INTERPRETABLE APPROACH TO RE-RANKING SYSTEM

P. A.D.L Samarakoon

208564X

Degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics
Faculty of Information Technology

University of Moratuwa
Sri Lanka

May 2024

INTERPRETABLE APPROACH TO RE-RANKING SYSTEM

P. A.D.L Samarakoon

208564X

Thesis/Dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics
Faculty of Information Technology

University of Moratuwa
Sri Lanka

May 2024

DECLARATION

I declare that this is my own work and this thesis/dissertation does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other University or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgment is made in the text. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: Date: 2024-07-16

The above candidate has carried out research for the PhD/MPhil/Masters thesis/dissertation under my supervision. I confirm that the declaration made above by the student is true and correct.

Name of Supervisor:

Signature of the Supervisor: Date:

ACKNOWLEDGEMENT

I am deeply grateful to my supervisor, Dr. Subha Fernando, for her constant support and guidance throughout my research journey. Her timely feedback and insightful suggestions have been crucial in shaping this project and encouraging me to think critically about the subject matter.

Moreover, I am immensely grateful to my family and friends for their unwavering support and patience throughout my studies. Their genuine interest, understanding, and belief in me have been a driving force behind my success. I could not have achieved this without them.

Finally, I would like to extend my sincere gratitude to the faculty and staff of the Faculty of Information Technology's Department of Computational Mathematics for their direction and coordination.

Thank you all for your contributions to this work.

ABSTRACT

Recommendation systems plays a crucial role in the domain of Artificial Intelligence, with the expansion of internet widespread use of smart devices, the requirement to create advanced ranking and re-ranking recommendation systems have become a necessity for businesses to maintain their competitiveness. Interpretability of any machine learning system has now become a trend as well as a legal requirement in many industries that use machine learning models for analytical purposes. Recent advancements in interpretable machine learning systems including ranking systems shows significant potential in this domain. In this research, we developed an intrinsically interpretable re-ranking model based on Tensorflow Ranking Generalized Additive Model (TFR GAM) architecture to score previously recommended offers. This model was compared against an industry leading Neural Collaborative Filtering model which is a black box Model. The evaluation metrics used to compare two models were Normalized Discounted Cumulative Gain (NDCG), Hit Ratio and Precision. These were used at top 2 and top 3 which gave two variations for each metric. Even though performances of these metrics are between 40%-50% for all matrices, all metrics show that both the models performed in similar manner even though the TFR GAM model had an interpretable model architecture. Overall, in this study metrics suggest that for re-ranking tasks we can use the TFR GAM model which provides similar results to that of an industry leading NCF Model. The study also suggest future research directions to improve the model's performance, such has trying out different combinations for the number of hidden layers, activation functions, and regularization techniques.the study concludes that there is greater potential for further work in this area and continued research development could lead to significant advancement in this domain of interpretable models.

Keywords: Interpretable Models, Re-Ranking System, Recommendation Systems, Tensorflow Ranking, Learning to Rank

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
SNS	Simple Notification Service
TRCSL	Telecom Regulatory commission of Sri Lanka
RS	Recommendation System
LTR	Learn to Rank
GAM	Generalized Additive Model
NCF	Neural Collaborative Filtering
API	Application Programming Interface
AWS	Amazon Web Services
TF	Tensorflow
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
ReLU	Rectified Linear Unit
NDCG	Normalized Discounted Cumulative Gain

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Prolegomena

The expansion of the internet and the widespread use of smart devices have significantly increased traffic to websites, mobile applications,and social networking sites. These platforms have also improved the variety of data and information they gather, which can be used to determine user preferences. There have been numerous studies that have built effective recommendation systems using these data. With their success of using this information to build comprehensive and proven recommender systems and other systems like churn systems, companies in other industries like in telecommunication which also have access to a wide range of user data and information have also started building comprehensive machine learning solutions [1]. According to TRCSL, the Sri Lankan telecommunications sector generated more than LKR 245 billion in revenue in 2021 and had about 29 million mobile phone subscribers as of June 2022, a 50% increase in subscribers over the previous ten years. Sri Lankan telecommunications firms began making significant investments in analytics to create machine learning solutions as soon as the subscriber base expanded to include all market sectors in Sri Lanka. Churn Prediction and Upsell & Cross of products for business sectors like Mobile, Television, and Home Broadband are examples of such high value solutions. In Sri Lanka, mobile marketing is the primary means of communication for the telecommunications sector's product recommendations. Due to the exponential expansion of smartphones and other smart mobile devices, which will total 6 plus billion by 2022 [2], this has occurred.

Due to the limited number of product recommendations made through mobile marketing and the multiple products that will be suggested to a single subscriber during a given day, subscribers may end up not reading any messages or missing out on highly relevant product offers, which causes subscriber dissatisfaction. This

led to the requirement to re-rank for each subscriber prior to disseminating them through appropriate communication channels. These methods are still not available in the telecommunications industry, while being available for Web sites, mobile applications, and SNS platforms [3]. In this study, We have developed an interpretable Learning to Rank Re-Ranking Model that can re-rank the offers recommended from other recommendation sysems . The purpose, background, and motivation, problem in brief, suggested solution, resource requirements, and action plan make up the remainder of the proposal.

1.2 Objectives

The following objectives have been identified to develop Learning to Rank Re-Ranking models to address our research problems.

1. A critical review of Literature about recommendation systems, Learning to Rank systems and Re-Ranking Systems
2. Study of technologies adopted in implementing Learning to Rank Re-Ranking system for products recommended through recommendation systems
3. Design and Develop Re-Ranking models to predict relevance score for Re-Ranking Task
4. Evaluation of the Proposed Solution

1.3 Background and Motivation

With the origin of Recommender Systems (RSs) by mid-1990s with the introduction of Tapestry [4], the RSs have evolved with recommendation requirements of Web Sites, Mobile applications and SNS platforms. Recommender systems (RSs) help users to find new items or services like books, music or even people, based on information about the users.

The recommender systems can be divided into First-Tier, Second-Tier, and Third-Tier generation groups. First-tier recommender systems focus on e-commerce and make decisions using content based, collaborative, and hybrid filtering strategies. In order to provide precise and varied recommendations, second-tier recommender systems store user portfolios and make use of social network and social contextual information. The creation of a knowledge-based model using location-based information and additional components, such as psychological and emotional factors, is covered in the third level [5].

Earlier RSs used relevancy as the key metric when recommending products and it has evolved to building RSs which recommends products based on revenue maximization [6]. Nowadays it has evolved to optimizing both relevancy and profits when recommending products [7]. To address their business use cases, such as Churn Recovery and Next Best Option (NBO)/Next Best Action, Sri Lankan Telecommunication Companies began employing RSs and churn models a few years ago. In keeping with the development of new machine learning models, the number of offers that were suggested for a single subscriber each day increased linearly as models improved. The Churn Tribe, Upsell Tribe, and Cross Sell Tribe are the key areas or tribes from which we are receiving offers, and the business areas or units to which these offers will be associated are Mobile, Television, Home Broadband, and Home. Identifying and ranking the best offers for each customer on a daily basis was necessary due to the increase in offers that needed to be sent to each subscriber without overwhelming them, as we can see in the current environment.

The ranking issues have been explored through various approaches within the framework of supervised learning. Most algorithms essentially embody the score-and-sort strategy, which represents a higher level of abstraction. Due to the complexity of directly optimizing the ranking loss function across all possible order permutations, the score-and-sort methods have become more prevalent. The main three types of learning techniques are pointwise, pairwise and listwise

techniques[8],[9],[10].

Currently there are several Learning to Rank algorithms, like LambdaMART and Google's Tensorflow - Ranking. These methods transform ranking into a pairwise classification or regression problem [11] and use a deep learning approach to ranking [12] respectively and optimizes the prioritizing engine based on several objective functions through an iterative process using Augmented Lagrange (AL) [12]. Current research studies have also considered the interpretability aspect of LTR models, with a focus on explaining the decision made by black-box models [13],[14],[15]. However, there is a limited number of studies focused on creating inherently interpretable ranking models, where every aspect of the model is easily understandable.

Additional, previous studies have proven that re-ranking from LTR models is successful as they have demonstrated significant model performances [16],[17] and re-ranking algorithms based on mathematical programming, demonstrating that a relaxed version of the exact problem yields the same optimal solution [17] .

1.4 Problem Definition

The ultimate objective of this project is to model relevance score for re-ranking tasks by using user level features and item level features. We have defined our research problem as : “Predicting the relevance score for interpretable re-ranking tasks in the context of the telecommunication industry has been a research challenge due to heavy computational cost and performance of interpretable architecture”.

1.5 Approach of Interpretable Learning to Rank System Development

Our aim is to develop a model that is interpretable in architecture, that has similar performance as that of a black box model. The hypothesis, input, output,process, features, and users can be defined as follows.
The hypothesis of our study can be defined as : developing an interpretable ranking modEl that performs comparably to existing black box models in the telecommunications sector. The main goal is to build a machine learning model capable of re-ranking recommended offers based on input features to achieve the performance similar to that of a black box model.

For our project, the input to the model consists of previously recommended products or offers to customers, along with their responses, specifically whether they accepted the offers. Additionally, the input includes item features and context features. Item features pertain to the characteristics of the recommended products, while context features are associated with the circumstances of the individual to whom the product was recommended. Finally, the model also considers feedback features, which indicate whether the product was purchased by the customer.

The primary output of this model is a score for each offer, generated based on a set of input features, which will be used to rerank the offers on an individual basis. In the following subchapter, we will outline the comprehensive methodology employed to achieve this result.

Our process employs a Generalized Additive Model (GAM) using the TensorFlow framework, offering interpretability comparable to black box models through unique architecture developed via extensive hyperparameter tuning. We integrated diverse data sources to enhance features from a data lake, to construct a robust training dataset. The model uses TensorFlow Keras APIs and techniques like sub-model distillation to optimize individual feature impacts, combining them with calculated weights for final scoring. The model was rigorously trained, evaluated,

and benchmarked against a Neural Collaborative Filtering (NCF) model to ensure its efficacy. The main stakeholder of this system will be the campaign managers and analytical systems which will use this output to identify offers that should be communicated to customers.

1.6 System Requirement

AWS Services such as S3 and Sagemaker and Snowflake Cloud server are needed to build, process and train the desired model due to heavy data processing power and model building power.

1.7 Thesis Structure

The structure of this thesis is, Chapter 2 includes a critical literature review on interpretable Learning to Rank systems, while Chapter 3 includes the technology adopted to solve the research problem. The approach to developing the desired model is presented in Chapter 4. Chapter 5 is focused on design and Chapter 6 focused implementation. Chapter 7, where the evaluation of the developed model is discussed. The concluding Chapter 8 outlines the research's conclusions and future works, and the thesis concludes with references and appendices.

1.8 Summary

In this chapter, we discussed the problems associated with using interpretable Learning to rank systems in the context of re-ranking system in the Telecommunication industry to cater to diverse business objectives and we discussed previous studies and future challenges in the problem domain. Our

research problem is defined as "Predicting the relevance score for interpretable re-ranking tasks in the context of the telecommunication industry has been a research challenge due to heavy computational cost and performance of interpretable architecture", and we proposed our solution to address the research problem.

CHAPTER 02

LITERATURE REVIEW ON INTERPRETABLE LEARNING TO RANK SYSTEMS.

2.1 Introduction

In the preceding chapter, we provided an introductory overview of the research domain, background context, and outlined our objectives and approach to address the research problem. This chapter presents our critical literature review on Interpretable Learning to Rank (LTR) Systems, with a focus on prioritizing products and determining if an inherently interpretable architecture can be employed for re-ranking tasks, to that of a black box model. In this chapter we will discuss the literature review under three sections: early developments, latest approaches, and future trends in LTR and recommender systems in terms of their contribution, limitations and the methodology used. The chapter concludes by defining the research problem and discussing challenges and proposing next steps for Recommendation and LTR Systems.

2.2 Overview of Recommendation and Learning to Rank Systems.

Recommendation systems play essential roles in today's information retrieval and e-commerce landscapes, helping consumers shift through vast amounts of data to find what they're looking for. In a time where consumers are inundated with information, recommendation systems become invaluable tools for simplifying decision-making processes. Whether individuals are selecting a movie, shopping online, or exploring music, they rely on recommendation algorithms to navigate extensive catalogs and tailor suggestions to their particular preferences [1]. By

analyzing past behaviors, user demographics, and item attributes, recommendation systems can provide personalized recommendations that enhance user satisfaction and engagement. Additionally, these systems are valuable in decision-making by helping consumers either maximize profits [6] or minimize risk [7]. Today, recommendation systems are widely used by companies like Google, Twitter, LinkedIn, and Netflix, among others [18], [19].

Similarly, Learning to Rank (LTR) systems play an important role in optimizing the presentation of search results and recommendations to users. In the context of information retrieval, the ranking of search results directly impacts user satisfaction and the effectiveness of the search experience. By learning from relevance feedback and user interactions, Learning to Rank algorithms aim to dynamically adjust the ordering of search results to better match user intent and preferences [11]. Further, LTR techniques are employed to rank products based on their relevance to a user's query and behavior, maximizing the likelihood of conversion and customer satisfaction [7].

Over the years, these systems have experienced substantial progress, driven by the increasing demand for personalized user experiences. This literature review aims to provide an in-depth exploration of recommendation and Learning to Rank systems, encompassing their historical context, methodologies, recent advancements, challenges, and future directions.

2.3 Historical Approaches of Recommendation and Learning to Rank Systems.

The growth of the internet and the widespread use of smart devices have dramatically boosted traffic to websites, mobile applications, and social networking sites. These platforms have also expanded their collection of diverse data and

information to identify user preferences.[1]. This paved the way for the recommender systems with the first system being Tapestry in 1992 [4], which was an e-messaging system that authorized the users to rate messages as good or bad. The main limitation was a manual collaborative filtering system where users had to select their own expert annotations for inclusion.

The recommender systems can be put into three tier generations, namely First-Tier, Second-Tier and Third-Tier. First-Tier recommender systems deals with E-Commerce, where it uses content-based, collaborative based and hybrid-based filtering techniques for decision-making. Second-Tier recommender systems deal with social network and social contextual information for accurate and diverse recommendations by maintaining user portfolio. Third-Tier deals with use of location based information and other factors such as psychological and emotional factors for generating a knowledge-based model [5].

Collaborative Filtering (CF) is an information filtering model that first emmerged in the 1990s [4]. Collaborative Filtering (CF), initially developed for first-tier recommender systems, is a model that builds a user's preference database using their evaluation data to predict items that match their taste and then uses this information for recommendations. CF works better when users are looking for specific information than when they are browsing out of general interest which shows CF systems are domain dependent [20]. Content-Based Filtering for recommendation which was begun in First-Tier recommender systems that help users find information by providing them with personalized suggestions.

Content-based filtering differs from information retrieval in how it represents a user's interests. Instead of relying on a query, an information filtering system attempts to model the user's long-term interests. These models utilize relevance feedback methods to learn a user model, as this approach is both efficient and dynamic. [21]

Numerous strategies have been investigated to address the ranking issue within the

context of supervised learning. Most algorithms essentially implement what is known as the score-and-sort approach. This method is widely used because optimizing the ranking loss function directly across all possible ordering permutations is often challenging.

There are essentially three types of learning methods: pointwise methods (regression), pairwise methods, and listwise methods. In the pointwise approach, a target score is assigned to each item in a set, and function **f** is determined using regression techniques to predict this score directly. The loss is typically calculated by the difference between the anticipated and actual scores.. The benefit of using regression-based methods lies in leveraging the extensive research and development in regression for ranking purposes. However, a major disadvantage is that forecasting an precise score for each item can be more difficult than merely ranking them.. Regression-based methods are most effective when there is a reliable scoring function available [8].

The goal of pairwise preference methods is to learn a function f such that f(xi (l)) > f(x(l)j) if xl i > xl j. Even if the labels {yl } are given as total orderings, Despite this, the pairwise preference methods would still derive and learn from the associated set of pairwise ordering. The pairwise preference approach offers several advantages, such as its compatibility with partial orderings and its applicability in scenarios where obtaining total ordering labels is difficult. It can also be adapted to work with existing classification systems with some modifications. However, a major limitation is that its learning objective is primarily aimed at reducing the number of incorrect pairwise orderings, which does not perfectly align with the true loss function for total orderings. Additional challenges include the assumption that pairs are independent and identically distributed, as well as the computational burden of evaluating all pairwise preferences. Nevertheless, these problems can be reduced to some degree by giving different weights to specific pairs [9].

List-wise approaches form the third category of ranking algorithms, distinguished by their focus on optimizing the entire list as the main objective. This addresses the

limitations of pairwise and regression methods, which often incorrectly assume independence among items within the same list. List-wise methods are generally divided into two types. The first type involves direct optimization of the desired loss function, or a smooth approximation of it, which can be challenging due to the non-smooth and non-differentiable characteristics of many ranking loss functions. The second type uses a loss function defined over the list, though this loss function may not directly correspond to the evaluation metrics used. [10].

Although research on interpretable machine learning has been ongoing since the 1990s [17], there's still no consensus on its definition or interpretation methods. Despite this, efforts have been made to clarify the concept within artificial intelligence [22]. Interpretable machine learning aims to create models that are accurate, transparent, and understandable to humans, facilitating the identification of cause-and-effect relationships in their inputs and outputs [14]. Learning-to-rank (LTR) models, known for their complexity, use many parameters to achieve accuracy. However, this complexity can make it hard for humans to understand why certain items are ranked as they are [23]. These "black box" models lack transparency, making it tough to see what factors influence their decisions. This opacity can lead to errors, biases, and ethical issues. Therefore, there's a growing need for interpretable LTR models [24].

2.4 A Review of Current Systems of Recommendation and Learning to Rank Systems.

In recent years, deep neural networks have delivered impressive achievements in fields like speech recognition, computer vision, and natural language processing. The application of deep neural networks to recommender systems has also begun to

gain interest. A comprehensive approach known as Neural Collaborative Filtering (NCF) has been introduced, which allows for replacing the traditional inner product with a neural architecture capable of learning any function from given inputs [25]. NCF is a general framework that can express and generalize matrix 10 factorization. This model suggests using a multi-layer perceptron to train the user-item interaction function to boost NCF modeling with non-linearities.

Recent innovations in recommender systems include hybrid recommender systems, which integrate the strengths of two or more recommender systems. Systems exist that combine product ratings, features, and demographic data to provide hybrid recommender systems. These systems offer a solution to pure content-based recommender systems, which produce recommendations of higher quality than those produced by pure content-based and demographic recommender systems [26].

Neural network tools like Google's Tensorflow - Ranking are recent additions to the Learning to Rank algorithm repertoire.This methods optimize the prioritizing engine based on several objective functions through an iterative process using Augmented Lagrange (AL). Large-scale search, recommender systems, document summarization, and question-answering are just a few of the useful applications of these. These algorithms have excellent training and inference scalability. The efficiency of this library in teaching ranking functions for extensive search and recommendation applications in Gmail and Google Drive has been rigorously demonstrated by Pasumarthi (2019) in his research.

Furthermore, when it comes to re-ranking from LTR models, Wang et al (2014), in his research, he introduced a re-ranking technique using learning-to-rank methods to improve the accuracy of landmark-based audio fingerprinting (AFP) for music retrieval. The re-ranking process is triggered when the confidence measure of the returned ranking from the AFP system is insufficient. The study suggests the incorporation of new features for re-ranking and utilizes learning-to-rank paradigms, such as pairwise and listwise approaches, to model query behavior

towards desired rankings. Experimental findings of the research demonstrate that this re-ranking method significantly enhances performance with minimal additional overhead on overall response time.

Another study by Rudin et al. (2018) highlights the effectiveness of learning-to-rank techniques for prioritization tasks, where items are ranked based on their estimated probabilities to allocate resources efficiently. However, their study has identified a significant issue with current learning-to-rank algorithms at that time, which rely on convex proxies that result in suboptimal approximations. In their study they have identified reranking algorithms based on mathematical programming, demonstrate through empirical analysis that a relaxed version of the exact problem yields the same optimal solution.

While many studies strive to improve the accuracy of ranking models, fewer address the issue of model interpretability. However, lacking interpretability can lead to practical challenges such as troubleshooting difficulties, susceptibility to biases , and increased maintenance costs [28]. Recent research has seen a surge in interest in interpretable learning-to-rank models, with a focus on explaining the decisions made by black-box models. Singh (2018) developed a system to quantify and visualize the impact of different terms on a decision in ranking models, aiming to make the ranking process more transparent.Verma et al (2019) also tackled similar challenges, aiming to quantify the contribution of terms in ranking models and identify potential shortcomings in existing systems. However, there is a limited number of studies focused on creating inherently interpretable ranking models, where every aspect of the model is easily understandable.

Currently, generalized additive models (GAMs) lead the way in making learning-to-rank models interpretable. In their study, Zhuang et al. (2019) expanded GAMs into ranking models and introduced a new formulation. They introduced neural ranking GAMs, which integrate neural networks with list-level context features. Their experiments showed that ranking GAMs outperformed traditional

ones while remaining interpretable. As a result, ranking GAMs have become the go-to method for LTR tasks due to their transparent and self-explanatory structures.

2.5 Challenges in Recommendation and Learning to Rank Systems.

Despite significant progress, recommendation systems face challenges such as data sparsity, cold-start problems, and algorithmic fairness. Data sparsity occurs when there are not enough user-item interactions to generate accurate recommendations, especially for new or niche items. Cold-start problems arise when there is limited information about new users or items, making it difficult to provide personalized recommendations. Algorithmic fairness concerns ensuring equitable treatment and representation of diverse user demographics in recommendation outcomes, mitigating biases and discrimination. [26].

Interpretability is crucial in Learning to Rank systems, offering insights into ranking decisions and aiding domain experts in understanding model behavior. It helps stakeholders detect biases or errors in rankings, guiding adjustments or interventions [15].

Furthermore, Scalability poses a significant challenge for Learning to Rank systems, particularly in handling large-scale datasets common in real-world applications [8]. As the volume of data continues to grow exponentially, traditional ranking algorithms may struggle to efficiently process and analyze massive datasets within acceptable time frames. Scalability concerns encompass not only computational efficiency but also memory management and parallelization strategies to optimize performance across distributed computing frameworks [15].

In conclusion, recommendation systems and LTR systems are essential components of modern information retrieval. However, they present several

challenges, as mentioned. By tackling these challenges and advancing the state-of-the-art in Learning to Rank, future research can open up new opportunities to enhance the interpretability, effectiveness, efficiency, and fairness of ranking algorithms in real-world applications.

2.6 Future and Next Steps of Recommendation and Learning to Rank Systems.

The evolution of recommendation and Learning to Rank (LTR) systems has opened a transformative era for information discovery and personalized content delivery, presenting a wide range of opportunities for future development. Thus, Future research in recommendation systems should prioritize exploring novel algorithms, enhancing user experience, and addressing ethical considerations [30]. Advanced techniques like reinforcement learning and graph neural networks offer promising avenues for improving recommendation accuracy and adaptability. Reinforcement learning enables systems to learn optimal decision-making policies through user interaction, while graph neural networks enhance representation learning by capturing complex relationships between users and items [1].

Improving user experience in recommendation systems is crucial, as personalized and explainable recommendations empower users to understand and manage their preferences effectively. The field of explainable recommendation systems is evolving dynamically, with ongoing exploration of novel approaches to enhance interpretability and effectiveness. One promising avenue for future research involves developing hybrid approaches that aggregate various explanation techniques, such as textual and visual explanations, to provide users with comprehensive and user-friendly insights into recommendations [31].

Another direction for future research involves integrating user feedback into the recommendation process. Enabling users to provide feedback on recommendations

can enhance the relevance and accuracy of future suggestions. Further, future trends in Learning to Rank (LTR) systems include the adoption of real-time applications. These applications feature adaptive algorithms that continuously update ranking models based on dynamic user interactions and evolving contexts, thus delivering timely and personalized recommendations [32].

2.7 Summary of the Literature Analysis

Following a comprehensive review on the problem domain, concerning the Interpretable Ranking Approach to Product Ranking System, we have compiled the key findings from various studies into Table 2.1. This table provides an overview of the advantages, limitations and technology used in each research.

Table 2.1 – Analysis of Literature Review

Research	Advantages	Limitations	Technology Used
D. Goldberg et al (1992) [7]	1.Use Collaborative Filtering 2.Use Content Based Filtering	Manual Collaborative Filtering and Content Based Filtering	Rule Based
R. van Meteren et al (2010) [10]	1.Does not require a similarity index between users. 2.Enough information to avoid cold start most of the time 3.Transparent behaviour	1. Insufficient diversity and novelty 2. Require lot of domain knowledge 3. Bounded Content Analysis	ML - PRES
W. S. Cooper et al (1992) [11]	1.The extensive research on regression can be effectively utilized for ranking purposes.	1. Scalability 2. Speed	Rule Base - Statistica
Freund et al (2006) [12]	1.Easy to apply to existing models	1. Low Accuracy 2. Cold Start	ML - Boosting

	2.Works on partial orderings	Problem	
F. Xia et al (2007) [13]	1.Better performance relative to other models available at that time	1. Scalability 2. Low Speed	Rule Base - Statistic
Xiangnan He et al (2017) [19]	1..Use generalized matrix factorization 2.Scalable 3.Speed	1. Resource intense 2. High Cost	Neural Network
M. A. Ghazanfar et al (2010) [20]	1.Handles Quality of Recommendation 2.Handles Cold Start Problem	1. Resource intense 2. High Cost	Statistical Optimizatio n
R. K. Pasumarthi et al (2019) [21]	1.Highly Scalable 2.High Speed 3.Highly Accurate	1. Resource intense 2. High Cost	Neural Network
M. Momma et al (2019) [31]	1.High Accuracy 2.Can accommodate multiple objective function	1. Low Speed 2. Use memory when objective functions increase	ML - Optimizatio n
Zhuang et al.(2021) [27]	1. intrinsically interpretable 2. Improved performance	1. Accuracy 2. Scalability	GAM based on Neural Network

2.8 Research Problem

Through our literature review, we've discovered a notable gap in interpretable learning to rank systems with similar performance to that of black box ranking models, particularly in the telecommunications sector. There's a scarcity of publicly available research in this specific area. Thus, our research objective is to fill this void by creating a customized interpretable Learning to Rank system tailored to the needs of a Quad Play telecommunications company, covering Mobile, TV, HBB,

and Home services. Moreover, we recognize the complexity posed by the dynamic nature of business objectives and shifting subscriber preferences. This dynamic environment makes it difficult to effectively rank products from multiple recommender systems within the telecommunications industry, especially considering the various objective functions involved.

2.9 Summary

This section provides a comprehensive analysis of the literature on Recommendation and Learning to Rank Systems, covering research history, achievements, current systems, challenges, and future directions. Furthermore, we identified the specific research issue as the need for an Interpretable Ranking Approach to Product Ranking Systems rather than relying on a black box model. Chapter 3 expands on this by thoroughly examining the technology utilized to address the research challenge.

CHAPTER 03

TECHNOLOGIES ADOPTED FOR LEARNING TO RANK AND RECOMMENDATION SYSTEMS.

3.1 Introduction

In the last chapter, we conducted an extensive literature review focusing on recommendation systems, LTR systems, and interpretability. This chapter focuses on the technologies employed in our research problem. It is structured into several sub-sections. The first section offers a broad overview of the technologies relevant to our research problem, while subsequent sections delve deeper into the specific technologies employed in each key aspect of our modeling approach.

3.2 Overview of the Technologies Used in Recommendation and Learning to Rank Systems.

The primary objective of our modeling approach is to utilize item and user features to predict relevance scores for performing item re-ranking tasks. This involved developing and applying two deep learning models aimed at assessing whether an intrinsically interpretable architecture could yield performances similar to those of a black box model. Our implementation encompasses two primary steps: an analytics part involving feature extraction using analytical technologies and an ML modeling part utilizing deep learning technologies to integrate a recommendation system and Learning to Rank (LTR) task, implementing two distinct models. By drawing inspirations from existing researches, two models were developed for the re-ranking task; Neural Collaborative Filtering model (NCF Model) and a Tensorflow Ranking Generalised Additive Model (TFR GAM Model) , utilizing TensorFlow Keras APIs. Furthermore, evaluations were conducted using custom functions tailored to calculate the performance metrics of these models namely NDCG, Hit Ratio and Precision. Detailed insights of technologies adopted in these model

developments and deployment enhancements using cloud solutions and ML model technologies are elaborated in subsequent subsections.

3.3 Machine Learning Technologies Used in Recommendation and Learning to Rank Systems.

3.3.1 Introduction to Machine Learning Technology.

In general, Machine Learning (ML) encompasses a diverse array of models and techniques aimed at extracting patterns and insights from data. These models include traditional algorithms such as linear regression and decision trees, as well as advanced methods like neural networks and ensemble models. In the context of recommendation systems and LTR, machine learning plays a crucial role over rule-based approaches by automatically adapting to changing data patterns and user behaviors. This flexibility has allowed for more accurate and dynamic ranking decisions, leading to improved user satisfaction and engagement. Furthermore, the successful implementation of ML-based LTR models can have significant business impacts, such as increased customer retention, higher conversion rates, and enhanced revenue generation.

3.3.2 Introduction to Deep Learning Technology.

Deep learning is a subset of machine learning that involves training artificial neural networks with multiple layers to learn hierarchical representations of data. Unlike traditional machine learning methods, deep learning algorithms can automatically discover and extract intricate patterns and features from large and complex datasets.Its widespread applications range from image and speech recognition to medical diagnosis, autonomous driving, and recommendation systems. With its

ability to handle vast amounts of unstructured data and achieve state-of-the-art performance in various tasks, deep learning continues to drive advancements in artificial intelligence.

In recommendation systems, deep learning models utilize rich user-item interaction data to learn latent representations, capturing nuanced relationships between users and items for more accurate predictions. Techniques like convolutional neural networks (CNNs) and recurrent neural networks (RNNs) incorporate contextual information, such as temporal dynamics and sequential patterns, improving the system's ability to capture evolving user preferences and offer timely recommendations.

3.3.3 Introduction to Recommendation systems and Learning to Rank.

Recommendation systems and Learning to Rank (LTR) are fundamental concepts in information retrieval and personalized user experiences. Recommendation systems aim to provide users with relevant and personalized content, products, or services based on their preferences and behavior. These systems utilize various techniques such as collaborative filtering, content-based filtering, hybrid approaches, matrix factorization, and deep learning to analyze user behavior and item characteristics and to make recommendations. Collaborative filtering techniques compare user preferences with those of similar users to generate recommendations, while content-based filtering recommends items based on their attributes and similarities to items previously liked by the user. Matrix factorization methods decompose the user-item interaction matrix to uncover latent factors that influence preferences. Deep learning algorithms, including neural collaborative filtering and deep content-based models, learn complex patterns from user-item interaction data for more accurate recommendations.

On the other hand, the Learning to Rank (LTR) approach incorporates techniques

from machine learning, deep learning, and natural language processing (NLP). It utilizes machine learning algorithms like support vector machines (SVM), gradient boosting machines (GBM), and random forests to develop ranking models based on features derived from user interactions and item characteristics. Deep learning models, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs), are used to extract hierarchical representations of data, capturing complex patterns in user behavior and item significance. Furthermore, NLP methods are applied to analyze textual data, such as user queries and document content, enhancing the accuracy and personalization of results by better understanding user intent and contextual relevance in ranking tasks.

3.3.4 Tensorflow Keras API

The TensorFlow Keras API is a high-level interface for neural networks, designed to facilitate the creation, training, and deployment of deep learning models. Keras offers a straightforward interface that enables developers to rapidly design and test different neural network structures and settings. It includes a wide array of components for constructing various kinds of neural networks, such as multi-layer perceptrons (MLPs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs). TensorFlow, a comprehensive open-source machine learning library, provides robust integration with Keras, ensuring efficient deep learning computations across CPUs, GPUs, and TPUs. By simplifying complex low-level details, the TensorFlow Keras API is accessible to both novices and experts, making it a favored tool for developing deep learning solutions in numerous fields due to its versatility, user-friendliness, and thorough documentation.

3.3.5 Neural Collaborative Filtering Model

Neural Collaborative Filtering (NCF) is a deep learning recommendation algorithm that leverages neural networks to learn latent representations of users and items. Unlike traditional collaborative filtering methods, which rely on matrix factorization techniques to model user-item interactions, NCF models directly learn user and item embeddings using neural networks. These embeddings capture complex relationships between users and items, allowing the model to make accurate predictions about user preferences.

The NCF model architecture comprises three primary components: the user network, the item network, and the wrapper network as shown in Figure 3.1.

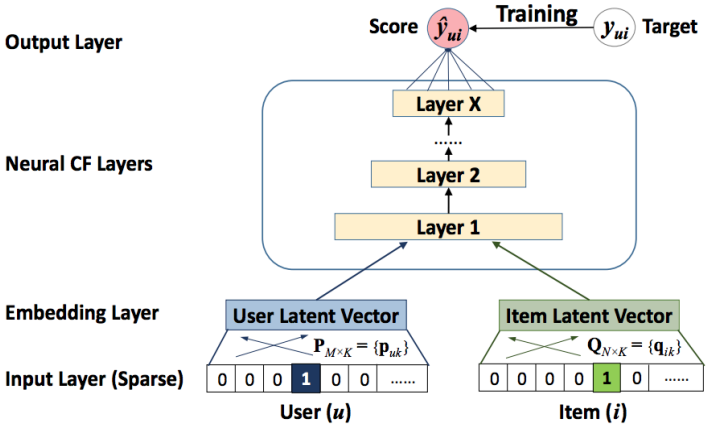

Figure 3.1: NCF Architecture

1. User Network: The user network is responsible for processing user-related data and learning the latent representation of users. It typically takes user features or

user-item interaction data as input and passes them through a series of layers, such as fully connected layers or embedding layers. The user network learns to extract meaningful features from the input data that represent the preferences and characteristics of individual users.

2. Item Network: The item network performs a similar function to the user network but focuses on processing item-related data. It takes item features or item-user interaction data as input and learns the latent representation of items. Like the user network, the item network consists of layers that transform the input data into a meaningful representation that captures the characteristics of each item.

3. Wrapper Network: The wrapper network combines the outputs of the user and item networks to generate predictions or recommendations. It takes the learned user and item representations as input and applies a final layer or set of layers of densely connected neurons, with each layer learning increasingly abstract representations of the input data. Through the training process, the neural network adjusts the parameters of the model to minimize the prediction error, optimizing the embeddings to accurately predict user-item interactions.

3.3.6 TF-Ranking GAM Model

TensorFlow Ranking (TFRanking) is a powerful framework developed by Google for building and training large-scale learning-to-rank models using TensorFlow. TFRanking provides a flexible and scalable solution for ranking tasks, offering support for various ranking algorithms and evaluation metrics.

One key algorithm supported by Tensorflow Ranking is the Generalized Additive Model (GAM), which can be used in learning-to-rank tasks. The GAM model architecture consists of several components designed to capture complex relationships between input features and relevance scores.

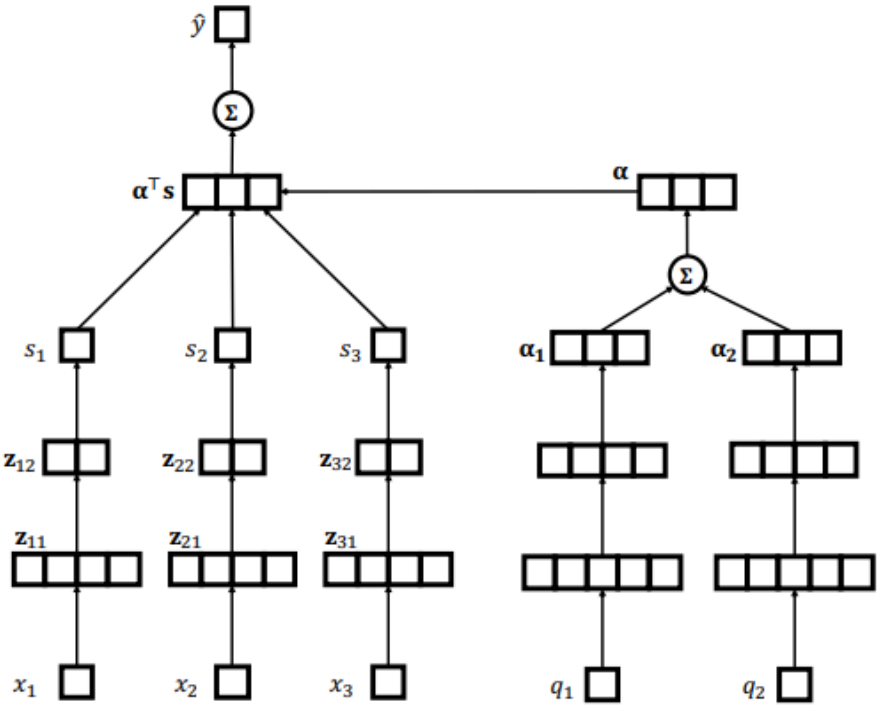

Figure 3.2: TFR GAM Architecture

The main components of the TFRanking GAM model architecture shown in Figure 3.2 are as follows:

1. Context Network: The context network processes input features representing the context of the ranking scenario, such as user demographics or query information. It extracts relevant features from the context and encodes them into a latent representation.

2. Item Network: The item network processes input features representing the attributes of the items being ranked, such as product characteristics or content features. Similar to the context network, it extracts meaningful features from the item data and encodes them into a latent representation.

3. Combining Network: The combining network integrates the representations learned by the context and item networks to produce the final ranking scores. It applies a series of transformations to combine the context and item features effectively, capturing the interactions between them.

3.4 Cloud Architecture

For our study, we will leverage cloud architecture to implement recommendation and learning-to-rank (LTR) systems, utilizing services provided by AWS and Snowflake. AWS offers a comprehensive suite of cloud computing services, including storage, compute, database, and machine learning capabilities, making it an ideal platform for hosting and scaling recommendation and LTR models. We will utilize AWS services such as Amazon S3 for data storage, Amazon EC2 for virtual servers, and Amazon SageMaker for building, training, and deploying the models. Additionally, Snowflake provides a cloud-based data warehousing solution that enables efficient and scalable data processing and analysis. By integrating AWS and Snowflake services, we can create a robust and scalable cloud architecture for implementing recommendation and LTR systems, allowing for seamless data processing, model training, and deployment.

3.5 Model Evaluation Methods.

In the evaluation stage of the two models NCF Model and TFR GAM model, 3

performance indicators were used to assess their perfromances. NDCG provides a measure of the ranking quality of the re-ranking system by weighting the relevance of each item according to its position in the re-ranked list. Higher values are given to items at the top of the list, reflecting the greater importance of ranking highly relevant items more prominently.

Hit Ratio used to assess the presence of at least one relevant item in the re-rank list was purchased before to check the relevance of the re-ranking output. This metric is useful for evaluating the overall recall capability of the re-ranking system. Precision in the context of a re-ranking system quantifies the accuracy of the re-ranking at top N by measuring the fraction of top N re-ranked items that are relevant to the user.

3.6 Summary

This chapter has outlined the technology utilized to address the research problem. Various techniques and technologies utilized in the study have been discussed in different subsections throughout the modeling process. The subsequent chapter will explore the approach adopted in our project.

CHAPTER 04

APPROACH

4.1 Introduction

In the third chapter, a comprehensive examination of technologies adopted to solve our research problem was discussed. Following is a detailed description of our approach for developing and applying two deep learning models aimed at assessing whether an intrinsically interpretable architecture could yield performances similar to those of a black box model.

This chapter is divided into six primary sections, namely hypothesis, input, output process, features and users of the solution, respectively. The hypothesis section provides a comprehensive explanation on the underlying principles on which the ranking model is based. The input section outlines the data inputs used in the model, whereas the output section outlines the expected model outputs. The process section describes the step-by-step procedure that is followed to generate the required output. The feature section outlines the model's essential characteristics. Finally, the users of the solution section describe the solution's intended audience and how they will benefit from this.

4.2 Hypothesis

The core hypothesis of our research project is that a new interpretable ranking model can be developed with performance comparable to that of black-box ranking models in the telecommunications industry. The ultimate objective of this project is to create a machine learning model that can re rank the already recommended products based on input features which has similar performance of that of black box model.

Recent studies have demonstrated that it is possible to create re-ranking models

with intrinsical interpretability using Generalized Additive Model (GAM) principles. These studies inspired the development of this hypothesis. Such findings are significant given the rising requirement on interpretability in machine learning models through new world policies on use of AI.

If the desired model with acceptable performance is developed in the context of the telecommunication industry, it can be directly used for re-ranking tasks in hyper personalized systems to improve customer experience and stickiness. This will also assist in achieving interpretability on AI systems which has now become a necessity on machine learning systems.

4.3 Input

The model input for our project is past recommended products/offers sent to customers and their feedback in terms of takeup aligned with the item and context features at the time of recommendation. These features mainly divided into three categories, firstly item features which are the features related to the product recommended, second set is the context features which are related to the context of the person to whom the product is recommend to by the system and finally the third on is the feedback feature in term of takeup where person has purchased the product or not.

In the following subsection, we will provide a more thorough explanation of the dataset used in our analysis, as well as description of how the data set was created.

4.3.1 Dataset Description

The dataset was created by considering past six months offers and take up data, below table 4.1 contains a description of features used in the model.

Table 4.1 : Feature Description

Feature Name	Feature Type	Description
AON	Context	Duration of the connection
Age	Context	Age of the person
NPS	Context	Persons Experience Indicator with the company
Last Offer Response	Context	Whether person has purchased the last offer sent
Churn Propensity	Context	Persons current churn propensity
Value	Item	Value of the Product
Product Validity	Item	Products validity after activating
RFM Score	Item	Products RFM score considering past 180 days
Relevance Score	Item	Relevance score from recommendation model
Take Up	Label	Whether person has purchased the offer or not

4.4 Output

The objective of our study is to create a model that can re-rank the offers recommended from other recommendation models. The key output from the model is the score for each offer that will predict from the re-ranking model for the given set of input features in order to re-rank the offers person wise. In the subsequent subchapter, we describe the detailed methodology used to generate the desired

outcome.

4.5 Process

Our modeling approach uses the Generalized Additive Model (GAM) approach with tensorflow framework. Our model architecture is unique from existing architectures which were identified through a series of hyper parameter tuning experiments. This unique architecture provides similar results with respect to black box models but with added interpretability.

Our model process began by applying various data engineering tasks to build the data model, which can be used to train the model. First, the data was collected from various data sources on recommended products/offers from two tribes, namely the upsell and churn tribes models, delivered via SMS, along with their feedback/takeup over a six-month period from the mobile-prepaid business unit, encompassing a total of 1.3 million subscribers. Customer context features such as Age On Network (AON), Age, and Net Promoter Score (NPS) were created using subscriber data accessible on the data lake. Additionally, features such as relevance score, churn propensity scores, and product information were obtained from tribe model outputs. Since RFM scores related to each product of each customer were not available directly in the data sources, those features were created using past 180 days of product activations. Product codes, validities, and prices were directly extracted from the product catalogs. Lastly, the takeup related features were calculated based on whether the subscriber purchased the product or not within the given offer period, considering total number of times the offer is sent and takeups for the past 6 months for the relevant product and forming the label accordingly.

After processing the data to create the ideal data model for training and evaluation, we developed the TFR GAM model and NCF Model architecture using tensorflow keras API's. We followed an approach known as sub-model distillation for TFR GAM Model which is to train smaller,simpler models by minimizing loss between

its outputs and that of a more larger,complex model.In our approach we distill each sub-model individually for all the context and item features. The final output is the weighted sum of the item and context feature outputs where weights were calculated using context features.Each sub-model layer also includes batch normalization and dropout layers for regularization purposes. Additionally, we defined several hyperparameters and activation functions to match models' behavior. Then we compiled and trained the model and evaluated its performance. For the NCF Model, we followed the NCF models suggested architecture and implemented that architecture to train and evaluate this black box model against a more interpretable TFR GAM Model.

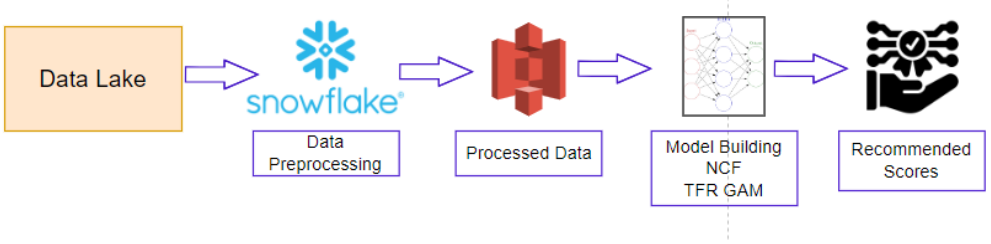

Figure 4.1: Process Design Diagram

We evaluated the model using two main performance indicators namely Normalized Discounted Cumulative Gain (NDCG), Hit ratio and a subsequent performance indicator Precision.

4.6 Features

Following are the features that were identified in th re-ranking system

- ML Module to predict the score to be used for re-ranking of products
- Intrinsically interpretable architecture that can be used to enterprise the models outcomes

4.7 Users

The main stakeholder of this system will be the campaign managers and analytical systems which will use this output to identify offers that should be communicated to customers.

4.8 Summary

We presented our approach under six distinct topics, which includes hypothesis, input, output, process, features and users. Specifically, under the process subchapter, we provided a thorough examination of data model creation, model architecture and model evaluation methods considered in the study. In the forthcoming chapter, we will delve into the design of our study approach.

CHAPTER 05

DESIGN

5.1 Introduction

In this chapter 4, we presented the approach of our study, which involves creating a GAM architecture based re-ranking model. In this chapter, we will provide a more in-depth discussion of the design of our solution. We will mainly focus on data preprocessing, model architecture in detail and model evaluation in this chapter.

5.2 Data

The dataset for this study is sourced from the outputs of churn and upsell offer recommendation models, which were communicated to subscribers via SMS. Spanning a period of six months, the raw dataset captures subscriber base within the mobile-prepaid business unit.

For the initial data set, we included customer context features such as Age on Network (AON), Age,churn propensity scores and Net Promoter Score (NPS). These features were created using subscriber date of join, NIC and prediction models data.

For the dataset, RFM score is also added as a marketing metric which provides a score to the customers based on their purchasing behavior. We use historical product activations data spanning the preceding 180 days to calculate following scores, Recency score which represent the time elapsed since a customer's last purchase, Frequency score which represent the total number of activations within the specified timeframe, and Monetary score which represent cumulative customer spend. Then the values obtained for R,F,M components will be mapped into a scoring system from 0-5. The combination of all three scores will produce the final

RFM score for each customer-product combination.

Furthermore, product-related information like product codes, validities and product prices were directly extracted from the product catalog to ensure accuracy and consistency. Finally, we have included two takeup related attributes, last offer response which represents whether the subscriber has taken a previous offer within a given time period as a customer context feature. Takeup label which is the target variable for our recommendation modeling task indicating whether subscribers purchased recommended products within the offer period, based on historical uptake data for relevant offers over the past six months. Further offer relevance from the recommendation model was used as an item feature.

To enable consistent training of data by algorithm, data was padded to stanrdadize the input sizes. This ensures efficient computation and helps handle variable-sized user interaction data effectively.

Overall, the data preprocessing stage involved gathering data from various sources, engineering features to understand customer context, and creating new metrics to fill data gaps. These tasks were performed using the Snowflake platform. This enabled us to seamlessly integrate diverse datasets and effectively manage the data throughout the preprocessing stage.

TABLE 5.1: Input Data format after snowflake preprocessing

	V_ENCR_INT	AGE	AON	NPS	CATEGORY_CODE	RELAVANCE_SCORE	RFM_SCORE	LAST_OFFER_RESPONSE	CHURN_PROPENSITY	VALUE	PRODUCT_VALIDITY	TAKEUP_PERC
8	399870924	28	39	3	25	0.65	4	0	0.28	81.0	14	20
9	421419114	30	51	5	25	0.43	9	0	0.38	81.0	14	20
10	202397596	62	8	5	25	0.49	5	0	0.23	81.0	14	20
11	152943554	25	54	3	7	0.49	9	0	0.11	649.94	1	20
12	743995581	51	245	2	7	0.5	4	1	0.33	649.94	1	20
13	568785925	65	44	4	4	0.43	6	0	0.26	505.51	1	20
14	858712925	21	52	3	4	0.49	5	1	0.35	505.51	1	20

5.3 Model Architecture

In the conducted study, we developed and applied two deep learning models, with the aim of identifying whether an intrinsically interpretable architecture can be used for re-ranking tasks which gives similar performances to that of a black box model. First model was the Neural Collaborative Filtering model (NCF Model) which was developed using tensorflow keras API's [25]. This model accepts the input training data and provides the relevance score as the output. Second model was the Tensorflow Ranking Generalized Additive Model (TFR GAM Model) which was also developed using tensorflow keras API's [15]. For this model also the same input training data were used and the output was the relevance score.

Both models, namely NCF Model and TFR GAM Model, the input training data were used to train the models to get the output, the relevance score. The hyperparameters of the models are defined first, which includes dropouts regularizer, normalization regularizers, number of hidden layers, size of input and output dimensions, activation functions and the architecture of the model on how the nodes are connected to form the model.

The NCF model architecture consists of three main components: the user network, item network and the wrapper network. The user network takes all the user level features to the model like Age, AON etc, the item network takes all the item or offer related features to the model like Offer Validity, Offer Value etc and finally the wrapper network concatenates both user network and item network to complete the model. The user network input layer has the same shape as the user features, this is followed by two dense layers with ReLU activation function and dropout layer. The item network input layer has the same shape as the item features, this is followed by three dense layers with ReLU activation and two dropout layers. Masking layer was used in both Item network and User Network after input layer to ignore padded input data. This helps the model to focus on relevant user interactions and improve

the accuracy by not learning from irrelevant data.The wrapper network concatenates both the item and user networks and is followed by two dense layers and two dropout layers. The output layer is a single node with sigmoid activation function which predicts the final relevance score. Table 5.2 shows a summarized architecture of the model.

The TFR GAM Model, the model architecture, consists of three main components: context network,item network and combining network. For the item network, individual sub-models are constructed for each input. Each sub-model starts with a dense layer with ReLu activation function, followed by a dropout layer and normalization layer. This is followed by another dense layer, dropout layer and a normalization layer. The output of each submodel is a single node without any activation function.(Linear Transformation of the input data). Parallel to item network, context network also consist of sub-models for each input a similar to item network and similar to item network similar dense, dropout and normalization layers are constructed. Context networks differ in its layer, which is the output layer which outputs weights for each item feature using softmax activation function, ensuring that output weights across item features are sum to one. The last component of the architecture which is the combining network, for each item feature, the corresponding weights from all context sub-models are summed and then multiplied with item-sub score to yield weighted scores. These weights scores across all item features are then summed to form a preliminary final score. Then the combined final score is passed through a dense layer with sigmoid activation function to predict the final relevance score.

Table 5.2: NCF MODEL ARCHITECTURE

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 4)]	0	[]
dense_2 (Dense)	(None, 16)	80	['input_2[0][0]']
input_1 (InputLayer)	[(None, 5)]	0	[]
dropout_1 (Dropout)	(None, 16)	0	['dense_2[0][0]']
dense (Dense)	(None, 128)	768	['input_1[0][0]']
dense_3 (Dense)	(None, 10)	170	['dropout_1[0][0]']
dropout (Dropout)	(None, 128)	0	['dense[0][0]']
dropout_2 (Dropout)	(None, 10)	0	['dense_3[0][0]']
dense_1 (Dense)	(None, 32)	4128	['dropout[0][0]']
dense_4 (Dense)	(None, 4)	44	['dropout_2[0][0]']
concatenate (Concatenate)	(None, 36)	0	['dense_1[0][0]', 'dense_4[0][0]']
dense_5 (Dense)	(None, 64)	2368	['concatenate[0][0]']
dropout_3 (Dropout)	(None, 64)	0	['dense_5[0][0]']
dense_6 (Dense)	(None, 32)	2080	['dropout_3[0][0]']
dropout_4 (Dropout)	(None, 32)	0	['dense_6[0][0]']
dense_7 (Dense)	(None, 1)	33	['dropout_4[0][0]']

Total params: 9671 (37.78 KB)
Trainable params: 9671 (37.78 KB)
Non-trainable params: 0 (0.00 Byte)

Table 5.3: TFR GAM MODEL ARCHITECTURE

Layer (type)	Output Shape	Param #	Connected to
AGE (InputLayer)	[(None, 1)]	0	[]
AON (InputLayer)	[(None, 1)]	0	[]
NPS (InputLayer)	[(None, 1)]	0	[]
LAST_OFFER_RESPONSE_CODES (InputLayer)	[(None, 1)]	0	[]
CHURN_PROPENSITY (InputLayer)	[(None, 1)]	0	[]
masking_13 (Masking)	(None, 1)	0	['AGE[0][0]']
masking_14 (Masking)	(None, 1)	0	['AON[0][0]']
masking_15 (Masking)	(None, 1)	0	['NPS[0][0]']
masking_16 (Masking)	(None, 1)	0	['LAST_OFFER_RESPONSE_CODES[0][0]']

Full architecture can be found in the Appendix C.

5.4 Model Evaluation

After training of both models, three main evaluation methods were used to assess their performnces. The first performance measure is Normalized Discounted Cumulative Gain. (NDCG). This indicator is used to measure the quality of the ranking results with respect to their relevance. It emphasizes placing highly relevant items at the top of the ranking list by penalizing the ranking of relevant items based on their position, thus valuing the correct ordering of items rather than just presence in the list. Figure 5.1 shows the formula of NDCG and the range of NDCG is from 0 to 1. The second performance measure is Hit Ratio which measures the proportion of top ranked items for which at least one relevant item is taken up in the past. It effectively quantifies the ability of the system to include at least one relevant item within the top N recommendations. The third method, Precision was used to evaluate the model where Mean Average Precision was used. This indicator also ranges from 0 to 1.

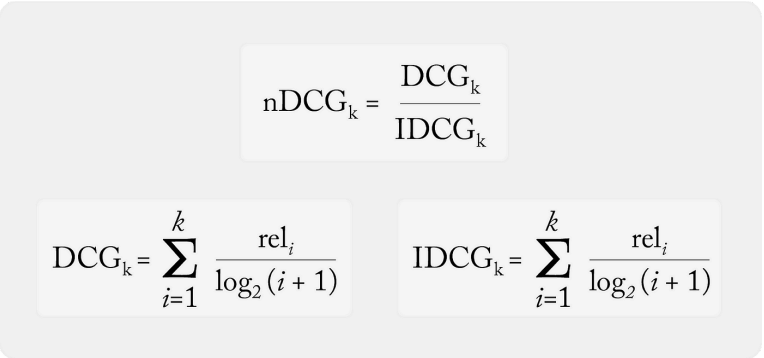

Figure 5.1: NDCG Formula

5.5 Summary

In this chapter, the focus was on the design of the solution which included the data preprocessing techniques, model architecture design and evaluation matrices used. The main two models developed are NCF model and TFR GAM model approaches with custom architectures. The main two evaluation matrices used were NDCG and Hit Ratio which was discussed in subsection 5.4. In the next chapter we will discuss how we have implemented our design solution.

CHAPTER 06

IMPLEMENTATION

6.1 Introduction

In the preceding chapter, we presented the design of our study, which delves into utilizing item and user features to predict the relevance scores to perform an item re-ranking task. In this chapter we delve into the implementation of our solution design. This chapter is divided into several sub sections, starting with discussion on data preprocessing techniques used for both input and target data. We then move into discussing the model's implementation in detail. All the implementations in this study were conducted in a python environment by using AWS sagemaker service and by using various python libraries. The ‘boto3’ library was used for data importing, ‘NumPy’ library was used for data transformation tasks and final models were built using ‘Tensorflow’ and ‘Keras’ Libraries. Finally evaluations were done using custom functions built to calculate the performances.

6.2 Data Preprocessing

Data sourced from the data lake was processed using Snowflake as the main platform, serving as the primary tool for querying raw tables and constructing the necessary dataset. Additionally, preprocessing of the data was conducted within the Snowflake environment and created 19 subscriber-product related initial features. As the initial step, subscriber data from various data marts was collected, including recommended offers from two tribes: the upsell and churn tribes models. These recommendations were communicated to subscribers via SMS, and their responses were tracked over a six-month period from September 1, 2023, to February 29, 2024. This process captured a mobile-prepaid subscriber base of 1.3 million

individuals.

Customer context features such as Age on Network (AON), Age, and Net Promoter Score (NPS) were derived. AON and Age were calculated using the subscriber's join date and NIC information, respectively. Net Promoter Score (NPS) was obtained from survey data, where customers rated their likelihood of recommending the company on a scale of 0 to 5. Higher scores indicate greater loyalty and satisfaction. The average score was computed from scores given by subscribers at various touch points considering the past six months. Scores were categorized as follows: Promoters (scores 4-5), Passives (scores 2-3), and Detractors (scores 0-1), representing enthusiastic, satisfied but not enthusiastic, and dissatisfied customers, respectively. In case of data unavailability, the mode value was imputed as the default for AON and Age. In case of unavailability of NPS scores, scores from the NPS prediction model were used.

Additionally, we incorporated outputs from churn predictive models, such as churn propensity scores. These models run on a batch process basis daily. Further the relevance score was taken from the recommendation model.

As the next step, To calculate the “RFM_SCORE”, Recency, Frequency, and Monetary component features were created as a score from 0 to 5 utilizing the NTILE function in SQL. This function partitions the dataset into six groups based on the specified criterion and assigns a rank to each row within these groups. In our case, we divided the data into six groups to generate scores from 0 to 5. First, the data is collected for the three features as follows.

For recency, we identified the time elapsed since the customer's last purchase of the specific product. This involved identifying the most recent activation date before the offer communicated date for each product and computing the time difference between that date and the offer communicated date minus one and divided it into six groups based on this duration. Using the NTILE function the rows were ranked

within each group, with the most recent purchases receiving the highest rank. Similarly, for Frequency score, we calculated the total number of product activations within the preceding 180 days before the offer communicated date and divided the data into six groups based on activation frequency. Each row was then ranked within its respective group with the highest frequency receiving the highest rank. Lastly, for Monetary score, we aggregated the total spend on the product by the customer across all activations within the past 180 days before the offer communicated date. Again, the data was partitioned into six groups based on cumulative spend highest spend receiving the highest rank.

After subtracting 1 from the resulting ranks, we derived scores ranging from 0 to 5 for each component, with 0 representing the lowest and 5 representing the highest. By integrating these three components—Recency, Frequency, and Monetary—using historical product activation data, we approximated RFM scores for each customer-product combination. As each component's score ranged from 0 to 5, the combined RFM score varied from 0 (suggesting minimal engagement) to 15 (indicating the highest engagement level). When these calculations are completed, if R, F, M scores for the recommended products are not available, a score of 0 is assigned as missing value treatment.

To calculate takeup related features, first the “LAST_OFFER_RESPONSE”, is calculated as a binary feature (yes/no) by considering whether the subscriber has purchased any offer sent within the last 4 weeks and “TAKEUP” target label is created considering.whether subscribers purchased recommended products within the offer period, based on historical uptake data for relevant offers over the past six months before the offer communicated date. This helped us establish the target variable for our predictive modeling tasks.

Even though 19 attributes were created in the initial dataset, all attributes were not needed in the context of our re-ranking task, Therefore, we constructed a compact version of the dataset tailored for recommender tasks. The compact dataset consists

of three parts: model relevant five context, four item features and label only. Finally, the compact table was transferred to an AWS S3 bucket for the modeling phase.

In the AWS Sagemaker side, padding was applied after grouping the data set by subcriber. This was done to standardize the length of each subscriber to the subscriber with maximum number of offers. The padding was performed by appending rows filled with '-1' for all the columns whenever the group size is less than maximum offer count. This helps to make sure a consistent input dimension is there for the algorithm. After this step the input data is inserted to models as user and item data separately as required by models as inputs.

6.3 Model Development

The study aimed to create a re-ranking model with intrinsically interpretable architecture which has similar performance to that of black box model. The black box model developed was the NCF Model and the intrinsically interpretable model developed was the TFR GAM model. Both the models predict the relevance scores for each offer which then used to re-rank the offers.

6.3.1 NCF Model

The NCF model consists of several layers,each with a specific purpose. The input layer defines the shape of the input data and it consists of two input networks namely item network and user network. The size of the input layer for the user network is of size 5 for features, AGE, AON, NPS, Last Offer Seen, Churn Propensity. The size of the input layer of the item network is of size 4 for features, Offer Value, Offer Validity, RFM score of the product and Offer Relevance Score from the recommendation system. The batch size is left unspecified in the input layer, since it was configured through a Data Generator function which was used to insert training data into the fir function.

In the user network, the first layer after the input layer is a masking layer. This allows the network to identify and ignore the padded values within input data. Then the first dense layer consists of 128 neurons and uses ReLU(Rectified Linear Unit) activation function. It is designed to learn a non-linear transformation of the input data, capturing complex patterns in its user features. This follows by a dropout layer with rate 0.2, this layer randomly sets input units to zero at each step during training, which is similar to dropping nodes, which helps in preventing overfitting by providing regularization. Second dense layer consists of 32 neurons, also using the ReLU activation function. This layer continues to refine the learning of high-level user features captured by previous layers.

In the item network a similar architecture is developed,the first layer after input layer is a masking layer. This allows the network to identify and ignore the padded values within input data. Then the first dense layer has 16 neurons and uses ReLU activation function. It starts the process of learning from item features. This follows by a dropout layer with rate 0.2, this layer randomly sets input units to zero at each step during training, which is similar to dropping nodes which helps to mitigate overfitting during the training process. Second dense layer with ReLU activation function continues to refine the learning of high-level user features captured by previous layers. This also followed by a dropout layer with rate 0.2 adds additional regularization. Third dense layer consists of 4 neurons using the ReLU activation function that continues the non-linear interaction in the item features from previous layers.

Finally a wrapper network, which concatenates the outputs from user and item networks which creates a single vector that includes output features from both user and item networks. First dense layer here consists of 64 neurons with ReLU activation function which begins the integration of the user and items out features into predictive output. This is followed by a dropout layer with rate 0.3 which adds robustness to the model against overfitting.Second dense layer continues processing combined features from previous layer with 32 neurons and ReLu activation

function. Another dropout layer with rate 0.4 is added to intensify the regularization as the model finalizes its predictions.

Finally, the output dense layer consists of 1 neuron with sigmoid activation function which outputs a relevance score indicating the likelihood that the user will buy the item.

Adam is used as the optimizer with a learning rate 0.01, Adam is an extension of stochastic gradient descent. It combines the best properties of AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. The loss function used is 'Binary Crossentropy' since this will output a score between 0 to 1.

6.3.2 TFR GAM Model

The TFR GAM Model consists of a bucket of sub models which can be mainly categorized as item and context networks. Each network is a collection of sub models. Each layer in the sub models contain their specific purpose.

The item network for each item features a separate sub model it developed. First dense layer consisting of 64 neurons in each sub model uses ReLU activation function to capture complex patterns within each individual item feature. This follows by a dropout layer with rate 0.2 for regularization and a normalization layer for normalizing the output of the previous layer, reducing the internal covariate shift. Second dense layer constant of 32 neurons uses the reLU activation function to continue learning more complex features followed by a dropout layer with rate 0.3 and a normalization layer for regularization and stabilization. Final layer of each sub-model of the item network consists of a single neural without any activation function which provides a linear transformation of the output value reflecting the score of that feature.

The context network also consists of sub-models for each feature. Similar to item network sub-models, first layers consist of dense layer, dropout layer and a normalization layer to process context features similar to item features, employing the same techniques for non-linearity,regularization and stabilization. These networks each sub-model consist of two such layers (dense, dropout and normalization). The final layer of this network consists of a dense layer with the number of neurons equal to the number of item features with softmax as the activation function. This ensures importance weights across all item features are summed to one, making them interpretable as weights.

The last section of the architecture contains a combining network where for each item feature, the importance weights from context features are summed and then multiplied with the item sub-score. This results in a context weighted score for each item feature, effectively modulating the item scores by the learned context importance(Importance Weights). Next layer is an add layer where all context-weights item scores are summed together to form a preliminary final score, finally this followed by a dense layer single neuron and sigmoid activation function to provide the final relevance score which indicates the likelihood users desire to purchase the item.

The learning rate is given as exponential decay where the learning rate is initially set high and decays exponentially. This helps the model to start with rapid learning and then refine its adjustments as it converges, optimizing the training process. Adam optimizer is used with a learning rate schedule which is defined in an exponential decay fashion. The loss function used is ‘Binary Crossentropy’ since this will output a score between 0 to 1.

6.4 Evaluation Implementation

The main evaluation indicators used were NDCG, Precision and Hit Ratio. NDCG assesses the effectiveness of the ranking algorithm based on the relevance of the

items it retrieves and the order in which these items are presented. In this study the input data was divided into training and testing and testing data was used to evaluate the models. Custom functions were used to calculate NDCG@2,NDCG@3, Hit Ratio@1,Hit Ratio@2,Hit Ratio@3, where for subscriber in the test set both NDCG values (2,3) and Hit Ratio values (1,2,3) were calculated and them they were averages using simple arithmetic mean to get the final values. Precision in the context of a re-ranking system quantifies the accuracy of the re-ranking at top N by measuring the fraction of top N re-ranked items that are relevant to the user. Precision@2,Precision@3 were calculated for each subscriber then they were averaged using simple arithmetic mean to get Mean Average Precision@2 and Mean Average Precision@3.

6.4 Summary

In this chapter, we discussed a detailed account of the implementation of our study. Specifically, we discussed the data preprocessing in detail, also we delved into implementation of the two models NCF Model and TFR GAM Model for the re-ranking task in our study, providing a thorough account of their specifications. Lastly, we outlined the evaluation matrices we used to measure the performances of each model. Moving forward in the next chapter, will present an extensive discussion of the results we obtain in related to performance indicators discussed in this chapter.

CHAPTER 07

EVALUATION

7.1 Introduction

In the prior chapter, we covered the implementation of our study. In this chapter we report the findings of our research and we evaluate the developed models. To evaluate the effectiveness of the interpretable re-ranking system compared to black box NCF model, we have done our evaluation using 3 performance matrices named NDCG, Hit Ratio and Mean Average Precision. The subscriber base used for evaluation is 15K +. This chapter is divided as Model Evaluation, Novelty of our research findings followed by a summary of the chapter.

7.2 Model Evaluation

We have mainly used three evaluation metrics to measure the performance of the two models, NCF Model and TFR GAM Model. Both models were set to train for 100 epochs with batch size of 64 and starting learning rate of 0.01 with exponential decay. After training both the models we evaluated the results. All the evaluation metrics were calculated using a test data set where it was not seen by model.

Figure 7.1 Contains the summary of performance metrics for the NCF model. NDCG_2 is the NDCG@2 and others also follow a similar naming convention. Figure 7.2 contains the summary of performance metrics for TFR GAM Model. NDCG was utilized to quantitatively measure the effectiveness of our ranking algorithm in predicting user relevance and presenting them in a meaningful order compared to the ideal order, a higher value in the performance metric indicates a

better model performance, even though performance of both the models are low, our objective which is to build a interpretable model which perform in par with black box NCF model can be seen using this performance matric. There are two variations of the NDCG matric we have utilized, NDCG@2 and NDCG@3, where they are changed according to the number of top items that were selected to calculate the matric. In the NDCG@2 top 2 items were considered and in NDCG@3, top 3 items were considered.

Hit Ratio assesses the effectiveness of our ranking algorithm in terms of user engagement. A higher value indicates a better model performance. We have utilized two variations of hit ratio, Hit Ratio@2 and Hit ratio@3, tailored to the number of top-selected items considered in the evaluation. Same as with NDCG, Hit ratio@2 related to top 2 items and Hot ratio@3 related to top 3 items. In here also despite overall results from both models are low, our goal to create interpretable models that perform comparably to a black box model is evident.

Precision was employed to quantitatively evaluate the accuracy of our ranking algorithm, focussing on how well the recommended items match the actual items of interest to the user, A high precision value indicates a better model performance. We have utilized two variations of precision, Precision@2 and Prevcision@3, which were adapted based on the number of top-ranked items considered in the evaluation. Although both models show relatively a modest performance, our objective of developing an interpretable model that parallels the performance of a complex model is seen.

Also two base models were evaluated using the same performance matrices NDCG, Hit Ratio and Precision. The overall performance of those models are slightly lower than the NCF and TFR GAM model. One limitation of these results can be biased towards the data set we created. These results are shown in the Figure 7.3, which contains the summary of all 4 models.

Overall, in our evaluation, results suggest that both have modest performance, however since both models have similar performance, it shows that our TFR GAM architecture which has intrinsic interpretability also performs at par with black box NCF model.

NDCG_2	NDCG_3	HIT_RATIO_2	HIT_RATIO_3	Precision_2	Precision_3
0.398781	0.464742	0.450745	0.451008	0.411512	0.408771

Figure 7.1 : NCF Model Performance Metrics

NDCG_2	NDCG_3	HIT_RATIO_2	HIT_RATIO_3	Precision_2	Precision_3
0.384946	0.452697	0.436426	0.438237	0.395762	0.396531

Figure 7.2 : TFR GAM Model Performance Metrics

Model	NDCG_2	NDCG_3	HIT_RATIO_2	HIT_RATIO_3	Precision_2	Precision_3
NCF	0.398781	0.464742	0.450745	0.451008	0.411512	0.408771
TFR	0.384946	0.452697	0.436426	0.438237	0.395762	0.396531
SVMRank	0.350895	0.394998	0.391824	0.37146	0.361517	0.339307
LambdaMart	0.407631	0.423664	0.425869	0.362702	0.415941	0.347098

Figure 7.3 : Model Performance Metrics Summary

7.3 Summary

In current chapter, we focussed on evaluation of two different models, NCF Model and TFR GAM Model, in the context of re-ranking tasks. The results indicated that the both models were having modest performance and those performance metrics are similar. Both the models used similar hyper parameters such as Adam Optimizer, Decaying Learning Rate with same initial rate (0.01) and same batch size of 64. Even Though both models show modest performance, both the models have shown similar performances regardless of their architecture change, but there is still room for improvement in the performance metrics by improving the models.

In Conclusion, we have successfully accomplished our objective by conducting a thorough literature review and discussing adopted technologies to address the research problem.

CHAPTER 08

CONCLUSION AND FUTURE WORKS

8.1 Introduction

In Chapter 1, we emphasized on the significance of recommendation systems in the modern world and then moved towards the importance of learning to rank re-ranking systems. Then we reviewed previous studies in the problem domain. We defined our problem as “Predicting the relevance score for interpretable re-ranking tasks in the context of the telecommunication industry has been a research challenge due to heavy computational cost and performance of interpretable architecture” and our solution to address it. In the next chapter, which is chapter 2, we provided a critical review of literature on the recommendation systems, learning to rank systems and re-ranking systems. We also discussed the historical approaches and evolution of recommendation systems aligned with challenges. In Chapter 3, we provided a detailed description of technologies adopted, aligned to existing and proposed solutions. In Chapter 4, we discussed the approach we adopted for the study within six primary sections which includes, input, output, process, hypothesis, features and users. In this Chapter we explained the underlying hypothesis, input data formats, required output formats, step by step procedure for converting input to output, essential characteristics of the model, and its intended audience and benefits that they will receieve. In the next two Chapters, which are Chapters 5 and 6, we discussed the design and its implementation in detail. We organized this into several sub chapters, data preprocessing, model design and implemented results. In Chapter 7, we presented the evaluation of results for developed two models in detail using various evaluation metrics such as NDCG, Hit ratio and Precision. Finally in the Chapter 8, we summarized our findings, provided recommendations, limitations of the study and future works to improve the performances of the proposed solution.

8.2 Conclusion

The area of recommendation systems has made significant strides in recent years and it has now evolved to recommendation ranking and re-ranking areas as well. Our main objective was to explore the potential of interpretable re-ranking model which provide similar performances as that of a deep learning black box model. Recent studies have demonstrated that it is possible to develop interpretable ranking systems. This inspired the development of the hypothesis behind this study which was, it is possible to create a new interpretable ranking model with similar performance to that of black box ranking models in the context of the telecommunication industry.

In this we developed two models, one of a NCF model which is a black box model and second one is an intrinsically interpretable model, which is based on TFR GAM archietcture. The study was aimed to tune the TFR GAM model architecture to provide similar results compared to the NCF model based on the performance metrics we used to evaluate the two models. Both the models had the same training and testing data sets to maintain consistency.

Following the design of high level model architectures, we developed the TFR GAM Model, with the aim of achieving similar performances to that of the NCF model. Output from both the models are relevance scores which were used to rank the offers subscriber wise. The main three evaluation metrics used were NDCG, Hit Ratio and Precision, For all metrics top 2 and top 3 values were calculated to evaluate the two models.Overall, for all the evaluation metrics, the intrinsically interpretable model had similar performance relative to the NCF model. But there is still room for improvement of the TFR GAM Model. According to the study, TFR GAM model can be used for re-ranking tasks which gives similar results to that of black box models (NCF Model). For use cases where interpretability is important, we can deploy this intrinsically interpretable TFR GAM model to achieve re-ranking tasks.

8.3 Summary of Key Limitations

The proposed model utilizes a TensorFlow Generalized Additive Model (GAM) architecture, which demonstrates comparable performance to the Neural Collaborative Filtering (NCF) model in re-ranking tasks. However, it has notable limitations. Firstly, the model does not effectively address the cold start problem. This limitation means that the model struggles with providing accurate recommendations for new users or items with insufficient interaction data, which can affect its overall applicability in dynamic and evolving datasets.

Secondly, since the TFR GAM is not a fully connected model, it sometimes falls short in performance compared to the NCF model. The absence of full connectivity in TFR can lead to suboptimal learning of complex interactions between features, thereby impacting the robustness and accuracy of the recommendations under certain conditions. Despite these limitations, the model provides a viable alternative for specific re-ranking tasks, but future work should focus on addressing these shortcomings to enhance its applicability and performance.

Beyond the architectural constraints, the proposed TFR GAM model is also subject to several data-related limitations inherent in recommendation systems. The quality and availability of data significantly influence the model's performance. Issues such as sparse data, incomplete records, and potential biases in the dataset can adversely affect the accuracy and reliability of the recommendations. Sparse data, where there are few interactions per user or item, hampers the model's ability to learn robust patterns. Incomplete records and missing values can lead to inaccurate predictions, while biases present in the data may cause the model to reinforce existing biases, leading to unfair or suboptimal recommendations. Addressing these data-related challenges is crucial for improving the model's effectiveness and ensuring it delivers reliable and unbiased recommendations.

8.4 Future Works

This study presented two models. NCF Model which is a black box model and TFR GAM model which is an intrinsically interpretable model. Both models had moderate to low performances on all the evaluation metrics used. To improve the performance of the model several recommendations can be made. For both the models additional features should be developed and incorporated to improve the model performances. These features come mainly as Contextual features which provide diverse contextual

information and Item features which provide information about the offer that is recommended to the subscriber. Future research can use NLP models to extract subscriber as wel as item features from unstructured data formats like text data which will greatly improve the performances of the model. Additionally, can try different combinations for the number of hidden layers, activation functions, and regularization techniques. Additionally, this study can be future improved to build a system that directly provides the final score and the sub scores and weights that were used to derive the final score.

Another extension to this study comes from the ethical and fairness domain, even though this model architecture supports better transparency into the model by intrinsically explaining why a certain offer is scored at that scoring value, this can further improved with continued studies in the field. Also new techniques can be discovered to ensure the fairness of the model.

Overall, there is a great potential for further work in this area, and continued research and development could lead to significant advancement in the field of developing interpretable recommendation models.

8.4 Summary

In this chapter, we presented the overall conclusion of our study on developing interpretable re-ranking models. Our study has shown that it is possible to create interpretable re-ranking models which have similar performances to that of a black box model. We have also provided a comprehensive summary of our study findings and discussed the recommendation for future works that can be done to improve the model. We believe that continuous research and development in this area could lead to significant advancements in this field of interpretable recommendation systems.

We hope that our study's contribution can pave the way for future research work to improve this field.

REFERENCES

[1] H. Ko, S. Lee, Y. Park, and A. Choi, “A Survey of Recommendation Systems: Recommendation Models, Techniques, and Application Fields,” p. 48, 2022, doi: 10.3390/electronics11010141.

[2] “Smartphone subscriptions worldwide 2027,” Statista. Accessed: Aug. 07, 2022. [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[3] G. Jain, S. Rakesh, M. Kamalun Nabi, and K. R. Chaturvedi, “Hyper-personalization – fashion sustainability through digital clienteling,” *Res. J. Text. Appar.*, vol. 22, no. 4, pp. 320–334, Nov. 2018, doi: 10.1108/RJTA-02-2018-0017.

[4] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, “Using collaborative filtering to weave an information tapestry,” *Commun. ACM*, vol. 35, no. 12, pp. 61–70, Dec. 1992, doi: 10.1145/138859.138867.

[5] R. Singh, K. Chuchra, and A. Rani, “A Survey on the Generation of Recommender Systems,” *Int. J. Inf. Eng. Electron. Bus.*, vol. 9, pp. 26–35, May 2017, doi: 10.5815/ijieeb.2017.03.04.

[6] W. Lu, S. Chen, K. Li, and L. V. S. Lakshmanan, “Show me the money: dynamic recommendations for revenue maximization,” *Proc. VLDB Endow.*, vol. 7, no. 14, pp. 1785–1796, Oct. 2014, doi: 10.14778/2733085.2733086.

[7] R. Louca, M. Bhattacharya, D. Hu, and L. Hong, “Joint Optimization of Profit and Relevance for Recommendation Systems in E-commerce,” p. 4, 2019.

[8] W. S. Cooper, F. C. Gey, and D. P. Dabney, “Probabilistic retrieval based on staged logistic regression,” in *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR ’92*, Copenhagen, Denmark: ACM Press, 1992, pp. 198–210. doi: 10.1145/133160.133199.

[9] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, “An Efficient Boosting Algorithm for Combining Preferences,” p. 37.

[10] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li, “Listwise Approach to Learning to Rank - Theory and Algorithm,” p. 9.

[11] C. J. C. Burges, “From RankNet to LambdaRank to LambdaMART: An Overview,” p. 19.

[12] R. K. Pasumarthi *et al.*, “TF-Ranking: Scalable TensorFlow Library for Learning-to-Rank,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Anchorage AK USA: ACM, Jul. 2019, pp. 2970–2978. doi: 10.1145/3292500.3330677.

[13] J. Singh and A. Anand, “EXS: Explainable Search Using Local Model Agnostic Interpretability,” 2018, doi: 10.48550/ARXIV.1809.03857.

[14] P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis, “Explainable AI: A Review of Machine Learning Interpretability Methods,” *Entropy*, vol. 23, no. 1, p. 18, Dec. 2020, doi: 10.3390/e23010018.

[15] H. Zhuang *et al.*, “Interpretable Ranking with Generalized Additive Models,” in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, Virtual Event Israel: ACM, Mar. 2021, pp. 499–507. doi: 10.1145/3437963.3441796.

[16] C.-C. Wang, M.-H. Lin, J.-S. R. Jang, and W. Liou, “An effective re-ranking method based on learning to rank for improving audio fingerprinting,” in *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014 Asia-Pacific*, Chiang Mai, Thailand: IEEE, Dec. 2014, pp. 1–4. doi: 10.1109/APSIPA.2014.7041558.

[17] C. Rudin, “Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead,” 2018, doi: 10.48550/ARXIV.1811.10154.

[18] J. Liu, P. Dolan, and E. R. Pedersen, “Personalized news recommendation

based on click behavior," in *Proceedings of the 15th international conference on Intelligent user interfaces*, Hong Kong China: ACM, Feb. 2010, pp. 31–40. doi: 10.1145/1719970.1719976.

[19] M. Rodriguez, C. Posse, and E. Zhang, "Multiple objective optimization in recommender systems," in *Proceedings of the sixth ACM conference on Recommender systems*, Dublin Ireland: ACM, Sep. 2012, pp. 11–18. doi: 10.1145/2365952.2365961.

[20] S. H. Park and K. Kim, "Collaborative filtering recommendation system based on improved Jaccard similarity," *J. Ambient Intell. Humaniz. Comput.*, vol. 14, no. 8, pp. 11319–11336, Aug. 2023, doi: 10.1007/s12652-023-04647-0.

[21] R. van Meteren and M. van Someren, "Using Content-Based Filtering for Recommendation," p. 10.

[22] R. Marcinkevičs and J. E. Vogt, "Interpretability and Explainability: A Machine Learning Zoo Mini-tour," 2020, doi: 10.48550/ARXIV.2012.01805.

[23] P. Barceló, M. Monet, J. Pérez, and B. Subercaseaux, "Model Interpretability through the Lens of Computational Complexity," 2020, doi: 10.48550/ARXIV.2010.12265.

[24] A. Zytek, I. Arnaldo, D. Liu, L. Berti-Equille, and K. Veeramachaneni, "The Need for Interpretable Features: Motivation and Taxonomy," 2022, doi: 10.48550/ARXIV.2202.11748.

[25] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural Collaborative Filtering." arXiv, Aug. 25, 2017. Accessed: Aug. 12, 2022. [Online]. Available: http://arxiv.org/abs/1708.05031

[26] M. A. Ghazanfar and A. Prugel-Bennett, "A Scalable, Accurate Hybrid Recommender System," in *2010 Third International Conference on Knowledge Discovery and Data Mining*, Jan. 2010, pp. 94–98. doi: 10.1109/WKDD.2010.117.

[27] C. Rudin and Y. Wang, "Direct Learning to Rank and Rerank," 2018, doi: 10.48550/ARXIV.1802.07400.

[28] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, “Hidden technical debt in machine learning systems. In Advances in neural information processing systems. 2,” *Neural Inf. Process. Syst.*, vol. 2, no. MIT Press, 2015, pp. 2503--2511.

[29] R. ElShawi, Y. Sherif, M. Al-Mallah, and S. Sakr, “ILIME: Local and Global Interpretable Model-Agnostic Explainer of Black-Box Decision,” in *Advances in Databases and Information Systems*, vol. 11695, T. Welzer, J. Eder, V. Podgorelec, and A. Kamišalić Latifić, Eds., in Lecture Notes in Computer Science, vol. 11695. , Cham: Springer International Publishing, 2019, pp. 53–68. doi: 10.1007/978-3-030-28730-6_4.

[30] Z. Chen, Z. Lian, and Z. Xu, “Interpretable Model-Agnostic Explanations Based on Feature Relationships for High-Performance Computing,” *Axioms*, vol. 12, no. 10, p. 997, Oct. 2023, doi: 10.3390/axioms12100997.

[31] Y. Zhang, P. Tiňo, A. Leonardis, and K. Tang, “A Survey on Neural Network Interpretability,” *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 5, no. 5, pp. 726–742, Oct. 2021, doi: 10.1109/TETCI.2021.3100641.

[32] B. Kveton, O. Meshi, M. Zoghi, and Z. Qin, “On the Value of Prior in Online Learning to Rank,” in *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, PMLR, May 2022, pp. 6880–6892. Accessed: Jan. 26, 2023. [Online]. Available: https://proceedings.mlr.press/v151/kveton22a.html

[33] M. Momma, A. B. Garakani, and Y. Sun, “Multi-objective Relevance Ranking,” p. 8, 2019.

Appendix - A: TFR Model Architecture Source code

```
    def user_embedding_net( self, input_dim ):
        """user embedding branch"""
        net_input = Input(shape=(input_dim,))
        masked_input = Masking(mask_value=-1)(net_input)
        net_output = Dense(128, activation='relu')(masked_input)
        net_output = Dropout(0.2)(net_output)
        net_output = Dense(32, activation='relu')(net_output)

        return Model(inputs=net_input,outputs=net_output)

    def item_embedding_net(self, product_dim):
        """item embedding branch"""
        net_input = Input(shape=(product_dim,))
        masked_input = Masking(mask_value=-1)(net_input)
        net_output =  Dense( 16 , activation='relu')(masked_input)
        net_output = Dropout(0.2)(net_output)
        net_output = Dense(10,activation='relu')(net_output)
        net_output = Dropout(0.2)(net_output)
        net_output = Dense(4,activation='relu')(net_output)

        return Model(inputs=net_input,outputs=net_output)

    def wrapper_net(self, user_net, item_net):
        """final ncf stream , concate the both item and user feature spaces"""
        net_output = concatenate([user_net.output,item_net.output])
        net_output = Dense(64,activation='relu')(net_output)
        net_output = Dropout(0.3)(net_output)
        net_output = Dense(32,activation='relu')(net_output)
        net_output = Dropout(0.4)(net_output)
        net_output = Dense(1,activation='sigmoid')(net_output)

        return Model(inputs=[user_net.input,item_net.input],outputs=net_output)
```

```
------
    def init_model(self):

        # define the ncf model
        user_net = self.user_embedding_net(self.user_dim)
        item_net = self.item_embedding_net(self.item_dim )
        model = self.wrapper_net(user_net, item_net)
----------
        initial_learning_rate = self.config['learning_rate']
        lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
            initial_learning_rate,
            decay_steps=1000,
            decay_rate=0.96,
            staircase=True)

        # add the optimizer and complile the model
        # opt = Adam(learning_rate= self.config['learning_rate'] )
----------
        opt = tf.keras.optimizers.Adam(learning_rate=lr_schedule)
        # model.compile(loss='binary_crossentropy', optimizer=opt,
        #               metrics=['accuracy',tf.keras.metrics.Precision()])
----------
        model.compile(loss=self.listnet_loss, optimizer=opt,
                      metrics=[tf.keras.metrics.Precision()])
----------
        return model
```

Appendix - B: NCF Model Architecture Source code

```
    def make_model(self,loss,drop_out):
         # Item and context inputs
        context_inputs = [tf.keras.Input(shape=(1,), name=f'{i}') for i in self.context_features if i != self.customer_id_column]
        item_inputs = [tf.keras.Input(shape=(1,), name=f'{i}') for i in self.example_features if i != self.customer_id_column]
-----
-----------
-----------
        num_item_features = len(self.example_features)
-----------
        # Sub-models for item features
        item_sub_models = []
        for item_input in item_inputs:
            masked_input = tf.keras.layers.Masking(mask_value=-1)(item_input)
            x = tf.keras.layers.Dense(self.item_layer_1, activation='relu')(masked_input)  # Using ReLU activation
            x = tf.keras.layers.Dropout(drop_out)(x)
            x = tf.keras.layers.BatchNormalization()(x)
            x = tf.keras.layers.Dense(self.item_layer_2, activation='relu')(x)  # Using ReLU activation
            x = tf.keras.layers.Dropout(0.3)(x)
            x = tf.keras.layers.BatchNormalization()(x)
            sub_score = tf.keras.layers.Dense(1, activation=None)(x)  # Linear activation is okay for the output layer
            item_sub_models.append(sub_score)

        # Sub-models for context features to derive importance weights
        context_sub_models = []
        for context_input in context_inputs:
            masked_input = tf.keras.layers.Masking(mask_value=-1)(context_input)
            x = tf.keras.layers.Dense(self.context_layer_1, activation='relu')(masked_input)  # Using ReLU activation
            x = tf.keras.layers.Dropout(drop_out)(x)
            x = tf.keras.layers.BatchNormalization()(x)
            x = tf.keras.layers.Dense(self.context_layer_2, activation='relu')(x)
            x = tf.keras.layers.Dropout(0.3)(x)
            x = tf.keras.layers.BatchNormalization()(x)
            # x = tf.keras.layers.Dense(self.context_layer_3, activation='relu')(x)
            importance_weights = tf.keras.layers.Dense(num_item_features, activation='softmax')(x)  # Softmax to derive weights
            context_sub_models.append(importance_weights)
---------------
        # Combine context weights with item sub-scores
        weighted_scores = []
        for i, item_sub_model in enumerate(item_sub_models):
            context_weights = [context_sub_model[:, i] for context_sub_model in context_sub_models]
            total_context_weight = tf.keras.layers.Add()(context_weights)
            weighted_score = tf.keras.layers.Multiply()([total_context_weight, item_sub_model])
            weighted_scores.append(weighted_score)




    ----------------
             # Final score
            final_score = tf.keras.layers.Add()(weighted_scores)
            final_score = tf.keras.layers.Dense(1, activation='sigmoid')(final_score)
    -----------
    -----------
    -----------

    -----------
            initial_learning_rate = 0.1
            lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
                initial_learning_rate,
                decay_steps=1000,
                decay_rate=0.96,
                staircase=True)
    -----------
            optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)

            # Create and compile model
            model = tf.keras.Model(inputs=item_inputs + context_inputs, outputs=final_score)
            # model.compile(optimizer=optimizer, loss='binary_crossentropy',metrics=[tf.keras.metrics.Precision()])
            model.compile(optimizer=optimizer, loss=self.listnet_loss,metrics=[tf.keras.metrics.Precision()])

            return model
```

Appendix C: TFR Architecture

masking_17 (Masking)	(None, 1)	0	['CHURN_PROPENSITY[0][0]']
dense_40 (Dense)	(None, 64)	128	['masking_13[0][0]']
dense_43 (Dense)	(None, 64)	128	['masking_14[0][0]']
dense_46 (Dense)	(None, 64)	128	['masking_15[0][0]']
dense_49 (Dense)	(None, 64)	128	['masking_16[0][0]']
dense_52 (Dense)	(None, 64)	128	['masking_17[0][0]']
VALUE (InputLayer)	[(None, 1)]	0	[]
PRODUCT_VALIDITY (InputLayer)	[(None, 1)]	0	[]
RFM_SCORE (InputLayer)	[(None, 1)]	0	[]
RELAVANCE_SCORE (InputLayer)	[(None, 1)]	0	[]
dropout_26 (Dropout)	(None, 64)	0	['dense_40[0][0]']
dropout_28 (Dropout)	(None, 64)	0	['dense_43[0][0]']
dropout_30 (Dropout)	(None, 64)	0	['dense_46[0][0]']
dropout_32 (Dropout)	(None, 64)	0	['dense_49[0][0]']
dropout_34 (Dropout)	(None, 64)	0	['dense_52[0][0]']
masking_9 (Masking)	(None, 1)	0	['VALUE[0][0]']
masking_10 (Masking)	(None, 1)	0	['PRODUCT_VALIDITY[0][0]']
masking_11 (Masking)	(None, 1)	0	['RFM_SCORE[0][0]']
masking_12 (Masking)	(None, 1)	0	['RELAVANCE_SCORE[0][0]']
batch_normalization_26 (BatchNormalization)	(None, 64)	256	['dropout_26[0][0]']
batch_normalization_28 (BatchNormalization)	(None, 64)	256	['dropout_28[0][0]']
batch_normalization_30 (BatchNormalization)	(None, 64)	256	['dropout_30[0][0]']
batch_normalization_32 (BatchNormalization)	(None, 64)	256	['dropout_32[0][0]']
batch_normalization_34 (BatchNormalization)	(None, 64)	256	['dropout_34[0][0]']

```
dense_28 (Dense)            (None, 64)                128       ['masking_9[0][0]']

dense_31 (Dense)            (None, 64)                128       ['masking_10[0][0]']

dense_34 (Dense)            (None, 64)                128       ['masking_11[0][0]']

dense_37 (Dense)            (None, 64)                128       ['masking_12[0][0]']

dense_41 (Dense)            (None, 32)                2080      ['batch_normalization_26[0][0]
                                                                  ']

dense_44 (Dense)            (None, 32)                2080      ['batch_normalization_28[0][0]
                                                                  ']

dense_47 (Dense)            (None, 32)                2080      ['batch_normalization_30[0][0]
                                                                  ']

dense_50 (Dense)            (None, 32)                2080      ['batch_normalization_32[0][0]
                                                                  ']

dense_53 (Dense)            (None, 32)                2080      ['batch_normalization_34[0][0]
                                                                  ']

dropout_18 (Dropout)        (None, 64)                0         ['dense_28[0][0]']

dropout_20 (Dropout)        (None, 64)                0         ['dense_31[0][0]']

dropout_22 (Dropout)        (None, 64)                0         ['dense_34[0][0]']

dropout_24 (Dropout)        (None, 64)                0         ['dense_37[0][0]']

dropout_27 (Dropout)        (None, 32)                0         ['dense_41[0][0]']

dropout_29 (Dropout)        (None, 32)                0         ['dense_44[0][0]']

dropout_31 (Dropout)        (None, 32)                0         ['dense_47[0][0]']

dropout_33 (Dropout)        (None, 32)                0         ['dense_50[0][0]']

dropout_35 (Dropout)        (None, 32)                0         ['dense_53[0][0]']

batch_normalization_18 (Ba  (None, 64)                256       ['dropout_18[0][0]']
tchNormalization)

batch_normalization_20 (Ba  (None, 64)                256       ['dropout_20[0][0]']
tchNormalization)

batch_normalization_22 (Ba  (None, 64)                256       ['dropout_22[0][0]']
tchNormalization)

batch_normalization_24 (Ba  (None, 64)                256       ['dropout_24[0][0]']
tchNormalization)
```

batch_normalization_27 (BatchNormalization)	(None, 32)	128	['dropout_27[0][0]']
batch_normalization_29 (BatchNormalization)	(None, 32)	128	['dropout_29[0][0]']
batch_normalization_31 (BatchNormalization)	(None, 32)	128	['dropout_31[0][0]']
batch_normalization_33 (BatchNormalization)	(None, 32)	128	['dropout_33[0][0]']
batch_normalization_35 (BatchNormalization)	(None, 32)	128	['dropout_35[0][0]']
dense_29 (Dense)	(None, 32)	2080	['batch_normalization_18[0][0]']
dense_32 (Dense)	(None, 32)	2080	['batch_normalization_20[0][0]']
dense_35 (Dense)	(None, 32)	2080	['batch_normalization_22[0][0]']
dense_38 (Dense)	(None, 32)	2080	['batch_normalization_24[0][0]']
dense_42 (Dense)	(None, 4)	132	['batch_normalization_27[0][0]']
dense_45 (Dense)	(None, 4)	132	['batch_normalization_29[0][0]']
dense_48 (Dense)	(None, 4)	132	['batch_normalization_31[0][0]']
dense_51 (Dense)	(None, 4)	132	['batch_normalization_33[0][0]']
dense_54 (Dense)	(None, 4)	132	['batch_normalization_35[0][0]']
dropout_19 (Dropout)	(None, 32)	0	['dense_29[0][0]']
dropout_21 (Dropout)	(None, 32)	0	['dense_32[0][0]']
dropout_23 (Dropout)	(None, 32)	0	['dense_35[0][0]']
dropout_25 (Dropout)	(None, 32)	0	['dense_38[0][0]']
tf.__operators__.getitem_20 (SlicingOpLambda)	(None,)	0	['dense_42[0][0]']

```
tf.__operators__.getitem_2  (None,)                  0         ['dense_45[0][0]']
1 (SlicingOpLambda)

tf.__operators__.getitem_2  (None,)                  0         ['dense_48[0][0]']
2 (SlicingOpLambda)

tf.__operators__.getitem_2  (None,)                  0         ['dense_51[0][0]']
3 (SlicingOpLambda)

tf.__operators__.getitem_2  (None,)                  0         ['dense_54[0][0]']
4 (SlicingOpLambda)

batch_normalization_19 (Ba  (None, 32)               128       ['dropout_19[0][0]']
tchNormalization)

tf.__operators__.getitem_2  (None,)                  0         ['dense_42[0][0]']
5 (SlicingOpLambda)

tf.__operators__.getitem_2  (None,)                  0         ['dense_45[0][0]']
6 (SlicingOpLambda)

tf.__operators__.getitem_2  (None,)                  0         ['dense_48[0][0]']
7 (SlicingOpLambda)

tf.__operators__.getitem_2  (None,)                  0         ['dense_51[0][0]']
8 (SlicingOpLambda)

tf.__operators__.getitem_2  (None,)                  0         ['dense_54[0][0]']
9 (SlicingOpLambda)

batch_normalization_21 (Ba  (None, 32)               128       ['dropout_21[0][0]']
tchNormalization)

tf.__operators__.getitem_3  (None,)                  0         ['dense_42[0][0]']
0 (SlicingOpLambda)

tf.__operators__.getitem_3  (None,)                  0         ['dense_45[0][0]']
1 (SlicingOpLambda)

tf.__operators__.getitem_3  (None,)                  0         ['dense_48[0][0]']
2 (SlicingOpLambda)

tf.__operators__.getitem_3  (None,)                  0         ['dense_51[0][0]']
3 (SlicingOpLambda)

tf.__operators__.getitem_3  (None,)                  0         ['dense_54[0][0]']
4 (SlicingOpLambda)

batch_normalization_23 (Ba  (None, 32)               128       ['dropout_23[0][0]']
tchNormalization)

tf.__operators__.getitem_3  (None,)                  0         ['dense_42[0][0]']
5 (SlicingOpLambda)
```

```
tf.__operators__.getitem_3  (None,)                  0         ['dense_45[0][0]']
6 (SlicingOpLambda)

tf.__operators__.getitem_3  (None,)                  0         ['dense_48[0][0]']
7 (SlicingOpLambda)

tf.__operators__.getitem_3  (None,)                  0         ['dense_51[0][0]']
8 (SlicingOpLambda)

tf.__operators__.getitem_3  (None,)                  0         ['dense_54[0][0]']
9 (SlicingOpLambda)

batch_normalization_25 (Ba  (None, 32)               128       ['dropout_25[0][0]']
tchNormalization)

add_5 (Add)                 (None,)                  0         ['tf.__operators__.getitem_20[
                                                               0][0]',
                                                                'tf.__operators__.getitem_21[
                                                               0][0]',
                                                                'tf.__operators__.getitem_22[
                                                               0][0]',
                                                                'tf.__operators__.getitem_23[
                                                               0][0]',
                                                                'tf.__operators__.getitem_24[
                                                               0][0]']

dense_30 (Dense)            (None, 1)                33        ['batch_normalization_19[0][0]
                                                               ']

add_6 (Add)                 (None,)                  0         ['tf.__operators__.getitem_25[
                                                               0][0]',
                                                                'tf.__operators__.getitem_26[
                                                               0][0]',
                                                                'tf.__operators__.getitem_27[
                                                               0][0]',
                                                                'tf.__operators__.getitem_28[
                                                               0][0]',
                                                                'tf.__operators__.getitem_29[
                                                               0][0]']

dense_33 (Dense)            (None, 1)                33        ['batch_normalization_21[0][0]
                                                               ']
```

```
add_7 (Add)                 (None,)                   0         ['tf.__operators__.getitem_30[
                                                                 0][0]',
                                                                  'tf.__operators__.getitem_31[
                                                                 0][0]',
                                                                  'tf.__operators__.getitem_32[
                                                                 0][0]',
                                                                  'tf.__operators__.getitem_33[
                                                                 0][0]',
                                                                  'tf.__operators__.getitem_34[
                                                                 0][0]']

dense_36 (Dense)            (None, 1)                 33        ['batch_normalization_23[0][0]
                                                                 ']

add_8 (Add)                 (None,)                   0         ['tf.__operators__.getitem_35[
                                                                 0][0]',
                                                                  'tf.__operators__.getitem_36[
                                                                 0][0]',
                                                                  'tf.__operators__.getitem_37[
                                                                 0][0]',
                                                                  'tf.__operators__.getitem_38[
                                                                 0][0]',
                                                                  'tf.__operators__.getitem_39[
                                                                 0][0]']


dense_39 (Dense)            (None, 1)                 33        ['batch_normalization_25[0][0]
                                                                 ']

multiply_4 (Multiply)       (None, 1)                 0         ['add_5[0][0]',
                                                                  'dense_30[0][0]']

multiply_5 (Multiply)       (None, 1)                 0         ['add_6[0][0]',
                                                                  'dense_33[0][0]']

multiply_6 (Multiply)       (None, 1)                 0         ['add_7[0][0]',
                                                                  'dense_36[0][0]']

multiply_7 (Multiply)       (None, 1)                 0         ['add_8[0][0]',
                                                                  'dense_39[0][0]']

add_9 (Add)                 (None, 1)                 0         ['multiply_4[0][0]',
                                                                  'multiply_5[0][0]',
                                                                  'multiply_6[0][0]',
                                                                  'multiply_7[0][0]']

dense_55 (Dense)            (None, 1)                 2         ['add_9[0][0]']

==================================================================================================
Total params: 24122 (94.23 KB)
Trainable params: 22394 (87.48 KB)
Non-trainable params: 1728 (6.75 KB)
```