximately 87% reduction from the full set), resulting in a modest performance drop of roughly 6% in NDCG@10. This demonstrates SUWR's ability to identify informative features effectively while maintaining competitive ranking performance.

In contrast, the performance drop on the Web10K dataset is more noticeable (approximately 21%), suggesting that SUWR's highly restrictive feature selection may be overly aggressive for large-scale datasets, potentially neglecting features whose information significantly contribute to ranking effectiveness. This pattern indicates that although SUWR excels at minimizing the number of features used, it might not adequately capture complex feature interactions critical for ranking in larger and more nuanced data scenarios.

Based on these findings, we address **RQ1** as follows: SUWR effectively reduces computational costs by selecting substantially fewer features, though this typically leads to a moderate decrease in ranking accuracy. Such a trade-off may be acceptable in scenarios where interpretability or efficiency is prioritized. Additionally, SUWR performs notably well on smaller datasets such as MQ2008, where it successfully identifies essential ranking signals. However, its unmasking strategy may limit its effectiveness on larger, more complex datasets such as Web10K, suggesting that future feature selection strategies should incorporate adaptive mechanisms capable of capturing more complex feature interactions in larger datasets.

Table 5.2: Ranking Performance on MQ2008 (Unconstrained Setting)

Model	NDCG@1	NDCG@5	NDCG@10	Avg. # Selected Features
Full-feature Neural Ranker	0.679 ± 0.016	0.722 ± 0.011	0.757 ± 0.008	46
LambdaMART	0.684 ± 0.020	0.742 ± 0.0015	0.765 ± 0.018	46
SUWR	0.646 ± 0.025	0.688 ± 0.021	0.719 ± 0.016	7
TabNet	0.661 ± 0.017	0.721 ± 0.015	0.734 ± 0.012	7 ± 3
LASSONET	0.659 ± 0.022	0.718 ± 0.016	0.732 ± 0.016	7 ± 3

Table 5.3: Ranking Performance on Web10k (Unconstrained Setting)

Model	NDCG@1	NDCG@5	NDCG@10	Avg. # Selected Features
Full-feature Neural Ranker	0.419 ± 0.010	0.473 ± 0.006	0.491 ± 0.005	136
LambdaMART	0.436 ± 0.005	0.490 ± 0.004	0.505 ± 0.003	136
SUWR	0.358 ± 0.013	0.373 ± 0.009	0.399 ± 0.008	10
TabNet	0.426 ± 0.006	0.468 ± 0.007	0.485 ± 0.004	10 ± 2
LASSONET	0.354 ± 0.008	0.367 ± 0.005	0.382 ± 0.007	10 ± 2

5.4.2. RQ2: Comparing SUWR with Existing Feature Selection Methods in Neural Rankers

This research question investigates how SUWR compares to existing feature selection methods in terms of ranking accuracy and feature sparsity. To thoroughly explore this comparison, we designed two experimental scenarios: an unconstrained scenario,

Table 5.4: Ranking Performance on Yahoo! (Unconstrained Setting)

Model	NDCG@1	NDCG@5	NDCG@10	Avg. # Selected Features
Full-feature Neural Ranker	0.640 ± 0.004	0.698 ± 0.003	0.726 ± 0.004	700
LambdaMART	0.695 ± 0.003	0.736 ± 0.002	0.775 ± 0.003	700
SUWR	0.629 ± 0.012	0.685 ± 0.008	0.718 ± 0.007	10
TabNet	0.688 ± 0.005	0.730 ± 0.004	0.773 ± 0.002	8 ± 2
LASSONET	0.632 ± 0.008	0.692 ± 0.006	0.721 ± 0.005	10 ± 2

where methods dynamically determine the number of features to select, and a fixed-budget scenario, where all methods select a predefined number of features.

In the unconstrained scenario, we train and evaluate two feature selection methods, TabNet and LassoNet, both of which inherently determine their number of selected features without explicit constraints. These methods require hyperparameter tuning to balance feature sparsity and ranking effectiveness. To ensure fairness in our comparison, we configure SUWR to select the same number of features as identified by TabNet and LassoNet, allowing us to directly evaluate whether SUWR can select more informative features with the same number of selection.

Tables 5.2, 5.3, and 5.4 show results with SUWR comparing against TabNet and LASSONET. Across all three datasets, TabNet consistently achieves the highest ranking performance. TabNet's superior accuracy largely benefits from its tree-style neural architecture, and also its attentive transformer blocks, which effectively transform and select features through sparsemax activation. This mechanism allows TabNet to implicitly capture intricate feature relationships and adaptively model feature importance. However, this approach might also introduce feature and label leakage by encoding additional information from unselected features or labels, potentially inflating its ranking performance.

In contrast, SUWR explicitly prevents feature leakage through a masking mechanism. Although SUWR achieves slightly lower ranking accuracy compared to TabNet, it consistently matches or outperforms LassoNet. This indicates that SUWR reliably isolates relevant features without relying on potentially encoding extra information from masked features. Nevertheless, the observed performance gap with TabNet, especially on larger datasets such as Web10K, highlights room for improvement to enhance SUWR's capability in modeling more complex feature interactions.

Figures 5.1, 5.2, and 5.3 summarize ranking performance on MQ2008, Web10k, and Yahoo! datasets under the fixed-budget scenario, where models are explicitly limited to selecting only a small subset of available features (approximately 10% for MQ2008 and Web10k, and roughly 1% for Yahoo!). Across all datasets, SUWR consistently outperforms CAE, demonstrating a clear advantage over it. It also performs comparably to L2X, with stronger performance on Web10k, slightly better results on MQ2008, and nearly identical outcomes on Yahoo!.

This overall performance highlights SUWR's effectiveness compared to other sampling-based feature selection methods, such as CAE and L2X. Although all three methods share a common sampling-based architecture, which is jointly optimizing a probabilis-

tic feature selector and a ranker, they differ notably in how feature importance distributions are constructed and optimized. SUWR sequentially unmasks features based explicitly on their incremental contribution to ranking performance, ensuring leakage-free and interpretable feature selection. In contrast, methods such as L2X rely on instance-level feature selection through repeated stochastic sampling, potentially resulting in less stable selections across instances, while CAE uses an autoencoder-based reconstruction strategy that may prioritize redundancy reduction rather than direct ranking relevance. SUWR's explicit sequential masking procedure allows it to consistently identify informative feature subsets that enhance ranking effectiveness in fixed-budger settings.

Collectively, the fixed-budget scenario clearly demonstrates that SUWR outperforms or matches other composable filtering-based feature selection methods, confirming its suitability and competitiveness when feature budgets are limited.

Relationship Between Selected Feature Proportion and Performance Figures 5.4, 5.5, and 5.6 further investigate the impact of increasing the proportion of selected features on ranking performance of SUWR compared with filtering-based methods (CAE and L2X).

A notable trend across all three datasets is that performance generally improves as the proportion of selected features increases, yet the rate and pattern of improvement differ between the methods. SUWR consistently demonstrates performance improvement even when selecting a very limited number of features, indicating that it effectively identifies highly influential features early in the selection process. This is particularly advantageous in practical ranking scenarios, where computational efficiency and interpretability often demand significantly reduced feature sets.

By comparison, CAE and L2X exhibit different behavior: CAE's performance improvements are generally modest and fluctuate inconsistently with increasing feature proportion. SUWR consistently outperforms CAE at nearly all feature proportions, suggesting that CAE's autoencoder-based reconstruction criterion may not effectively prioritize the most ranking-relevant features, especially at limited feature budgets.

On the other hand, L2X improves consistently and notably across datasets, demonstrating a steady ability to identify increasingly informative subsets of features. SUWR achieves stronger performance compared to L2X on MQ2008 and Web10k datasets but slightly falls short on Yahoo! dataset, which has much higher feature dimensions. This result suggests that L2X's repeated sampling and selection maximization may more effectively model complex feature interactions in highly dimensional input spaces for ranking tasks.

Conclusion for RQ2 Our analyses yield clear insights into RQ2: SUWR consistently excels in fixed-budget scenarios, effectively outperforming or matching other filtering-based feature selection methods (CAE, L2X) in the context of neural ranking in LTR, and shows distinctive advantages in early selection stages across incremental feature selection proportions. The results underscore SUWR's strength in identifying highly informative features quickly and efficiently, providing clear advantages in

terms of interpretability and computational efficiency without substantial sacrifices in ranking performance. However, TabNet still outperforms SUWR in all three datasets, suggestingn future enhancements in SUWR should consider improved strategies for leveraging additional complementary features, especially in large-scale, more complex ranking tasks.

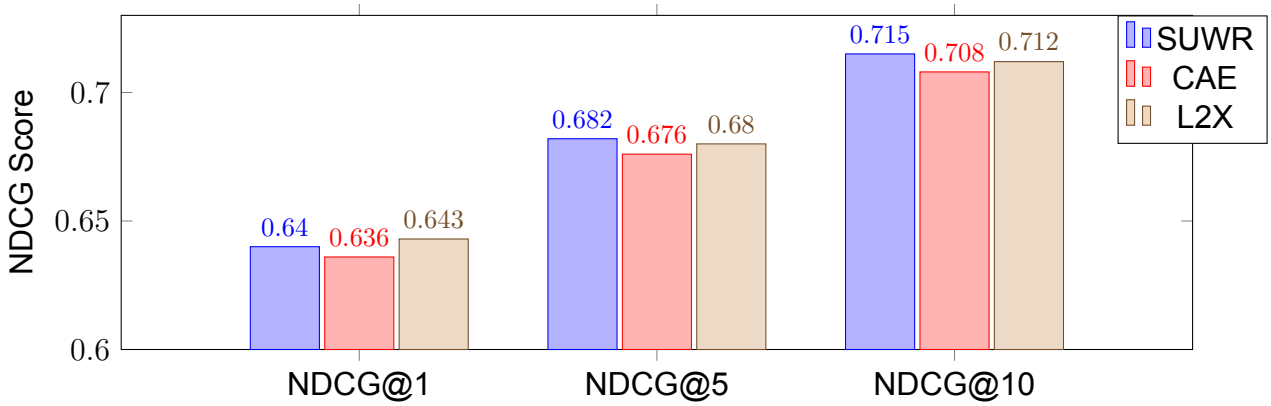


Figure 5.1: Ranking Performance on MQ2008 (Fixed Budget: 5 Features)

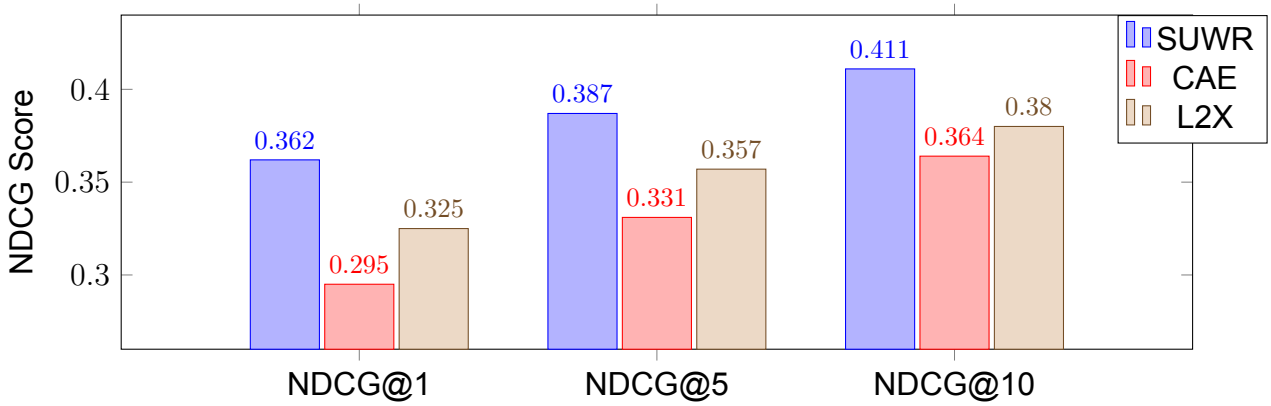


Figure 5.2: Ranking Performance on Web10k (Fixed Budget: 14 Features)

The results indicate that even when only a small fraction of the full feature set is used, our method is capable of achieving reasonable ranking performance. And as more features are revealed, the performance of our method steadily improves.

While incorporating more features generally leads to improved ranking accuracy, a carefully selected subset of highly informative features can capture most of the essential relevance signals. Our method effectively identifies these critical features, as evidenced by its competitive performance even at low feature budgets. In addition, our method consistently outperforms CAE, and its performance is comparable to that of L2X.

Overall, these findings highlight that with only a small proportion of features, our method achieves a favorable balance between ranking performance and computational efficiency. This is particularly important in real-world applications, where reducing the number of features can lead to significant efficiency gains without sacrificing much in terms of ranking quality.

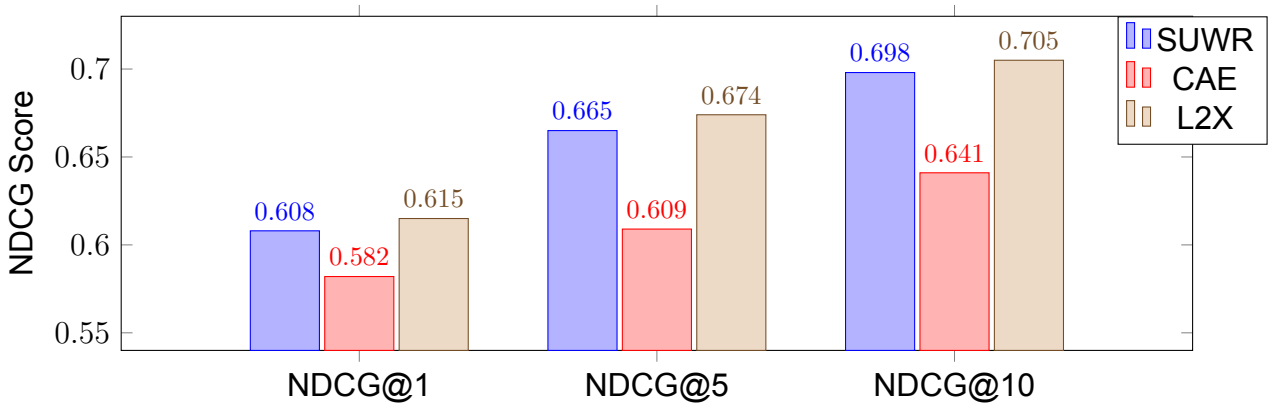


Figure 5.3: Ranking Performance on Yahoo! (Fixed Budget: 7 Features)

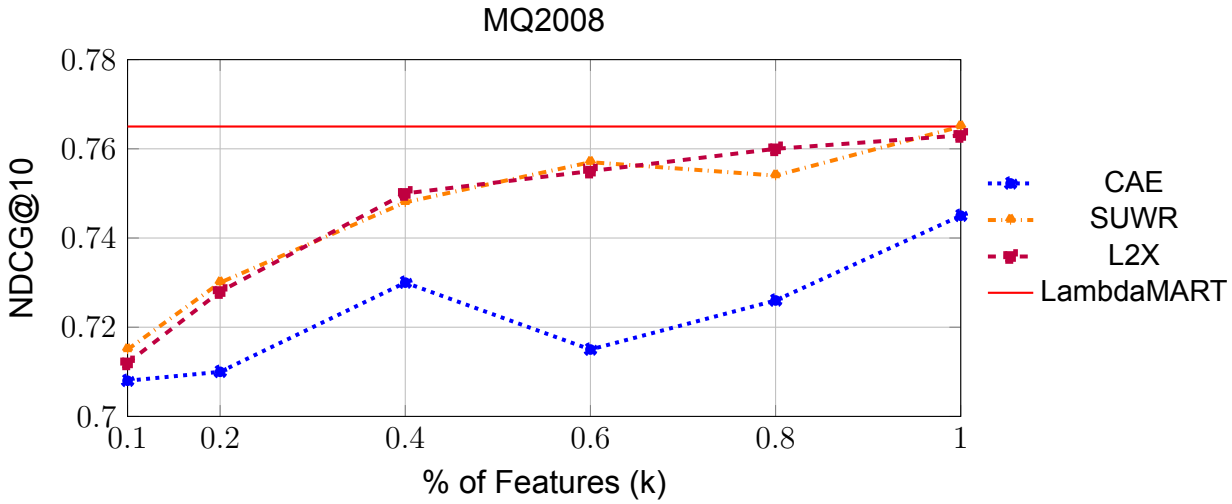


Figure 5.4: Impact of Unmasked Feature Proportion on Ranking Performance on MQ2008.

5.4.3. RQ3: Impact of Different Loss Functions on SUWR Performance

In RQ3, we investigated how the choice of training loss function—pointwise (MSE), pairwise (RankNet), and listwise (NeuralNDCG)—affects the ranking performance of SUWR, given an identical number of selected features. Figures 5.7, 5.8, and 5.9 summarize the comparative performance across MQ2008, Web10k, and Yahoo! datasets.

Pointwise loss treats ranking as a regression problem, where the model predicts the relevance score of each document independently. We use Mean Squared Error (MSE) as our pointwise loss, which minimizes the squared difference between predicted and true relevance scores. While simple and widely used, MSE does not consider the relative ordering of documents, which may limit its effectiveness in ranking tasks.

Pairwise loss functions aim to optimize the relative order between pairs of documents instead of absolute relevance scores. We use logistic loss, which is commonly used in RankNet, to minimize the probability of ranking a less relevant document above a more relevant one. This loss function better captures relative ranking but does not directly optimize global ranking metrics like NDCG.

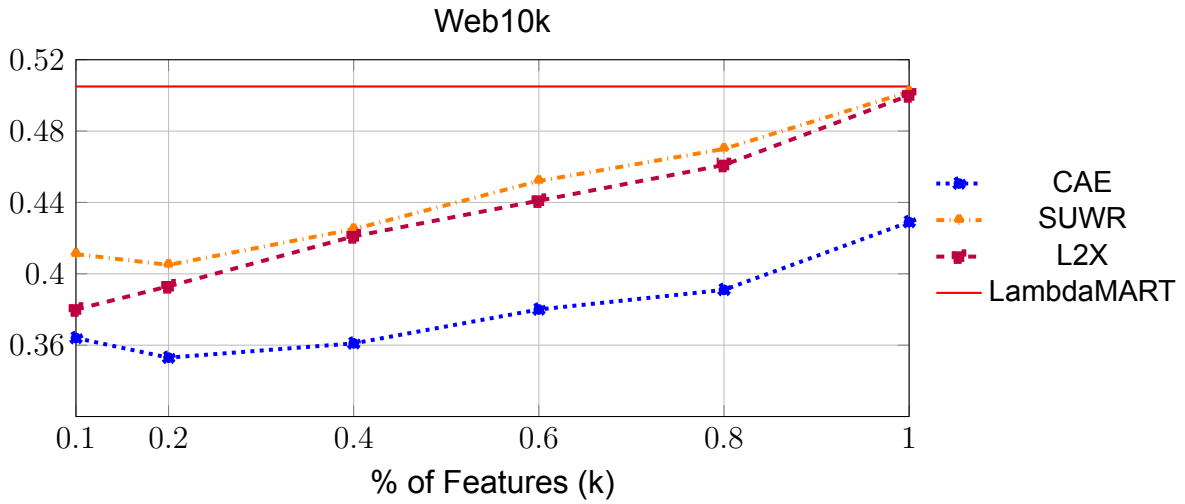


Figure 5.5: Impact of Unmasked Feature Proportion on Ranking Performance on Web10k.

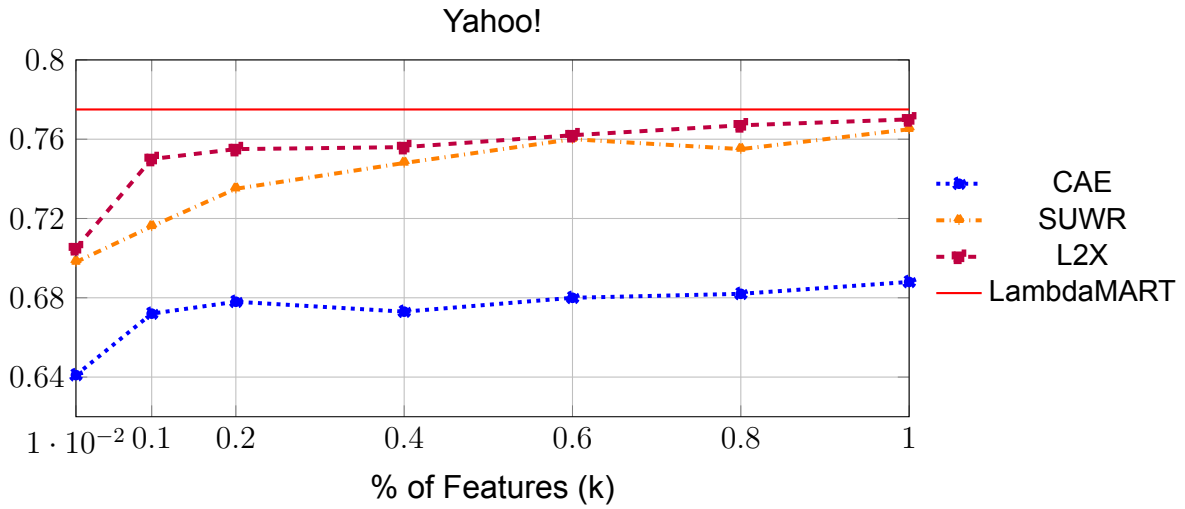


Figure 5.6: Impact of Unmasked Feature Proportion on Ranking Performance on Yahoo!.

Listwise loss functions optimize the ranking of entire document lists, rather than treating individual items or pairs separately. We use NeuralNDCG, a differentiable approximation of NDCG, which directly aligns the learning objective with ranking metrics, leading to improved ranking performance, especially in neural ranking models.

A clear and consistent pattern emerges across all datasets: the listwise loss function achieves significantly better ranking performance compared to both pairwise and pointwise losses. The pointwise loss consistently yields the weakest results, reflecting its fundamental limitation in optimizing a ranking problem, as it treats each item independently, neglecting inter-item relationships crucial to effective ranking. The pairwise loss notably outperforms the pointwise loss but still falls short of the listwise loss performance, as it explicitly optimizes the relative ordering between pairs of documents but may miss higher-order dependencies within ranked lists.

The superior performance of the listwise loss can be directly attributed to its alignment

with the nature of ranking tasks. By optimizing ranking metrics over entire ranked lists, the listwise loss effectively enhances SUWR's ability to select the most informative features for better ranking. This list-level optimization encourages the selector to choose features that consistently improve the accuracy of ranking predictions.

Overall, these results demonstrate that a listwise training approach aligns best with the SUWR method's objective. The clear advantage of NeuralNDCG over RankNet and MSE underscores the importance of matching loss functions explicitly to the structure of the ranking problem. In practical terms, adopting a listwise loss function like NeuralNDCG is essential for effectively leveraging SUWR's sequential, leakage-free feature selection strategy, enabling it to consistently identify features that robustly improve ranking performance.

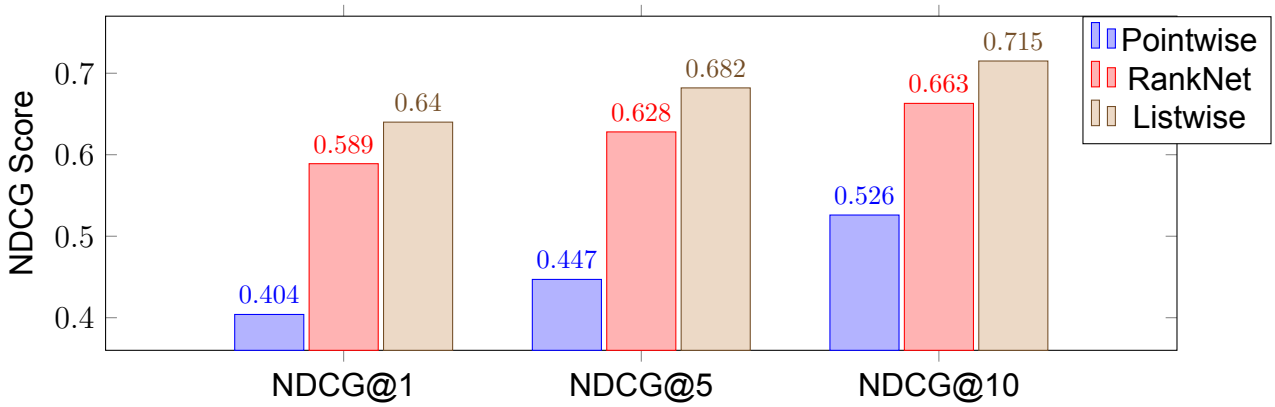


Figure 5.7: Impact of Loss Functions on Ranking Performance with 5 features selected (MQ2008)

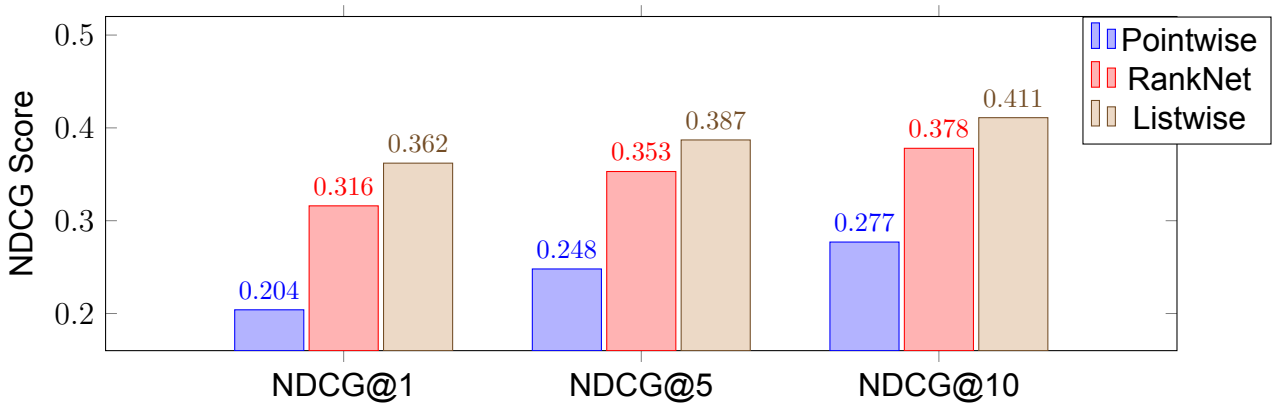


Figure 5.8: Impact of Loss Functions on Ranking Performance with 14 features selected (Web10k)

5.4.4. RQ4: Consistency and Interpretability of Selected Features

RQ4 investigates whether SUWR selects features consistently across different queries, thereby enhancing interpretability and trustworthiness in ranking decisions. Tables 5.6 and 5.5, along with the accompanying heatmaps, illustrate the most frequently selected features by SUWR on the Web10k and MQ2008 datasets, as well as their selection frequencies relative to L2X.

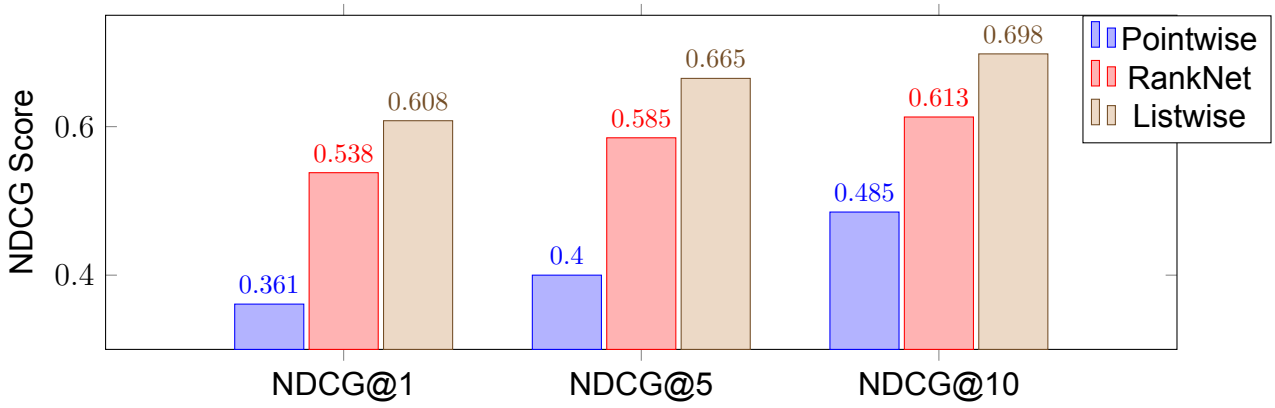


Figure 5.9: Impact of Loss Functions on Ranking Performance with 7 features selected (Yahoo!)

Feature Name	Feature ID
Query–document cosine similarity (body)	20
TF-IDF (body)	17
BM25 (body)	15
BM25 (title)	16
LMIR.JM of title	38

Table 5.5: Top 5 most frequently selected features on MQ2008, ranked in descending order of selection frequency.

The results clearly demonstrate that SUWR consistently prioritizes features widely recognized as critical ranking signals within LTR tasks. On Web10k, SUWR consistently selects traditional ranking metrics such as language modeling scores (e.g., LMIR.JM with Jelinek-Mercer smoothing), BM25 scores, inlink counts, and quality indicators like QualityScore. Similarly, on MQ2008, SUWR prominently selects fundamental text-based features including query-document cosine similarity, TF-IDF, and BM25 scores across title and body texts. These features are well-known signals that strongly correlate with document relevance in ranking literature.

Comparing SUWR's selection patterns with L2X highlights the robustness and consistency of SUWR's choices. For instance, on MQ2008, SUWR and L2X both frequently select top-ranked features, though their prioritization differs. The heatmap (Figure 5.11) indicates that SUWR consistently selects certain features such as cosine similarity (body) and TF-IDF (body) across queries with relatively high frequency, whereas L2X exhibits a somewhat less consistent and different selection pattern. This discrepancy suggests that while both methods identify relevant features, SUWR's selection process is more stable and interpretable due to its explicit leakage-free mechanism, enhancing the reliability and trustworthiness of the identified features.

The critical advantage of SUWR is its explicit design to eliminate feature and label leakage. SUWR ensures that each selected feature genuinely contributes to improved ranking performance without leveraging hidden correlations. Thus, the features identified by SUWR are inherently more faithful and trustworthy compared to those from other selection methods that might encode unintended or leaked information.

Feature Name	Feature ID
LMIR.JM (body)	121
BM25 (anchor)	107
Inlink number	128
BM25 (body)	106
QualityScore	132
Covered query term ratio (whole doc)	10
BM25 (title)	108
Sum of TF*IDF (body)	71
LMIR.DIR (body)	116
Query-url click count	134
LMIR.JM (title)	123
LMIR.DIR (title)	118
PageRank	130
URL click count	135

Table 5.6: Top 10 most frequently selected features on WEB10K, ranked in descending order of selection frequency.

Consequently, the features selected by SUWR provide reliable, transparent insights into the factors driving ranking decisions, directly addressing RQ4. SUWR consistently selects a core set of meaningful features across different queries, as demonstrated by the high and stable selection frequencies of well-known relevance signals such as BM25, TF-IDF, cosine similarity, and language-model-based scores. This consistency enhances interpretability by allowing us to identify which features the model repeatedly relies on to produce high-quality rankings.

Furthermore, because SUWR eliminates both feature and label leakage through its sequential masking mechanism, the features it selects can be trusted as responsible for the model's ranking behavior, not as a result of hidden or indirect signals. This makes SUWR's selections not only consistent but also faithful to the task, enabling researchers to acquire credible insights from them. In summary, SUWR produces stable, interpretable, and trustworthy feature selections, thus effectively answering RQ4 and confirming that it is well-suited for scenarios where understanding and explaining ranking decisions is critical.

5.5. Summary

In this chapter, we conducted a set of experiments to evaluate the effectiveness of the proposed sequential feature selection method, SUWR, in neural Learning-to-Rank models. Through these experiments, we addressed all four research questions outlined in Chapter 1.

For **RQ1**, we demonstrated that SUWR is capable of achieving competitive ranking performance while selecting only a small subset of input features, offering significant improvements in computational efficiency without severe loss in accuracy compared

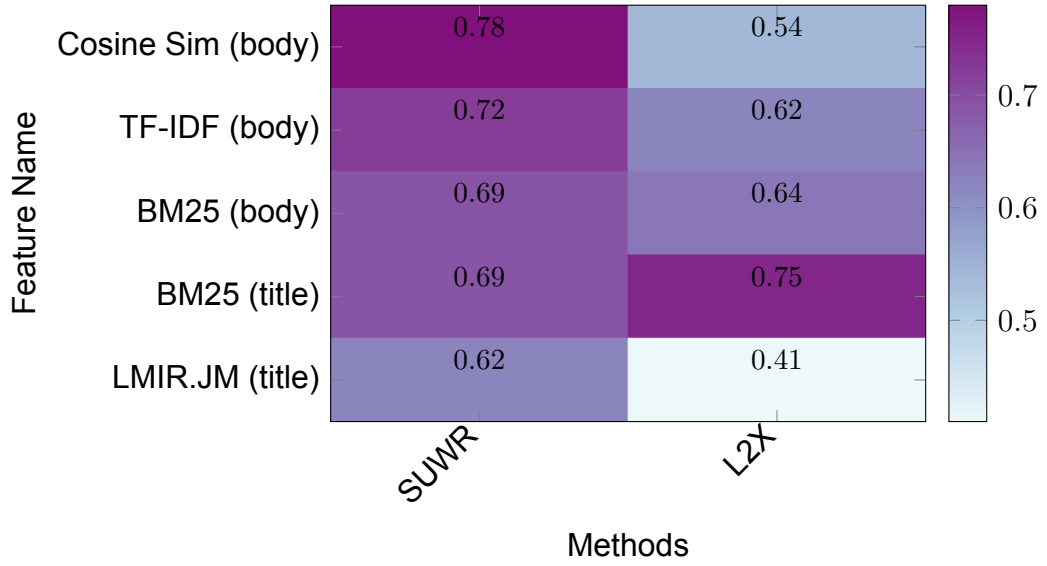


Figure 5.10: Heatmap showing how frequently the top 5 features selected by SUWR were chosen across all queries on MQ2008, along with their corresponding selection frequencies by L2X. Higher values indicate more frequent selection.

to full-feature neural rankers and LambdaMART baselines.

In **RQ2**, we compared SUWR against existing feature selection methods in both unconstrained and fixed-budget settings. SUWR consistently outperformed CAE and performed competitively with L2X, particularly in smaller datasets. These results highlight SUWR's strength in selecting compact and relevant feature subsets, especially under fixed budgets.

Addressing **RQ3**, we examined the impact of different loss functions on SUWR's performance. The results show that listwise losses, lead to the best ranking outcomes, reinforcing the importance of aligning the optimization objective with the structure of the ranking task.

Finally, for **RQ4**, we analyzed the consistency and interpretability of the features selected by SUWR. We found that SUWR selects stable and widely accepted relevance features across queries, and due to its leakage-free design, these selections can be considered trustworthy and faithful to the model's ranking behavior.

Collectively, these findings validate SUWR as a robust, interpretable, and efficient approach for feature selection in neural Learning-to-Rank systems.

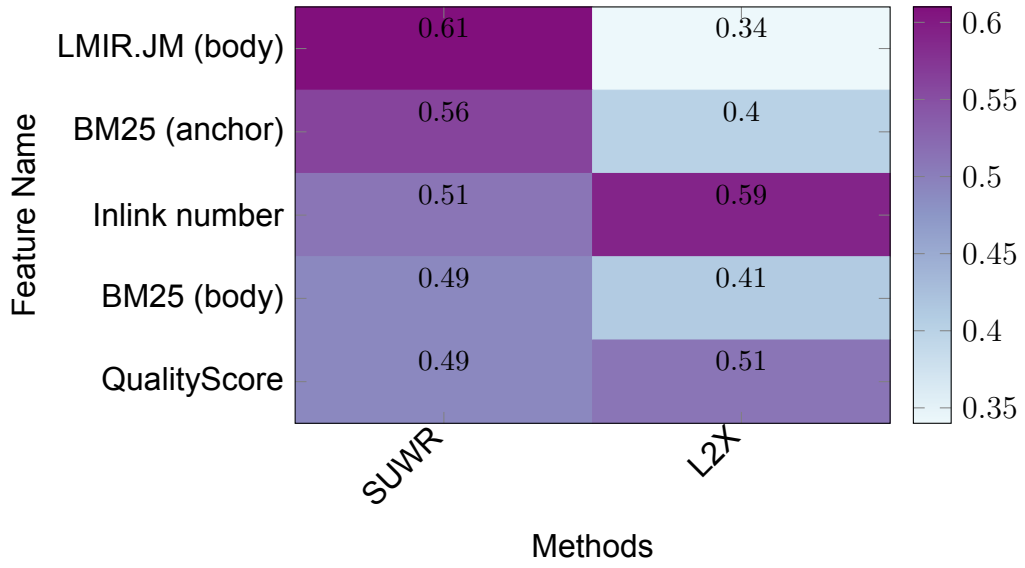


Figure 5.11: Heatmap showing how frequently the top 5 features selected by SUWR were chosen across 1000 queries on Web10k, along with their corresponding selection frequencies by L2X.Higher values indicate more frequent selection.

6

Conclusion

This chapter summarizes the main findings and contributions of the research presented in this thesis, highlights its limitations, and proposes future directions for extending this work.

6.1. Summary of the Thesis

This thesis investigates the application of SUWR, a sequential feature selection method originally proposed in prior work [25], within the context of neural Learning-to-Rank (LTR) models. The method aims to improve both interpretability and computational efficiency, while explicitly addressing the challenges of feature and label leakage. To better adapt SUWR to the LTR setting, we extend it by incorporating a selector that sequentially unmasks input features to the ranker, ensuring that only the selected features contribute to the final ranking output, and by integrating NeuralNDCG, a differentiable approximation of the standard NDCG metric, into the training objective. The masking mechanism guarantees that unselected features remain entirely inaccessible to the model, thereby preventing leakage and preserving the integrity of the feature selection process.

The proposed method was evaluated on three widely used benchmark LTR datasets: MQ2008, Web10k, and Yahoo!. Experimental results showed that SUWR achieves competitive ranking performance while drastically reducing the number of features used, thus significantly lowering model complexity. Furthermore, the sequential selection process yields interpretable insights into feature importance, revealing which signals are most influential in ranking decisions. These characteristics contribute to building ranking models that are not only efficient and accurate, but also transparent and trustworthy.

In sum, this thesis demonstrates that sequential, leakage-free feature selection can serve as a robust and interpretable approach to enhancing neural ranking models, with strong potential for application in real-world Learning-to-Rank scenarios.

6.2. Limitations

While the proposed framework demonstrates promising results, several limitations remain. First, the balance between ranking performance and feature sparsity depends on hyperparameters. Although tuning these parameters yielded good results, it also increases the risk of overfitting, which may reduce the overall robustness of the method. Additionally, despite the potential for lowering inference costs through sequential feature selection, the training procedure can become computationally demanding when applied to very deep neural networks or extremely large datasets. This computational complexity may limit the immediate applicability of the approach in real-time systems with strict latency requirements.

Furthermore, the range of baseline feature selection methods evaluated here, while representative, remains relatively narrow; structured or group-wise selection techniques and more advanced neural architectures were not explored in depth. This narrower scope may have constrained the discovery of potentially more effective or interpretable feature selection strategies.

Another limitation concerns interpretability. Although the step-by-step unmasking process elucidates which features drive the final ranking decision, the analysis of interpretability in this thesis is largely confined to numerical and heuristic validation rather than comprehensive user-centric evaluations. In practical deployments, real users could require more comprehensive interpretability assessments such as user studies to confirm the practical usefulness of this transparency. The potential impact of feature semantics also varies across datasets; for instance, the Yahoo! dataset does not disclose detailed feature definitions, limiting the ability to interpret or validate the method's selection choices in that scenario.

Overall, these limitations suggest that additional work is needed to broaden the applicability and deepen the evaluation of the proposed sequential feature selection framework. Enhancements in scalability, more extensive testing across diverse ranking contexts, and more rigorous interpretability evaluations may help address these shortcomings and further strengthen the method's practical utility.

6.3. Future Directions

Future research should address these limitations and explore several promising avenues for further improvement. One direction is to scale the approach to large-scale and real-time environments by optimizing the selection and masking mechanisms—such as through parallel or distributed inference—to accommodate industrial-scale datasets and meet stringent latency requirements. Another promising direction is to investigate the integration of sequential feature selection with emerging neural paradigms, such as Transformers, large language models, or multimodal architectures, which could broaden the impact and applicability of the framework, particularly for complex retrieval tasks. Finally, conducting comprehensive interpretability studies, including user studies or practitioner-focused evaluations, will be important for assessing how effectively the step-by-step unmasking process provides an intuitive

and trustworthy rationale for the model's ranking decisions.

In conclusion, this thesis has demonstrated that integrating feature selection into neural ranking models can reduce computational cost, enhance interpretability, and maintain competitive performance. Although our approach addresses several key challenges associated with traditional feature selection methods, open questions remain regarding the automation, extension, and generalization of these ideas. The results presented herein suggest that sequential feature selection offers a promising framework for building accurate, efficient, and interpretable ranking systems, and they pave the way for multiple avenues of future exploration.

References

[1] Abubakar Abid, Muhammad Fawaz Balin, and James Zou. “Concrete Autoencoders: Differentiable Feature Selection and Reconstruction”. In: *International Conference on Machine Learning (ICML)*. 2019, pp. 444–453.

[2] Suleyman Arik and Tomas Pfister. “TabNet: Attentive interpretable tabular learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2021.

[3] Christopher JC Burges. *From RankNet to LambdaRank to LambdaMART: An Overview; 2010*. Tech. rep. MSR-TR-2010-82. Available from: https://www. microsoft. com/en-us/research …, 2010.

[4] Christopher JC Burges. “Learning to rank with gradient descent”. In: *Advances in neural information processing systems* 19 (2007).

[5] Christopher JC Burges et al. “Learning to rank using gradient descent”. In: *Proceedings of the 22nd international conference on Machine learning* (2005), pp. 89–96.

[6] Zhe Cao et al. “Learning to rank: from pairwise approach to listwise approach”. In: *Proceedings of the 24th international conference on Machine learning*. 2007, pp. 129–136.

[7] Girish Chandrashekar and Ferat Sahin. “A survey on feature selection methods”. In: *Computers & electrical engineering* 40.1 (2014), pp. 16–28.

[8] Olivier Chapelle and Yi Chang. “Yahoo! Learning to Rank Challenge Overview”. In: *Journal of Machine Learning Research* 14 (2011), pp. 1–24.

[9] Jianbo Chen et al. “Learning to explain: An information-theoretic perspective on model interpretation”. In: *Proceedings of the International Conference on Machine Learning (ICML)* (2018).

[10] Jerome H Friedman. “Greedy function approximation: A gradient boosting machine”. In: *Annals of statistics* (2001), pp. 1189–1232.

[11] Jianfeng Gao, Michel Galley, and Lihong Li. “Neural Approaches to Conversational AI”. In: *Foundations and Trends in Information Retrieval* 13.2-3 (2019), pp. 127–298. doi: 10.1561/1500000074.

[12] Xiubo Geng et al. “Feature selection for ranking”. In: *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. 2007, pp. 407–414.

[13] Andrea Gigli et al. “Fast feature selection for learning to rank”. In: *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*. 2016, pp. 167–170.

[14] Jiafeng Guo et al. “A Deep Relevance Matching Model for Ad-hoc Retrieval”. In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (CIKM)*. ACM, 2016, pp. 55–64. doi: 10.1145/2983323.2983769.

[15] Isabelle Guyon and André Elisseeff. “An Introduction to Variable and Feature Selection”. In: *Journal of Machine Learning Research* 3 (2003), pp. 1157–1182.

[16] Po-Sen Huang et al. “Learning Deep Structured Semantic Models for Web Search Using Clickthrough Data”. In: *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management (CIKM)*. ACM, 2013, pp. 2333–2338. doi: 10.1145/2505515.2505665.

[17] Eric Jang, Shixiang Gu, and Ben Poole. “Categorical reparameterization with gumbel-softmax”. In: *arXiv preprint arXiv:1611.01144* (2016).

[18] Thorsten Joachims. “Optimizing search engines using clickthrough data”. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM. 2002, pp. 133–142.

[19] Ibrahim Lemhadri, Feng Ruan, and Robert J Tibshirani. “LassoNet: A Neural Network with Feature Sparsity”. In: *Journal of Machine Learning Research* (2021).

[20] Benjamin J Lengerich et al. “Learning Feature Selection Interactions”. In: *Proceedings of the 37th International Conference on Machine Learning (ICML)*. 2020, pp. 5809–5819.

[21] Ping Li. “Mcrank: Learning to rank using multiple classification and gradient boosting”. In: *Advances in Neural Information Processing Systems*. 2007, pp. 897–904.

[22] Tie-Yan Liu et al. “Learning to rank for information retrieval”. In: *Foundations and Trends® in Information Retrieval* 3.3 (2009), pp. 225–331.

[23] Lijun Lyu et al. “Is Interpretable Machine Learning Effective at Feature Selection for Neural Learning-to-Rank?” In: *European Conference on Information Retrieval*. Springer. 2024, pp. 384–402.

[24] Christopher D Manning. *An introduction to information retrieval*. 2009.

[25] Harrie Oosterhuis, Lijun Lyu, and Avishek Anand. “Local feature selection without label or feature leakage for interpretable machine learning predictions”. In: *arXiv preprint arXiv:2407.11778* (2024).

[26] Przemysław Pobrotyn and Radosław Białobrzeski. “Neuralndcg: Direct optimisation of a ranking metric via differentiable relaxation of sorting”. In: *arXiv preprint arXiv:2102.07831* (2021).

[27] Tao Qin and Tie-Yan Liu. “Introducing LETOR 4.0 Datasets”. In: *arXiv preprint arXiv:1306.2597* (2013). url: https://arxiv.org/abs/1306.2597.

[28] Tao Qin et al. “LETOR: A Benchmark Collection for Research on Learning to Rank for Information Retrieval”. In: *Information Retrieval* 13.4 (2010), pp. 346–374.

[29] Zhen Qin et al. “Are neural rankers still outperformed by gradient boosted decision trees?” In: *International Conference on Learning Representations*. 2021.

[30] Ashwini Rahangdale and Shital Raut. “Deep neural network regularization for feature selection in learning-to-rank”. In: *IEEE Access* 7 (2019), pp. 53988–54006.

[31] Ashish Vaswani et al. “Attention Is All You Need”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 5998–6008. url: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[32] Jing Xu et al. “AdaRank: A boosting algorithm for information retrieval”. In: *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. 2007, pp. 107–114.

[33] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. “INVASE: Instance-wise variable selection using neural networks”. In: *International Conference on Learning Representations (ICLR)*. 2018.