FAKULTÄT FÜR
INFORMATIK

Otto-von-Guericke-University Magdeburg

Faculty of Computer Science

Department of Latex Templates

Leveraging explainability to improve ranking as part of information retrieval

Master Thesis

Author:

Shivani Jadhav

Examiner and Supervisor:

Prof. Dr.-Ing. Ernesto William De Luca

2nd Examiner:

Dr. Konstantin Mergenthaler

Supervisor:

M.Sc. Erasmo Purificato

Magdeburg, 27.07.2022

Contents

Acknowledgements 3

Abstract 4

1 Introduction 5
- 1.1 Motivation . . . 6
- 1.2 Goals . . . 7
- 1.3 Research Questions . . . 8
- 1.4 Research Methodology . . . 9
- 1.5 Thesis Outline . . . 9

2 Related Work 10

3 Concepts 12
- 3.1 Information retrieval . . . 12
- 3.2 BM25 . . . 13
- 3.3 Machine learning . . . 13
  - 3.3.1 Neural Networks . . . 14
- 3.4 Bipartite ranking . . . 15
- 3.5 Learning to Rank (LTR) . . . 16
  - Pointwise . . . 18
  - Pairwise . . . 18
  - Listwise . . . 20
- 3.6 Evaluation metrics . . . 23
- 3.7 Explainable AI (xAI) . . . 24
  - 3.7.1 SHapley Additive exPlanations (SHAP) . . . 24

4 Used Technologies 26
- 4.1 Algorithms . . . 26
  - 4.1.1 Traditional Machine learning for LTR . . . 26
    - Pointwise LTR using Random Forest Classifier . . . 26
    - Pointwise and Pairwise LTR using XGBoost . . . 28
  - 4.1.2 Neural Networks for LTR using tensorflow-ranking . . . 29
    - Pointwise LTR using tensorflow-ranking . . . 31
    - Pairwise LTR using tensorflow-ranking . . . 31
    - Listwise LTR using tensorflow-ranking . . . 32
- 4.2 Feature Selection Methods . . . 32
- 4.3 Sampling Strategies . . . 35
- 4.4 Benchmarking . . . 36

5 Demand Processing Pipeline; Design, Implementation and Evaluation 37
5.1 Data . . . 37
5.1.1 Raw Data . . . 37
5.1.2 Feature Generation using OpenSearch . . . 39
5.2 Experimental Setup . . . 44
5.2.1 Experiments with the handcrafted features . . . 44
Pipeline . . . 44
Results . . . 46
5.2.2 Intermediate experiments with features generated using OpenSearch 47
Pipeline . . . 47
Results . . . 48
5.2.3 Final experiments with features generated using OpenSearch . . . 49
Pipeline . . . 50
Results . . . 51
5.3 User Study . . . 54
5.3.1 Observations . . . 54

6 Discussion 57

7 Conclusion and Future Work 61

Appendices 62
A. 79 handcrafted features . . . 62

Bibliography 65

Statement of Authorship / Selbstständigkeitserklärung 70

Acknowledgements

I would like to thank Konstantin Mergenthaler and Erasmo Purificato for their constant guidance and feedback while doing this thesis. I would like to thank Prof. Dr.-Ing. Ernesto William De Luca for giving me the opportunity to write my thesis under his supervision. Coming to Germany and doing my master's at Otto von Guericke university has been the best decision of my life. I want to thank the entire OVGU faculty who have helped me grow both professionally and personally. Thank you to Konstantin Mergenthaler and Nischal Padmanabha for being awesome mentors to me. I would like to thank my colleagues at Scoutbee, from whom I have learned a lot of things. Lastly and most importantly, I would like to thank my family and friends for trusting me and being there for me.

Abstract

In the supply chain industry, information retrieval can be defined as finding relevant suppliers given a demand. Information retrieval is finding relevant information from the available noisy data by ranking the information based on its relevance to the requested query. Learning to Rank (LTR) solves information retrieval tasks using Machine Learning. LTR consists of three approaches referred to as *pointwise*, *pairwise* and *listwise*. In this thesis, we explore whether the use of ranking models that require extensive manual feature generation is justified in the supplier search context. Therefore, all three approaches are experimented by using traditional machine learning algorithms and neural networks to find a ranking model that ranks the relevant suppliers at the top, given the specific demand. One of the challenges is the class imbalance of the dataset, which is tackled using various sampling strategies, and the sampled data is used to train the respective ranking models. A user study involving domain experts is conducted to choose the best performing model, along with measuring the performance of all the ranking models on a test set using the evaluation metrics *Precision@k* and *Mean Average Precision*. Furthermore, this thesis aims to justify the use of self-explanatory features. The results show that a basic model, i.e., a random forest classifier with fewer self-explanatory features, is preferred by the ranking systems involving human-in-the-loop. However, having just one ranking model fails to generalize over all the demands as the demands belong to a wide range of domains. An explainable AI module to know the feature importance is implemented to gain insights from the ranking models, which helps to develop the next set of features for the next version of the ranking models.

1 Introduction

Data is in abundance. One of the challenges is to find relevant information from this plethora of data. This is where information retrieval (IR) comes into the picture. Information retrieval stands for retrieving relevant information as per the requirement. One use-case is from the supply chain industry, where finding relevant suppliers of the required item is like finding a *needle in a haystack*. Elaborating further, manufacturers depend on various suppliers when building their products. For example, a car manufacturing company would require rubber tyres, and querying noisy internet data to find suitable suppliers of rubber tyres is time-consuming. Scoutbee is a company that operates specifically in the procurement space, and it provides, as a service, a long list of potential suppliers for a particular requirement demanded by a customer.

When a demand is made, a query is formulated. The list of potential suppliers retrieved is further filtered and sorted by a ranking model. Overall, for this problem statement, the input space consists of a list of candidate suppliers associated with a demand (query), the hypothesis includes a scoring function, a loss function based on evaluation measure, and the output space contains the ranked list of suppliers.

BM25 [Rob97] is a traditional ranking function with its share of advantages and disadvantages. When machine learning is used for ranking, it is a Learning to Rank (LTR) [Liu11] problem. LTR can be solved using one of the three approaches, viz. Pointwise, Pairwise, and Listwise. In this thesis, the advantages of the BM25 model are exploited in the feature generation part of the ranking architecture. These generated features are then used to build machine learning, and deep learning models for the three LTR approaches.

The ranking problem is also solved by the Google search engine where, given a query, a list of websites is displayed, with the topmost one being the most relevant concerning the query [PBMW99]. On the other hand, the e-commerce business Amazon deals with the ranking of the products when a customer queries it with a focus on sales conversion along with relevancy [P18]. Both cases address ranking in information retrieval, and similar to most IR systems, they lack an explanation for the provided rankings. Thus, leaving transparency out of the process. According to the SWIRL report [CDS18] and [OGGdR+21], transparency is one of the issues that must be addressed by the IR community. Hence, one of the areas of focus for this thesis is explainability, which would explain why and how a particular item was ranked at a particular position. The insights gained from the explainability feature will not only provide transparency in the entire ranking process for the stakeholders, making the ranking process more reliable but can also be leveraged in guiding further ML-based feature generation and modeling by the developers.

1.1 Motivation

The data-driven intelligent systems nowadays rely less on hard-coded rules and more on examples and human feedback. These humans are, in most cases, non-technicians but can be domain experts. The technicians or programmers build a bridge between human feedback and the intelligence that drives the data-driven applications. "A set of strategies that combine human and machine intelligence is called human-in-the-loop machine learning" [Mon21]. The human-in-the-loop contributions can assist the machine learning models, increase the accuracy of the machine learning models, and make machine learning models more efficient and reliable.

The problem this thesis is trying to solve is that given a demand and a list of potential suppliers, the ranking model should provide an ordered list of the suppliers, with the most relevant suppliers at the top. The problem statement is denoted in the first three blocks of the Fig. 1.1

Figure 1.1: Ranking model with human-in-the-loop. The demand consists of demand_properties and an associated list of suppliers, which is consumed by the ranking model. The ranking model sends the ranked suppliers based on the demand to the domain experts, who will then provide the curated supplier list to the customer.

However, the top N suppliers are not directly considered as the final list of the suppliers for the given demand. Instead, domain experts go through the ranked list and mark the suppliers as relevant and irrelevant until the required number of suppliers is found. These domain experts act not only as the stakeholders but also as the human-in-the-loop for the ranking model. Their significant contribution is to make the ranking model more accurate by annotating the suppliers, which further generates labeled training data. The primary motivation of this thesis is to reduce human efforts by building a precise ranking model.

Learning to Rank (LTR) for information retrieval is an approach to solving information retrieval tasks using machine learning, and many surveys have been done in the field of LTR for information retrieval [HWZL08], [CZH+12], [GDR16]. These surveys provided a direction for the thesis to experiment with various ranking approaches in an attempt to address the aforementioned problem statement.

Explainability in machine learning (xAI) [GA19] deals with making the machine learning models more transparent [ADRDS+20]. It has helped to ensure impartiality in decision-making, directed the provision of robustness, and to some extent provided trustworthiness [ADRDS+20]. These three points have been surveyed already [AB18]. There already exists some research surrounding the point of obtaining insights from explainability [LEC+20], [KWG+18], [LNV+18]. There is not much research available when it

comes to explaining the rankings predicted by the ranking models apart from [VG19], [ZWB+20], [SWKA20].

Considering the example with the demand for rubber tyres, additional criteria for suppliers operating at a particular location may be demanded. This requirement for a product at a particular location makes the demand multi-faceted, and providing explanations becomes even more necessary. This motivated the thesis to explore explainability in ranking to some extent. Self-explaining models are the models in which explainability plays a key role already during learning [AMJ18]. In this thesis, the generation of self-explanatory features provides a way to build self-explaining models. Explainability and self-explanatory features will not only be used to provide transparency to the domain experts but will also prove as a deciding factor in building the next version of the ranking model, and this will provide an efficient data-driven solution in curating the final long list of suppliers.

1.2 Goals

Revisiting the previous sections, the main goal of the thesis is to build a ranking model that could reduce the manual efforts required to curate the long list of suppliers for a particular demand. Ideally, the ranking model should be so efficient and reliable that the role of domain experts in the Fig. 1.1 is completely eradicated. However, it would be an over-ambitious goal. Hence the narrowed-down goal of this thesis is to build a ranking model that provides the majority of the relevant suppliers at the top, which would reduce the manual efforts put in by the domain experts in reviewing the suppliers.

Although machine learning models have provided much impetus to data-driven applications, there is a demand for explainability [PHB+18]. This brings us to the second goal of the thesis, which is to answer why a ranking model assigned a particular supplier with a specific relevancy score, in the ranked list of suppliers, for a particular demand.

This thesis is not trying to reinvent the wheel but is built on top of the research already conducted in the research area of LTR for information retrieval with an add-on of explainability. There already exists a plethora of theoretical as well as experimental surveys for LTR for information retrieval. However, most of them are done on benchmark datasets. The major challenge with benchmark datasets is that they may not capture all the nuances in the model compared to the real-life data that drives the intelligence of data-driven applications. Thus, this thesis will capture the pros and cons of different LTR approaches when applied to real-life data.

Conventional machine learning and deep learning models will be experimented with, with all the three Learning to rank (LTR) approaches (Pointwise, Pairwise, and Listwise). Furthermore, SHAP (SHapley Additive exPlanations) [LL17] values will be plotted for each model to interpret their predictions. A systematic overview of LTR using conventional machine learning models, LTR using deep learning models, and xAI in the form of SHAP plots and exploitation of the self-explainability of the ranking model will be done in this thesis. Along with this, the results provided by each ranking model will be critically discussed and analyzed.

The thesis not only aims to build a model-driven solution but also a product-oriented one [KKH22]. Hence, this thesis touches on the machine learning lifecycle [ZCD+18]. There is a lot of hidden technical debt that needs service when it comes to Machine learning models [SHG+15]. One way to overcome one of the debts is by tracking all the ML experiments in such a way that, at any given time, any model whose results are interesting can reproduce the behavior. Along with this, a thorough analysis of all the results with the domain experts in the loop motivated the third goal of this thesis to develop a product-oriented solution.

To summarize, the work presented in this thesis aims to leverage the explainability of all the experimented ranking models, with each ranking model being easily reproducible and maintainable, to gain further insight towards improving the ranking model—these goals led to the formulation of the following research questions.

1.3 Research Questions

"Occam's razor is the problem-solving principle that entities should not be multiplied beyond necessity" [Hey97]. It can be approximately paraphrased as, 'always go with the simplest approach in case of equally good solutions.' When it comes to machine learning, what is more important than choosing a powerful algorithm, is to know what to include and what to not in the model. In this work, this means that even if the simple ranking models do not out-perform the complex ranking models, they will at least perform similarly to the complex ranking models. This will be answered by the following research question.

- **RQ1** Will simpler models with fewer features yield better explainable results and thus be preferred in LTR problems with human-in-the-loop?

In this era, it is not sufficient to just build an accurate model but a model which could answer the question 'Why this prediction?' to some extent. To assist the domain experts, just providing a ranked list of suppliers is not enough. A ranked list of suppliers and some intuition as to why the ranking model considered a particular supplier to be at a particular position in the ranked list is required. The previously mentioned goal of the thesis to provide transparency in the ranking pipeline is particularly interesting. This is because if this goal is achieved, the benefit of answering the following research question will accrue.

- **RQ2** Can gained explainability be leveraged to improve the feature generation of the ranking model?

In most of the tasks, building a generalized machine learning model which is universally valid is very difficult [SPM19]. Hence, various ensemble techniques are implemented. It would be interesting to see if one generalized ranking model is enough to solve the demand-supplier ranking problem. This formulates into the following research question.

- **RQ3** Can one ranking model generalize well on all of the demand-supplier sets?

1.4 Research Methodology

The data used in this thesis consists of demand with an associated list of suppliers. Each of the suppliers is assigned a label as relevant or irrelevant. The task is to use this binary-labeled data to learn the ranking of the suppliers concerning a demand.

Various models trained on differently sampled data with different sets of features are experimented with to solve the aforementioned ranking problem. These sets of features may differ in the following ways, the way the features were generated in a set, whether or not a feature selection algorithm was applied to them, and the number of features in a set. To answer the **RQ1**, models trained on different sets of features are compared. Each model is also passed to the explainability module that will state the importance of the features for the respective model. This knowledge about feature importance will be exploited to gain further insights for feature generation, and this will be an attempt to answer the **RQ2**. Each demand is different in a certain way. Hence, a granular level analysis of the outcomes obtained from the ranking models is conducted on the demands on which the various ranking models were tested. The finding from this analysis can be used to answer the **RQ3**.

1.5 Thesis Outline

Describe the structure of this document.

2 Related Work

This section gives this thesis a steady piece of land to stand on, as the vast landscape of various experiments to solve the aforementioned ranking problem is explored.

'Learning to Rank for Information Retrieval' [Liu11] can be considered the holy grail for LTR. This literature laid a strong foundation for building this thesis. Its content ranges from the introduction of the ranking problem and defining LTR approaches to implementing the various LTR approaches on the LETOR dataset [TTJ+07]. The performance of various approaches on the LETOR dataset is drilled down, which allows comparing these findings with the findings of this thesis which uses real-world data. Apart from this, one of the tasks mentioned as future work was giving more importance to feature engineering. This gave a direction to this thesis which has feature engineering and features analysis as the major area of focus.

Addressing the task of feature engineering in the future work of the above-mentioned tutorial [Liu11], a survey on feature selection for LTR has been studied in [SK18]. The paper divides the feature selection approaches into three categories, i.e., Filter-based, Wrapper, and Embedded Methods. Some of the algorithms belonging to each of the categories have also been explained in depth in this paper. The paper explains all the feature selection approaches theoretically and shows the effect of feature selection approaches on the OHSUMED dataset [HBLH94] where RankSVM [Joa06] was used as an LTR approach. In this thesis, some of the feature selection methods described in [SK18] have been implemented, and the effect of feature selection methods on the traditional machine learning based pointwise LTR approach is shown. This led to the choosing of a new approach to solve the LTR problem.

Selecting an algorithm to implement the various LTR approaches needed thorough research. One of the research works is conducted by Ibrahim et al. in the paper 'Comparing pointwise and listwise objective functions for random-forest-based learning-to-rank [IC16]. This paper compared random forest based pointwise, listwise, and a hybrid LTR approach extensively on six smaller to moderate size datasets and two large datasets using NDCG@10 and MAP. In some cases, random forest listwise and random forest hybrid approaches outperformed random forest pointwise. However, the performance difference was not so significant except in some cases of random forest hybrid. Experimenting with hybrid approaches is beyond the scope of this thesis. Hence, the random forest based pointwise approach is only explored in this thesis.

Selecting the algorithm for the pairwise approach is based on [QKF20], where Qomariyah et al. solved a personalized recommendation task using various learning to rank approaches for the IMDB movies dataset by Kaggle. In the experiments, the pairwise approach outperformed the pointwise and the listwise approaches. All the approaches were implemented using the ranking module in XGboost [CHB+15]. In this

thesis, the pairwise approach, which is LambdaMART [Bur10] is implemented using XGboost.

Google released the open-source library, Tensorflow Ranking [PBW+19] which claims to solve large-scale LTR problems and is deployed into production for solving some ranking tasks at Google. In this thesis, this deep learning based framework for LTR is used to implement the pointwise, pairwise, and listwise approaches using Neural networks.

Like the machine learning models, while building the ranking models, hyperparameter tuning of the models is done. Van et al. [VRH17] demonstrated the importance of hyperparameters while building Random forest and Adaboost in a classification setting. They experimented with the importance of the respective hyperparameters for the two ML models across 100 datasets. They concluded their work by jotting down the important hyperparameters, and this thesis deals with random forest. So the hyperparameters that are selected for tuning the random forest model are the top four hyperparameters as per the conducted survey on hyperparameters. However, not much relevant research has been based on hyperparameter tuning pairwise and listwise approaches.

Another issue that needs to be tackled is the class imbalance nature of the datasets, as a small fraction of the available information is relevant. In the paper [IC14], the imbalance nature of LTR opensource benchmark datasets has been investigated. In an attempt to reduce the training time which is considered, a focused investigation of undersampling of irrelevant documents using a random approach and a more deterministic approach is conducted, which reduces the amount of training data and thus an overall reduction in the training time. Even though, in this thesis, training time reduction is not a priority to solve, the class imbalance nature of the opensource benchmark dataset for LTR tasks is very well reflected in the demand-supplier dataset that is consumed by the ranking model. That being said, *out-of-the-box* ML engineering problems like class imbalance should be used to improve the quality of the training data, which in turn will lead to the development of more accurate and more reliable ranking models is one of the main focus of this thesis. Implying LTR-based ranking models using various sampling techniques are thoroughly compared.

As the title of this thesis suggests leveraging the explainability of the ranking models, the last component of this thesis is the xAI module. Interpretability of Machine learning models is like flossing. Everybody knows it is important, but only some do it. Christoph Molnar [Mol18] has classified machine learning interpretability into two types, post hoc, where interpretability techniques are applied after training the machine learning models, and intrinsic, where self-explanatory Machine learning models are built. This classification forms the basis of [DLH19] where a further clear distinction between local and global interpretability is explained, and [ZWB+20] where an intrinsically interpretable LTR model is built using Generalized additive models(GAMs). In this thesis, we try to incorporate explainability as explained in these papers.

3 Concepts

This chapter gives an overview of all the concepts starting from defining information retrieval, one of the most traditional ranking functions BM25, to recent ranking methods using LTR. Further it explains the evaluation metrics and is concluded with the explanation of the SHAP values calculation.

3.1 Information retrieval

Finding the information of interest from this plethora of data could have been a never-ending task had it not been for systems that find relevant information. In the context of this thesis, this system is an information retrieval system.

"Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)" [MRS10].

In simple terms retrieving relevant information as per the need is information retrieval. To retrieve relevant information, a scoring mechanism is needed, which is done by ranking in information retrieval.

Conventional ranking models in IR can be divided into two types, Query-dependent ranking models that retrieve documents based on the presence of the query terms in the documents and Query-independent ranking models that rank documents based on their self-importance [MRS10]. In this thesis, the problem statement is to rank suppliers based on the demand, where a demand corresponds to a query and the suppliers correspond to the documents. Implying an interest on Query dependent ranking models. The most basic model is the Boolean model [BYRN+99] which will just answer whether the supplier is relevant or not for the given demand with a limitation of not being able to answer how much the supplier is relevant for the demand. This is solved by Vector Space Models (VSM) [BYRN+99] in which the suppliers and the demands are represented as vectors with $Tf - IDF$ weighting.

$Tf(t, p)$ =number of times term t occurs in a supplier p

$IDF(t) = \log \frac{S}{sf_t}$

S =total number of webpages

sf_t = number of webpages with term t

The assumption of term independence is overcome by Latent Semantic Indexing (LSI) [DDF+90] which maps the vector representations to a latent space using Singular Value Decomposition (SVD). The following section explains the probabilistic ranking model

BM25 [Rob97] which does not use the ground truth at all, which is used for feature generation in this thesis.

3.2 BM25

BM25 [Rob97], as a scoring method, allows sorting by relevancy based on the score given by the equation 3.1.

Given a demand d, containing the terms t_1,....,t_N, the BM25 score for supplier webpage p is calculated as follows:

$$BM25(d,p) = \sum_{i}^{N} \frac{IDF(t_i).Tf(t_i,p).(k_1+1)}{Tf(t_i,p) + k_1.(1-b+b.\frac{Len(s)}{avsl})} \tag{3.1}$$

where $Len(s)$ =number of words in a webpage s

$avsl$ =average length(number of terms) of webpage

b and k_1 are free parameters

In conventional IR models, there are free parameters that need to be tuned. Consider the equation 3.1 of BM25 model, b and k_1 need to be tuned on a validation set. However, to tune them is not so trivial considering the fact that the IR evaluation metrics 3.6 are non- differentiable and non-continuous w.r.t the parameters. The traditional information retrieval approaches don't make use of feedback. In our case, the feedback from the domain experts is present in the form of whether a supplier is relevant or irrelevant for a demand. Machine learning has proven to be effective when it comes to automatically tuning parameters, and hence, applying machine learning to solve the aforementioned ranking task seems reasonable.

3.3 Machine learning

"Machine learning is the study of computer algorithms that allow computer programs to automatically improve through experience" [MM97].

Based on the availability of the ground truth, machine learning can be classified into supervised (when the ground truth is present) and unsupervised (when the ground truth is absent). The demand-supplier data consists of ground truth in the form of relevancy of the supplier with respect to the demand, narrowing down the usage of machine learning to supervised learning. Deep learning is a subfield of machine learning that uses neural networks. In this thesis, all the algorithms are divided into neural networks based, and the ones which do not use neural networks for learning are considered traditional/conventional machine learning based.

3.3.1 Neural Networks

Artificial neural networks, also known as neural networks and the most basic unit of a neural network, i.e., perceptron [Ros58] were proposed by Rosenblatt, the pioneer of neural networks, in 1958. A single layer perceptron for one instance s_1 from the Table 3.3 is shown in the Fig. 3.1, the equation of which is:

$$\widehat{y} = g(W^T s_1) \tag{3.2}$$

where W is a vector of all the weights $w_{n=1,2,3..5}$ and s_1 is vector of dimension n and s_{11} denotes the value of f_1 in s_1, and g is an activation function that "defines the output of a node given the input".

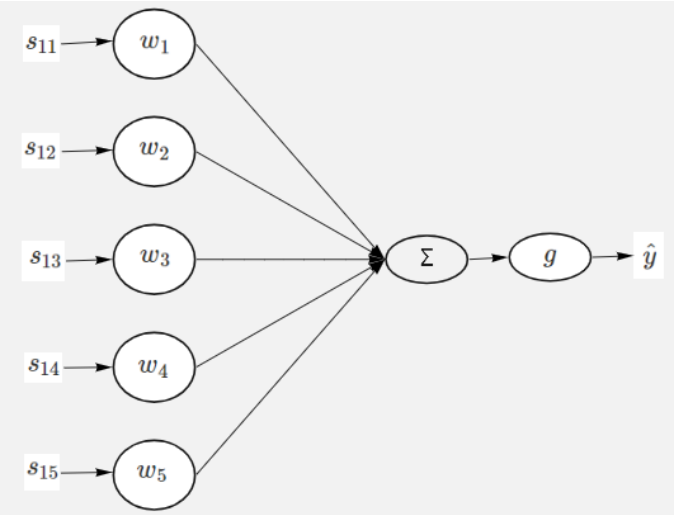


Figure 3.1: Single layer perceptron

A single layer perceptron has a limitation that it cannot solve non-linearly separable problems, which can be very well proved by taking the example of the XOR function, which cannot be represented by a single layer perceptron [GBC16]. This led to a solution in the form of a multi-layer perceptron, which is shown in the Fig. 3.2. The equation to get the output $\widehat{y}$ by the multi-layer perceptron, which is an extension of eq. 3.2 is as follows: (For simplicity $W = W^T$)

$$\widehat{y} = g(W^{out} g(W^1 g(W^0 s_1))) \tag{3.3}$$

where W^l is the weight for the layer *l-1* and in our example there is one input layer $l = 0$, one output layer $l = out$ and two hidden layers $l = 1$ and $l = 2$. g is a non-linear activation function as stacking linear functions will make the entire network linear. The most common non-linear activation functions are *sigmoid*, where $g(z) = \frac{1}{1+e^{-z}}$, *tanh*, where $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$. However, the default non-linear activation function in most of the systems is *relu*, where $g(z) = max\{0, z\}$ as it is closely linear [GBC16]. For simplicity, the bias b is not considered. However, after adding bias, the equation will become,

$$\widehat{y} = g(W^{out} g(W^1 g(W^0 s_1 + b^0) + b^1) + b^{out}) \tag{3.4}$$

Multi-layer perceptrons are also known as feedforward neural networks, and these are the

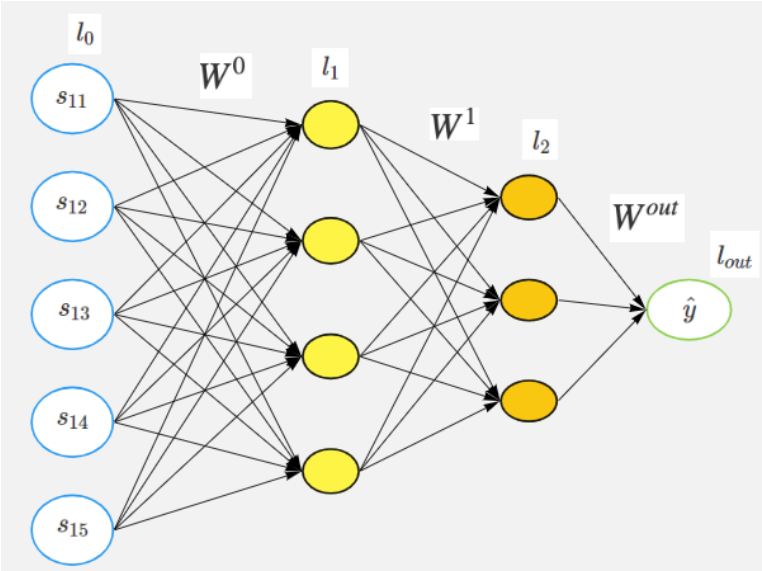


Figure 3.2: Multi-layer perceptron

type of neural networks that are explored in this thesis. All the neural networks in this thesis are implemented using the open-source library TensorFlow (TF) 2.0 [AAB+15].

When machine learning is applied to build a computer program that would rank the suppliers according to the demand based on previous demand-supplier data, it can be said that the suppliers are ranked using the Learning to Rank (LTR) approach. The various LTR approaches are explained below after explaining the ranking problem this thesis is trying to solve.

3.4 Bipartite ranking

In the Bipartite ranking problem, instances belong to either of the binary classes (+ (positive) and - (negative)), and the aim is to learn a ranking function that scores the + positive instances with higher values than the - negative ones [Aga05]. The following example differentiates the bipartite ranking problem and the classification problem.

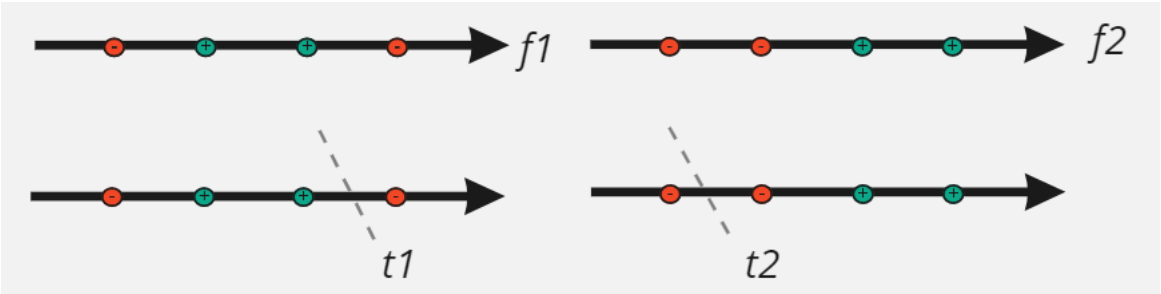


Figure 3.3: Bipartite ranking vs. Classification. The arrow is heading towards high. There are 2 positive and 2 negative instances and two functions *f1*, *f2* which rank them and classify them with thresholds *t1* and *t2* respectively

Considering the example in Fig. 3.3, if *f1* and *f2* were classifiers then the classification error for both of them would be 1/4. However, when it comes to ranking, *f2* is better than *f1* as it has positive instances at a higher rank than the negative instances.

In this thesis, we are dealing with bipartite ranking.i.e. in the demand-supplier data, the suppliers have binary library 1,0 to denote their relevancy to a demand.

3.5 Learning to Rank (LTR)

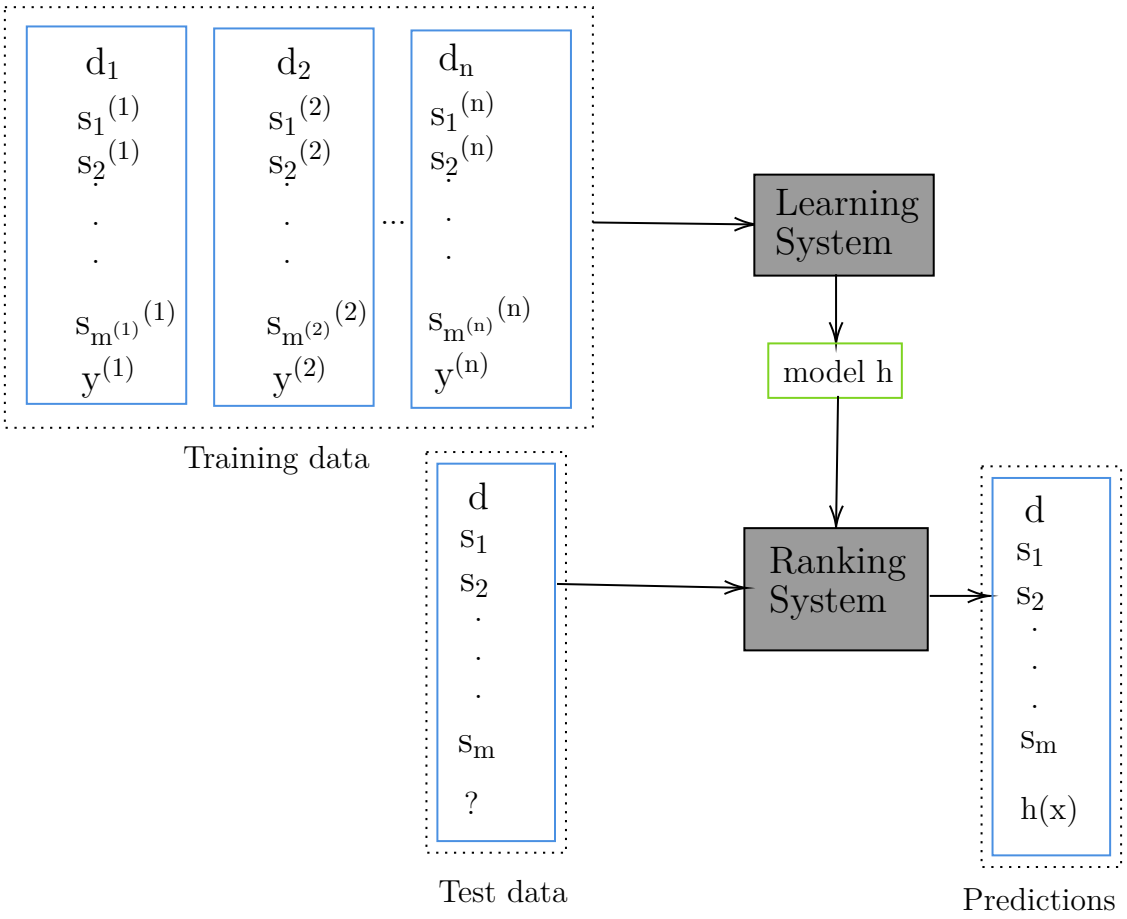


Figure 3.4: LTR Framework [Liu11]. The training data consists of n demands $d_i (i = 1, ..., n)$ and associated supplier vectors $s^{(i)} = \left\{s_j^{(i)}\right\}_{j=1}^{m^{(i)}}$ $m^{(i)}$ is the number of suppliers associated with a demand d_i. $y^{(i)}$ denotes the relevancy of the suppliers. A learning system is used to obtain the model hypothesis h from the hypothesis space. The hypothesis h is implemented along with a scoring function f. $h(s) = sort(f(s))$. The scoring function provides a relevancy score to each supplier, and this score is used for sorting the suppliers to obtain the ranked list of suppliers.

Restating the definition of LTR, LTR is applying machine learning to solve the ranking problem. [Liu11] states that the two important properties for any ranking method to be classified as a Learning to Rank method are:

- Feature based

  Each supplier is represented by feature vectors. In LTR methods, an optimal way of combining these features is learned.

- Discriminative training

  The four main components of discriminative training, as mentioned in [Mit99] are the *input space* that consists of a list of suppliers (represented by a set of features) associated with a demand, *output space* that consists of binary labels 0,1 with value as 1 when supplier is relevant for the demand and 0 when supplier is not relevant for the demand, *hypothesis space* that defines a class of functions that provides a mapping between the *input space* and the *output space* and *loss function* that measures the equality between the predictions given by the hypothesis and the ground truth. Each LTR method has these four components.

Fig. 3.4 depicts the overall workflow of the LTR method for ranking suppliers given a demand. The learning system can be built using various learning-to-rank algorithms. Based on the four components of discriminative training, the LTR approaches are divided into three categories, pointwise, pairwise, and listwise. These three approaches are explained theoretically and with the help of a toy example below.

Consider the following toy example depicted in Table 3.1 in which there are two demands where one demand has three suppliers, and another demand has four suppliers. This data mimics the real-world demand-supplier data on which all the experiments in this work are conducted. The relevancy of each supplier is given in terms of binary labels. The

demands	Suppliers	relevancy
d_1	$s_1^{(1)}$	1
	$s_2^{(1)}$	1
	$s_3^{(1)}$	0
d_2	$s_1^{(2)}$	0
	$s_2^{(2)}$	1
	$s_3^{(2)}$	1
	$s_4^{(2)}$	0

Table 3.1: Toy example to explain LTR approaches. IT consists of two demands denoted by d_1, d_2. For a demand d_i, the associated supplier is given by $s_j^{(i)}$, where the lower bound of j is one and the upper bound is number of suppliers associated with the demand. The ground truth for a supplier to be relevant or irrelevant for the demand is given by relevancy and consists of binary labels 0,1 where 0 denotes the supplier being irrelevant and 1 denotes the supplier being relevant.

demands d_1, d_2 contain the features (text data) as required by the customer to obtain the suppliers which will fulfill those requirements. The suppliers contain the text data which is nothing but the website content of the supplier.

Using the demand and content on the website of the suppliers, let's say five features are handcrafted. The outcome is represented in the following Table 3.2:

demands	Suppliers	f_1	f_2	f_3	f_4	f_5	relevancy
d_1	$s_1^{(1)}$	0.15	0.18	0.12	0.08	0.17	1
	$s_2^{(1)}$	0.19	0.24	0.06	0.19	0.20	1
	$s_3^{(1)}$	0.07	0.05	0.25	0.16	0.14	0
d_2	$s_1^{(2)}$	0.12	0.13	0.12	0.13	0.15	0
	$s_2^{(2)}$	0.16	0.16	0.06	0.19	0.14	1
	$s_3^{(2)}$	0.13	0.12	0.06	0.16	0.09	1
	$s_4^{(2)}$	0.15	0.10	0.31	0.06	0.07	0

Table 3.2: 5 handcrafted features $\{f_i\}_{i=1,..,5}$for each demand in the toy example 3.1

Pointwise

In the pointwise approach, given a demand, each supplier is scored independent of other suppliers. As the ground truth in this thesis is binary, this task can be considered a classification task. After training the classifier, during prediction, instead of classifying the supplier as relevant or irrelevant, the probability of assigning the supplier to the relevant class is used as a score for that particular supplier. This score is used to decide the position of the supplier in the ranked list of suppliers. This approach does

Instance	f_1	f_2	f_3	f_4	f_5	relevancy
s_1	0.15	0.18	0.12	0.08	0.17	1
s_2	0.19	0.24	0.06	0.19	0.20	1
s_3	0.07	0.05	0.25	0.16	0.14	0
s_4	0.12	0.13	0.12	0.13	0.15	0
s_5	0.16	0.16	0.06	0.19	0.14	1
s_6	0.13	0.12	0.06	0.16	0.09	1
s_7	0.15	0.10	0.31	0.06	0.07	0

Table 3.3: Pointwise approach data where no association between the suppliers and the demand is seen, unlike the data in Table 3.2

not consider the association of suppliers to a particular demand. Hence irrespective of the demand, each row becomes an instance which is shown in the instance column in Table 3.3

Pairwise

In the pairwise approach, pairs of suppliers belonging to a particular demand are used to learn the ranking. This approach models the ranking as a pairwise classification task. This is done by swapping every supplier that belongs to a particular demand, and the impact of this swap is used to calculate the new set of labels on which a regressor will be fit. This impact can be calculated by taking any ranking evaluation metric 3.6. This transformation is applied to the toy example 3.2. This calculated impact is denoted as lambda in the transformed table. The lambda for all the instances is initialized to 0, and all the suppliers are sorted as per their relevancy. This is shown in the Table 3.4.

demands	Sorted Suppliers	f_1	f_2	f_3	f_4	f_5	relevancy	lambda
d_1	$s_1^{(1)}$	0.15	0.18	0.12	0.08	0.17	1	0
	$s_2^{(1)}$	0.19	0.24	0.06	0.19	0.20	1	0
	$s_3^{(1)}$	0.07	0.05	0.25	0.16	0.14	0	0
d_2	$s_2^{(2)}$->$s_1^{(2)}$	0.16	0.16	0.06	0.19	0.14	1	0
	$s_3^{(2)}$->$s_2^{(2)}$	0.13	0.12	0.06	0.16	0.09	1	0
	$s_1^{(2)}$->$s_3^{(2)}$	0.12	0.13	0.12	0.13	0.15	0	0
	$s_4^{(2)}$	0.15	0.10	0.31	0.06	0.07	0	0

Table 3.4: Pairwise approach data. All the suppliers associated with demand are sorted in Table 3.2 wrt. to relevancy such that for two suppliers s_j^i and s_k^i belonging to a demand d_i,$j > k$, if relevancy of s_j^i is 1 and relevancy of s_k^i is 0. The impact column is denoted by lambda, which is initialized to 0

For simplicity the impact is calculated using prec@k 3.7 with k=2. The algorithm for transformation is as follows:

1: for each demand d_i,
2: sort the suppliers based on relevancy
3: for each supplier s_{ij},
4: for each supplier s_{ik},
5: **if** k>j **then**
6: swap(s_{ij},s_{ik})
7: calculate $prec$@2 after this swap
8: $prec@2diff = ideal_{prec@2} - prec@2$
9: $lambda[s_{ij}] += prec@2diff$
10: $lambda[s_{ik}] -= prec@2diff$
11: **end if**

For demand d_1, the ideal prec@2 is 1.0 and if supplier $s_1^{(1)}$ is swapped with $s_2^{(1)}$, the prec@2 remains unchanged i.e. 1.0. The difference in the ideal prec@2 and the computed prec@2=1.0-1.0=0, hence the lambda still remains 0. If supplier $s_1^{(1)}$ and $s_3^{(1)}$ are swapped, the prec@2 changes to 0.5. The difference between the ideal prec@2 and computed prec@2 is 0.5 and lambda for $s_1^{(1)}$ becomes 0+0.5=0.5 and for the supplier $s_3^{(1)}$ it is 0-0.5=-0.5. When $s_2^{(1)}$ is swapped with $s_3^{(1)}$, the prec@2 is 0.5 which differs from the ideal prec@2 by 0.5 and the updated lambda for $s_2^{(1)}$ is 0+0.5=0.5 and for $s_3^{(1)}$, it is -0.5-0.5=-1.0. This is also done for demand d_2 and the transformed table is represented in the Table 3.5. A regression tree is then fit on this transformed data and tries to predict 'lambda.' And when multiple regression trees are fit, the whole algorithm of LambdaMART is formed where MART stands for Multiple Additive Regression Trees and 'lambda' in LamdaMART [Bur10] denotes this impact.

The limitation of the pairwise approach is that the relative ordering between only two suppliers is considered, which may fail to capture the ordering of all the suppliers in the final ranked list.

demands	Sorted Suppliers	f_1	f_2	f_3	f_4	f_5	relevancy	lambda
d_1	$s_1^{(1)}$	0.15	0.18	0.12	0.08	0.17	1	0.5
	$s_2^{(1)}$	0.19	0.24	0.06	0.19	0.20	1	0.5
	$s_3^{(1)}$	0.07	0.05	0.25	0.16	0.14	0	-1
d_2	$s_1^{(2)}$	0.16	0.16	0.06	0.19	0.14	1	0.5
	$s_2^{(2)}$	0.13	0.12	0.06	0.16	0.09	1	0.5
	$s_3^{(2)}$	0.12	0.13	0.12	0.13	0.15	0	-1
	$s_4^{(2)}$	0.15	0.10	0.31	0.06	0.07	0	-1

Table 3.5: Pairwise approach data. Transformation of Table 3.4 after applying the transformation algorithm 11

demands	Instance	Suppliers	f_1	f_2	f_3	f_4	f_5	relevancy
d_1	1	$s_1^{(1)}$	0.15	0.18	0.12	0.08	0.17	1
		$s_2^{(1)}$	0.19	0.24	0.06	0.19	0.20	1
		$s_3^{(1)}$	0.07	0.05	0.25	0.16	0.14	0
d_2	2	$s_1^{(2)}$	0.12	0.13	0.12	0.13	0.15	0
		$s_2^{(2)}$	0.16	0.16	0.06	0.19	0.14	1
		$s_3^{(2)}$	0.13	0.12	0.06	0.16	0.09	1
		$s_4^{(2)}$	0.15	0.10	0.31	0.06	0.07	0

Table 3.6: Listwise approach data. The training set consists of $\left\{(s^{(i)}, relevancy^{(i)})\right\}_{i=1}^{2}$ where $s^{(i)} = \left\{s_j^{(i)}\right\}_{j=1}^{m^{(i)}}$ and $relevancy^{(i)} = \left\{relevancy_j^{(i)}\right\}_{j=1}^{m^{(i)}}$; $m^{(i)}$ denotes the number of suppliers associated with demand d_i. Hence, all the three suppliers belonging to demand d_1 form instance 1 and the four suppliers belonging to demand d_2 form instance 2

Listwise

In the listwise approach, the aim is to determine the optimal ordering of all the suppliers associated with their respective demands. In this approach, the loss function considers the positions of suppliers in the ranked list of all the suppliers belonging to the same demand group, unlike pointwise and pairwise.

In the pairwise approach, the training data is transformed, and then the learning takes place, whereas in the listwise approach, the transformation of ground truth and the predicted scores takes place, and then the learning takes place. The scores of each supplier given by the model and the relevancy of each supplier as marked by the domain experts are transformed into a probability distribution. The learning happens with the aim of minimizing the difference between the two probability distributions. Permutation probability and top one probability are the two models for the transformation of the scores.

Referring to Table 3.6, the transformations of the scores for the suppliers associated with demand d_1 are shown below:

Therefore, the suppliers to rank are $s^{(1)} = (s_1^{(1)}, s_2^{(1)}, s_3^{(1)})$ The number of possible permutations for these three suppliers are *3!* and are as follows:

$(s_1^{(1)}, s_2^{(1)}, s_3^{(1)})$

$(s_1^{(1)}, s_3^{(1)}, s_2^{(1)})$

$(s_2^{(1)}, s_1^{(1)}, s_3^{(1)})$

$(s_2^{(1)}, s_3^{(1)}, s_1^{(1)})$

$(s_3^{(1)}, s_1^{(1)}, s_2^{(1)})$

$(s_3^{(1)}, s_2^{(1)}, s_1^{(1)})$

The six permutations above are denoted by Ω_6, π denotes a single permutation, and $\pi(k)$ is the supplier at position k in the respective permutation. Let $z = (z_1, z_2, ...z_n)$ depict the score given by a model to each of the suppliers in π, and z_j is the score given by the model for the supplier at j^{th} position. The probability of one of the permutations is given by the formula:

$$P_z(\pi) = \prod_{j=1}^{m^{(i)}} \frac{\phi(z_{\pi(j)})}{\sum_{k=j}^{m^{(i)}} \phi(z_{\pi(k)})} \tag{3.5}$$

where $m^{(i)}$ is the number of suppliers in one permutation set, and for simplicity, $\phi(x) = e^x$

Assuming the scores predicted for the three suppliers by a model as 1.62 for $s_1^{(1)}$, -0.61 for $s_2^{(1)}$, and -0.53 for $s_3^{(1)}$ and based on the eq. 3.5, the probability of permutation for $(s_1^{(1)}, s_2^{(1)}, s_3^{(1)})$ is:

$P_z((s_1^{(1)}, s_2^{(1)}, s_3^{(1)})) = \prod_{j=1}^{3} \frac{e^{z_{\pi(j)}}}{\sum_{k=j}^{3} e^{z_{\pi(k)}}}$

$P_z((s_1^{(1)}, s_2^{(1)}, s_3^{(1)})) = \prod_{j=1}^{3} \frac{e^{z_{\pi(j)}}}{\sum_{k=j}^{3} e^{z_{\pi(k)}}} = \frac{e^{s_1^{(1)}}}{e^{s_1^{(1)}} + e^{s_2^{(1)}} + e^{s_3^{(1)}}} \cdot \frac{e^{s_2^{(1)}}}{e^{s_2^{(1)}} + e^{s_3^{(1)}}} \cdot \frac{e^{s_3^{(1)}}}{e^{s_3^{(1)}}} = 0.39$

Similarly, calculating it for the remaining five sets, the permutation probability is as follows:

$(s_1^{(1)}, s_2^{(1)}, s_3^{(1)}) -> 0.39$

$(s_1^{(1)}, s_3^{(1)}, s_2^{(1)}) -> 0.42$

$(s_2^{(1)}, s_1^{(1)}, s_3^{(1)}) -> 0.078$

$(s_2^{(1)}, s_3^{(1)}, s_1^{(1)}) -> 0.009$

$(s_3^{(1)}, s_1^{(1)}, s_2^{(1)}) -> 0.085$

$(s_3^{(1)}, s_2^{(1)}, s_1^{(1)}) -> 0.009$

The top one probability for $s_2^{(1)}$ is 0.07+0.009=0.087. However, calculating permutation probability can be practically inefficient; hence in the paper [CQL+07], top one probability was proposed using the following equation:

$$P_z(j) = \frac{e^{z_j}}{\sum_{k=1}^{m^{(i)}} e^{z_k}} \tag{3.6}$$

For $s_2^{(1)}$, the top one probability using the equation 3.6 is 0.087. Similarly the top one probabilities of $s_1^{(1)}$ and $s_3^{(1)}$ are 0.82 and 0.095. The probability distribution of the ground truth which is given by relevancy is also calculated using $relevancy^{(1)} = (relevancy_1^{(1)}, relevancy_2^{(1)}, relevancy_3^{(1)}) = (1, 1, 0)$. The two probability distribution for predicted scores and the relevancy scores is shown in Fig. 3.5. Any loss function that measures this difference in distribution can be used, like Kullback- Leibler (KL) divergence. In ListNet [CQL+07], the model is a neural network, listwise loss cross-entropy is used,

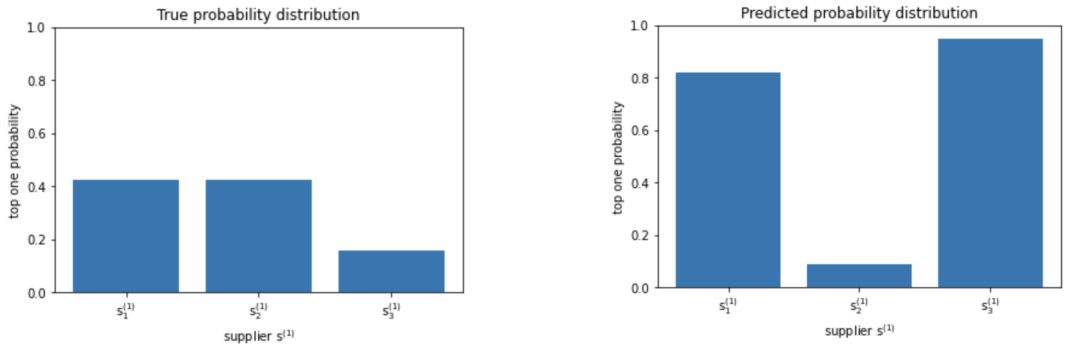


Figure 3.5: Probability distribution true vs. predicted

and Gradient descent is the algorithm.

The LTR approaches are summarized in the Fig. 3.6 and Table 3.7.

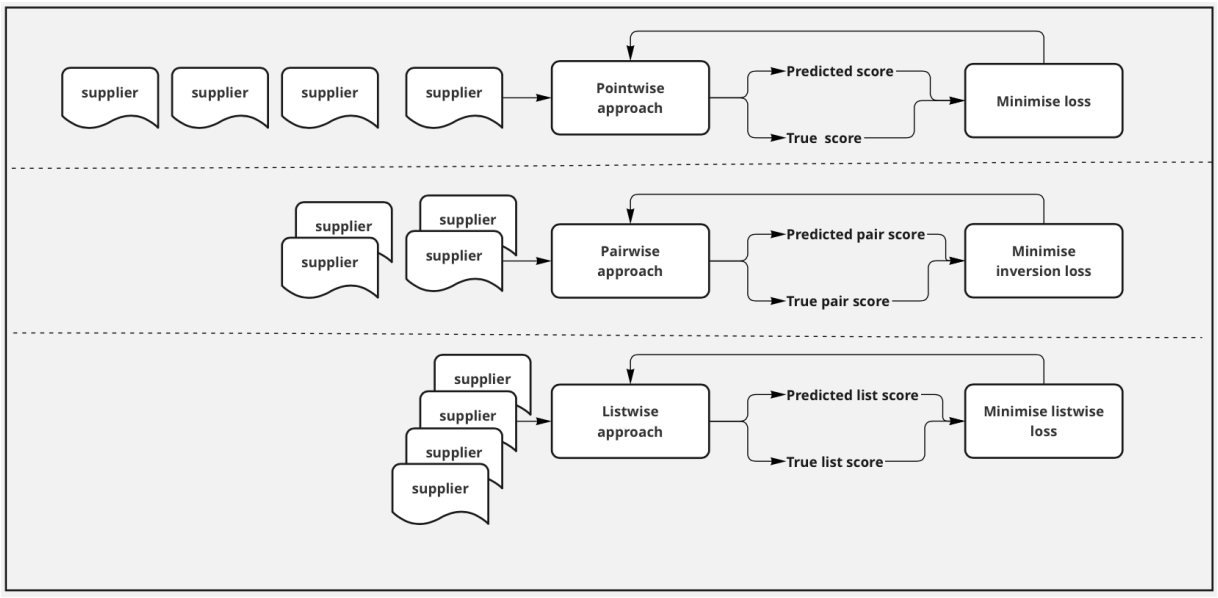


Figure 3.6: Training comparison of pointwise, pairwise and listwise LTR approaches. In the pointwise, each supplier is consumed one by one, and a loss is obtained; in pairwise, pairs of suppliers belonging to one demand are consumed to obtain the loss, and in the listwise, all the suppliers belonging to a demand are used together to obtain the loss for training

Approach	Components	Description
Pointwise	*Input space*	feature vector of supplier
	Output space	relevance degree of each supplier
	Hypothesis space	scoring functions which take feature vector from input space and predict relevancy of each supplier
	Loss function	as the ranking task is modeled as a classification task, the corresponding loss function is classification loss
Pairwise	*Input space*	feature vectors of pairs of suppliers
	Output space	pairwise preference between each pair of supplier
	Hypothesis space	bi-variate functions that take pairs of suppliers as input and, outputs the relative ordering between them.
	Loss function	classification loss on pair of suppliers
Listwise	*Input space*	feature vectors of all the suppliers of a demand
	Output space	relevancy of all the suppliers of a demand
	Hypothesis space	multivariate functions that take group of suppliers as input and predict their relevancies
	Loss function	defined on the basis of approximation or bound of widely used IR evaluation measures

Table 3.7: Theoretical comparison between pointwise, pairwise, and listwise LTR approaches w.r.t the input space, output space, hypothesis space, and the loss function

3.6 Evaluation metrics

In the case of ranking, unlike the classification task, even the positions of an item in the ranked list, along with the relevancy, matters. Hence, accuracy cannot be used as an evaluation metric. Thus, the evaluation metrics Precision@k and Mean Average Precision that are the evaluation metrics for information retrieval systems are formulated based on the paper [Liu11] as follows.

Precision@k (Prec@k) Prec@k is calculated as follows:

$$prec@k = \frac{\{number\ of\ relevant\ suppliers\ till\ position\ k\}}{k} \tag{3.7}$$

Mean Average Precision (mAP)

$AvgPrec(d)@n = \frac{\sum_{k=1}^{m} Prec@k(d).b_k}{\#\{total\ number\ of\ relevant\ suppliers\ for\ d\}}$

where m is the total number of suppliers associated with the demand d, b_k is 1 if the supplier at position k is relevant and 0 otherwise.

$mAP@n = \frac{\sum_D AvgPrec(d)@n}{D}$

when $n = m$, then $mAP@n = mAP$

These are the evaluation metrics used for evaluating all the experiments that were conducted in this thesis.

3.7 Explainable AI (xAI)

3.7.1 SHapley Additive exPlanations (SHAP)

The game theory forms the basis of SHAP values [LL17] and the game here is to reproduce the predictions given by the model, and the players are the features included in the model. SHAP quantifies each feature's contribution to the outcome of the model per instance. The idea for calculating SHAP values is that every possible combination of features will be used to determine the importance of a single feature.

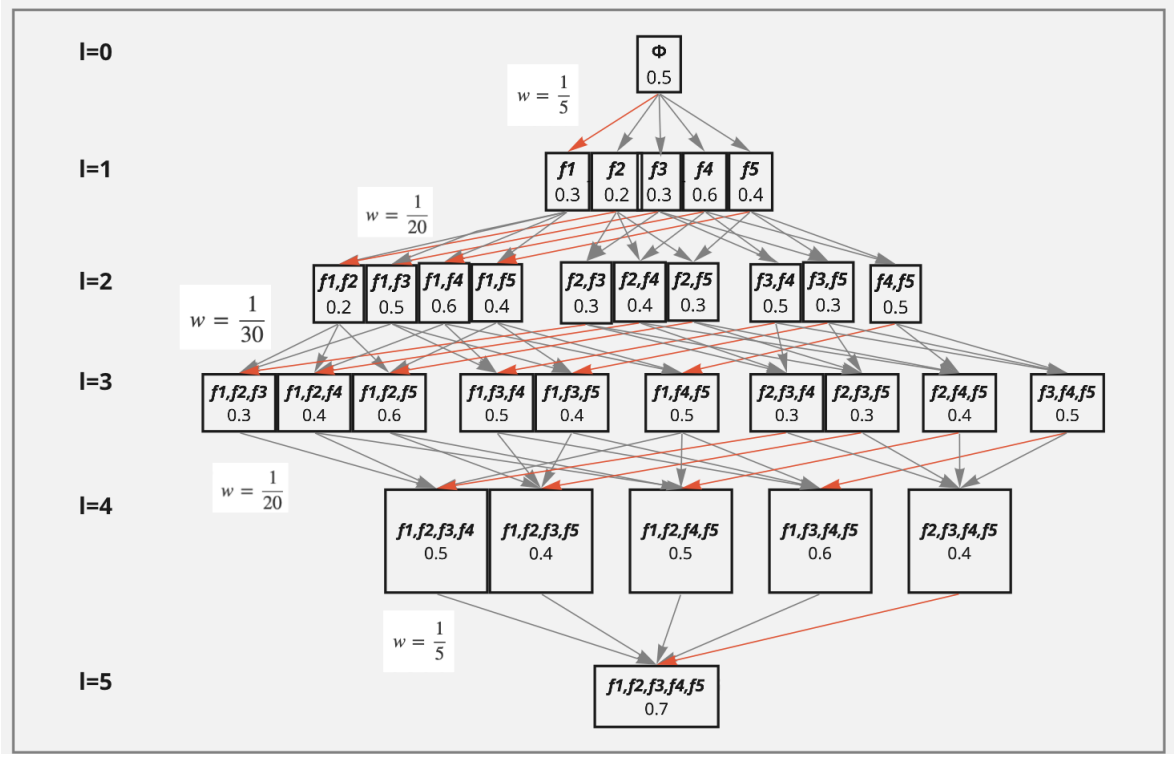


Figure 3.7: Power set for 5 features

Considering the 5 features in the toy example, the combination of all the features can be represented as a power set as shown in Fig. 3.7 where each node represents one combination(represented in square blocks) of features and each edge represents the inclusion of a feature that was absent in the previous combination (one level above). SHAP needs to train distinct models with the same hyperparameters for each of the combinations of features. For simplicity, each node is one distinct model. Each edge represents the marginal contribution of each feature to the outcome of the model. Consider an unknown instance given to the model for prediction and later for an explanation by SHAP. Imagining 32 different classifiers turned rankers have been trained on the same training data, each node

has a score for that instance given that particular combination of features. The SHAP value for feature f_1 is calculated as follows:

The edges which connect the nodes such that the upper node does not contain the feature f_1 and the lower node contains the feature f_1 are only considered, which are highlighted in the Fig. 3.7. The weight w given to each edge is the reciprocal of the total number of edges at the same level.

At level l=0, a model with no features will give 0.5 for the instance to belong to the group with relevancy=1.

At level l=1, the prediction of the model with just feature f_1 is 0.3. Implies that the marginal contribution of feature f_1 to the model containing just feature f_1 is 0.3-0.5=-0.2.

At level l=2, the marginal contributions of f_1 are 0 (0.2-0.2), 0.2 (0.5-0.3), 0 (0.6-0.6), 0(0.4-0.4).

At level l=3, the marginal contributions of f_1 are 0 (0.3-0.3), 0(0.4-0.4), 0.3 (0.6-0.3), 0(0.5-0.5), 0.1 (0.4-0.3), 0(0.5-0.5).

At level l=4, the marginal contributions of f_1 are 0.2 (0.5-0.3), 0.1 (0.4-0.3), 0.1 (0.5-0.4), 0.1 (0.6-0.5)

At level l=5 the marginal contribution of f_1 is 0.3 (0.7-0.4).

And the overall SHAP value for feature f_1 is a weighted average of all the marginal contributions of f_1 at each level where weight is w.

$SHAP(f_1) = (-0.2 * \frac{1}{5}) + (0.2 * \frac{1}{20}) + (0.3 * \frac{1}{30}) + (0.1 * \frac{1}{30}) + (0.2 * \frac{1}{20}) + (0.1 * \frac{1}{20}) + (0.1 * \frac{1}{20}) + (0.1 * \frac{1}{20}) + (0.3 * \frac{1}{5}) = 0.0683$

As the feature importance for each feature is calculated using the above idea, this gives feature importance relative to other features per instance, i.e., local, unlike the global feature importance given by some models like Random forest [KJ+13], [GMR+18] which is only global. SHAP is a better option for this work as it gives both local and global feature importance, unlike the other SOTA model-agnostic interpretability tool, LIME which may require some workaround to obtain global interpretability [RSG16].

4 Used Technologies

4.1 Algorithms

All the LTR approaches can be solved using either conventional machine learning or deep learning algorithms. All the algorithms that were implemented in this thesis are explained as follows.

4.1.1 Traditional Machine learning for LTR

In this thesis, machine learning algorithms that do not use Neural Networks are considered traditional machine learning algorithms. This includes pointwise LTR using Random Forest Classifier, pointwise LTR using XGBoost, and pairwise LTR using Random Forest Classifier. These are explained in detail as follows.

Pointwise LTR using Random Forest Classifier

Random Forest [Bre01] Classifier is a machine learning algorithm that takes the data input and predicts the class to which the input might belong. In the case of the problem statement in this thesis, based on the input supplier's features, the Random Forest Classifier will assign the supplier as relevant or irrelevant.

Following the explanation above, the question that is raised is how can a classifier be converted into a ranker? The classifier is transformed into a ranker by using the class probability of the supplier being relevant [FM08]. Consider a classifier is trained, and when this classifier is used for predictions on unseen data, instead of assigning binary labels, the probability that the classifier set for an instance to be positive is considered as the ranking score for that instance. Based on the descending order of their ranking score, all the instances in the unseen data are sorted. Then ranking evaluation measures are applied to obtain the performance of that particular model.

Before understanding Random Forest, it is important to understand the building blocks of the Random Forest, decision trees, also a supervised machine learning algorithm. The three components of a decision tree are a root node, decision nodes, and leaf nodes. A root node is the starting point, where the splitting starts. A root node is split to form decision nodes, and these nodes further divide to form decision nodes or leaf nodes which will not split any further. A split is decided based on the purity of the split. This purity is calculated using the metrics *Gini Index* and *Entropy*

These metrics are one of the hyperparameters for decision trees. In this way, a decision tree is constructed using these impurity measures for the instances with given features.

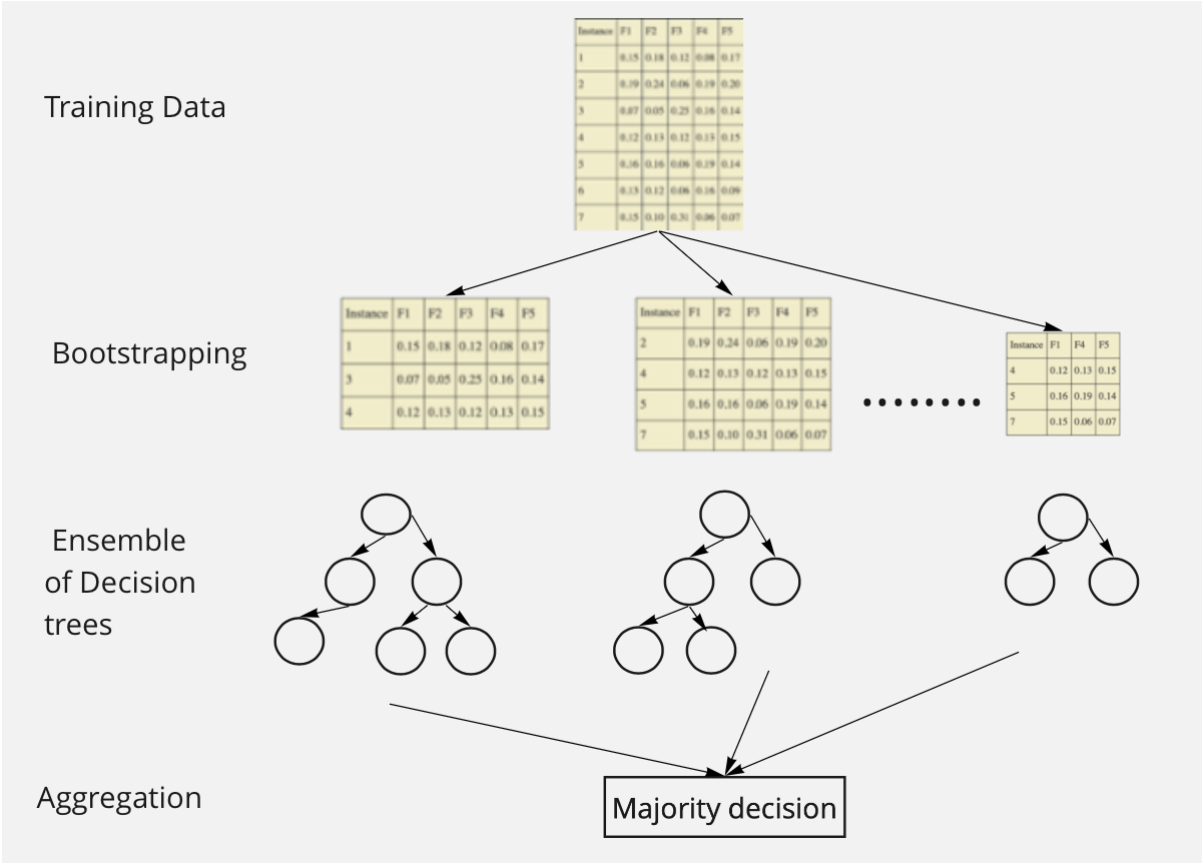


Figure 4.1: Random Forest. Bootstrapping is used to create subsets of the training dataset and the features with replacement, and decision trees are trained on these individual subsets. The predictions given by each of the decision trees are aggregated to form the final prediction.

Random Forests are a combination of multiple decision trees that extends the principle of bagging [Bre01], which is an ensemble technique. The difference between Random Forests and Bagging is that in Random Forests, along with a subset of training data, subsets of features are also sampled with replacement. In contrast, in bagging, only subsets of training data are sampled with replacement [Bre01]. Bootstrapping the features reduces the correlation between the various decision trees and thus, the variance. A random forest is represented in Fig. 4.1

The hyperparameters that are associated with Random Forest Classifier and were tuned while building the ranking model are stated below:

1. criterion These are nothing but the metrics *Gini Index* and *Entropy* that can be chosen to test the purity of the split.

2. max_depth As the name suggests, it specifies the maximum depth of the tree. If *None*, the tree grows until all the leaf nodes are pure or until all the leaf nodes contain no more than min_samples_split.

3. min_samples_split The minimum number of samples needed for the node to be able to split further.
4. n_estimators The total number of decision trees
5. max_features The maximum number of features that are taken into account when finding the best split.

Like bagging, the other ensemble technique in machine learning is boosting which is explained in the following section.

Pointwise and Pairwise LTR using XGBoost

Boosting can be understood by comparing it to bagging. The difference between bagging and boosting is that in bagging, the instances in the data are sampled with replacement. However, in boosting, the instances are given weight and are sampled on this weight. The intuition is to sample instances that were mispredicted more often by assigning them higher weight. Because of this weighting of the instances, the entire process of boosting becomes sequential, unlike bagging, which can be parallelized. Boosting is redefined by Gradient boosting to minimise the objective function of the model by adding weak learners (e.g., a single decision tree) using gradient descent. The pseudocode for Gradient boosting is as follows:

Training set: $\{(s_i, r_i)\}_{i=1}^{n}$, n is total number of instances, LossFunction=$Loss(r, F(s))$ and total number of iterations=M, $m \in M$ m=1

1: Initialise model with a constant value
$F_0(s) = \arg\min_\gamma \sum_{i=1}^{n} Loss(r_i, \gamma)$
2: **while** $m \neq M$ **do**
3: Compute pseudo-residuals
$p_{im} = -\left[\frac{\partial Loss(r_i, F(s_i))}{\partial F(s_i)}\right]_{F(s)=F_{m-1}(s)} for\ i = 1\ to\ n$
4: Fit a regression tree $h_m(s)$ to pseudo-residuals i.e. with new train set $=\{(s_i, p_{im})\}_{i=1}^{n}$
5: Compute γ_m using the following optimisation equation:
$\gamma_m = argmin_\gamma \sum_{i=1}^{n} Loss(r_i, F_{m-1}(s_i) + \gamma\ h_m(s_i))$
6: Update the model: $F_m(s) = F_{m-1}(s) + \gamma_m\ h_m(s)$
7: $N \leftarrow N + 1$
8: **end while**
9: Output $F_M(s)$

The training step from step3 to step7 in the above pseudocode trains the next learner on the Gradient of error based on the predictions loss of the previous learner. In layman's terms, the mistakes of the prior model are corrected during training.

XGBoost is an abbreviation for 'eXtreme Gradient Boosting' and is a scalable end-to-end tree boosting system [CHB+15]. XGBoost implements the Gradient boosting algorithm efficiently by using second-order derivatives and advanced regularisation. The overall workflow of XGBoost for LTR is shown in Fig. 4.2

The gradient boosting algorithm requires an objective or loss function that needs to be minimized.

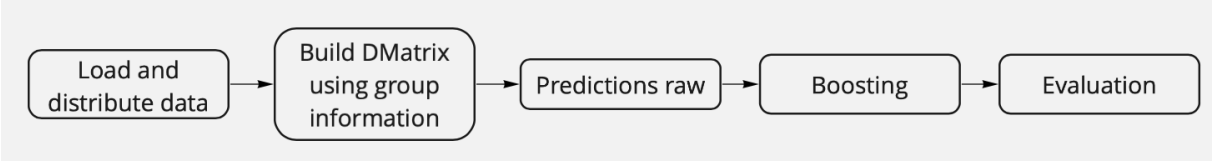


Figure 4.2: XGBoost workflow for LTR training

For training pointwise LTR using XGBoost, the objective function 'binary:logistic' is used. During prediction, this classifier is converted into a ranker by considering the probability that the classifier assigns for an instance to be positive as the score for that instance. For training pairwise, XGBoost provides a wrapper XGBRanker which implements LambdaMART [Bur10] when the objective function 'rank:pairwise' is given. The working of LambdaMART has already been explained in the pairwise part of the subsection 3.5.

The hyperparameters that are associated with XGBoost and were tuned while building the ranking model are stated below:

1. learning_rate This is used for shrinking the feature weights after each step to avoid overfitting, analogous to the learning_rate in Gradient Boosting Trees.
2. gamma The minimum reduction in the loss that is required before doing a split.
3. min_samples_split The minimum number of samples needed for the node to be able to split further.
4. max_depth Analogous to Gradient Boosting Trees, the maximum depth of the tree
5. subsample It gives the proportion of observations that needs to be randomly sampled at every iteration.
6. colsample_bytree It denotes the subsampling of the features/columns while constructing each tree.
7. scale_pos_weight It is used to tackle class imbalance.
8. objective It denotes the loss function that needs to be minimized.

4.1.2 Neural Networks for LTR using tensorflow-ranking

TF-Ranking Architecture

TensorFlow is a popular open-source library for training, evaluation, and serving of machine learning and deep learning models, and TensorFlow Ranking is built on top of it [PBW+19]. TensorFlow framework supports distributed training of neural networks via TensorFlow Estimator [CHH+17]. The estimator encapsulates two components: *input_fn* and *model_fn*. Hence the training block of 4.3 is structured using these two components.

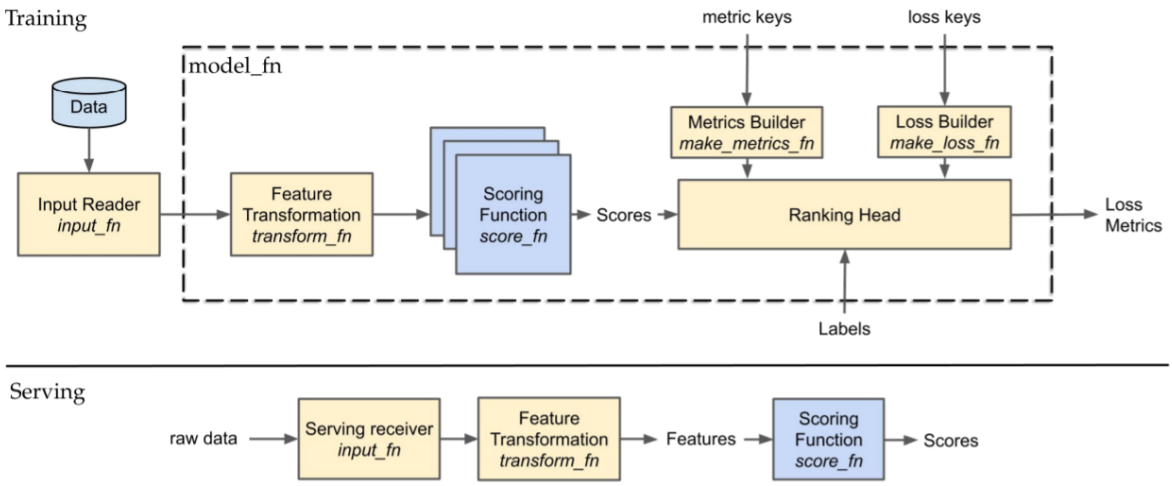


Figure 4.3: TF-Ranking Architecture [PBW+19]

- **Training**

  Input Reader *input_fn*

  Raw data is passed as an input to the *input_fn*, which returns the required type of tensors along with the labels. *input_fn* block is present during training and serving. The major difference is that in serving *input_fn*, the inputs are set up in a single batch, unlike the training *input_fn* where the batch size can be specified.

  Feature Transformation *transform_fn*

  The tf-ranking architecture requires List_Size or Group_Size, which denotes the number of candidate suppliers associated with a demand. The number of candidate suppliers associated with a demand varies. However, tf-ranking architecture requires a fixed number. Tf-ranking architecture handles this by either trimming the number of candidate suppliers associated with a demand if the number exceeds the specified List_Size or padding if the number of candidate suppliers associated with a demand are less than the List_Size. The *transform_fn* takes the output from the *input_fn*, List_Size or Group_Size and forms dense tensors.

  Scoring Function *score_fn*

  The *score_fn* outputs the scores internally during training and inference. The neural network used for the ranking is defined in the *score_fn*.

  Ranking Loss *make_loss_fn*

  The ground truth (labels) and the scores computed using the *score_fn* are inputs by the ranking loss function to return a weighted loss value.

  Ranking Metrics *make_metrics_fn*

  Tf-ranking architecture provides a metric function that takes the predictions and the ground truth label as input and provides a scalar value that denotes the overall performance of the model. More than one metric in the metric function can be defined. Precision, Recall, mean average precision, and so on can be defined using the metric function.

Ranking Head

The *make_metrics_fn* and *make_loss_fn* which are Estimator compatible are encapsulated in the ranking head.

model_fn

From Fig. 4.3, it can be seen that the *transform_fn*, *score_fn*, ranking head which encapsulates *make_metrics_fn* and *make_loss_fn* are combined in the model builder, model_fn.

- **Serving**

  During training, the ranking model receives a set of candidate suppliers associated with a demand. However, while serving, it may receive a group of independent candidate suppliers to be ranked. Tf-ranking architecture handles this discrepancy by exporting the graph generated by the model_fn as a SavedModel [OFG+17].

Learning in Neural Networks

As stated earlier, in a feedforward Neural Network, the input's initial information propagates through each layer, and finally, an output is obtained. This is known as forward propagation [GBC16] and the stopping point for this process is when a scalar loss is produced. This loss calculates how different the prediction is from the ground truth. The backpropagation algorithm then propagates this loss backward for calculating gradients and using the gradient descent algorithm, the weights and biases are updated, and in this way, the learning happens in the Neural Networks. The overall base architecture for all the LTR approaches using tensorflow-ranking is the same as explained in the architecture 4.3 and the critical difference lies in the loss function for each approach. The loss functions used in this thesis are mentioned in the respective subsections.

Pointwise LTR using tensorflow-ranking

The loss function for the pointwise approach is MeanSquaredLoss(MSELoss) which is given in the equation below:

$MSELoss(y, \widehat{y}) = \frac{1}{n}\sum_i^n (y_i - \widehat{y}_i)^2$

where n is number of dimensions and y_i and $\widehat{y}_i$ are the ground truth and predicted output respectively.

Pairwise LTR using tensorflow-ranking

$PairwiseHingeLoss(y, \widehat{y}) = \sum_i \sum_j (I[y_i > y_j] max(0, 1 - (\widehat{y}_i - \widehat{y}_j)$

$PairwiseLogistciLoss(y, \widehat{y}) = \sum_i \sum_j I[y_i > y_j] \log(1 + \exp(-(\widehat{y}_i - \widehat{y}_j)))$

$PairwiseSoftZeroOneLoss(y, \widehat{y}) = \sum_i \sum_j I[y_i > y_j](1 - sigmoid(\widehat{y}_i - \widehat{y}_j))$

In all the equations for Pairwise Loss functions, $I[y_i > y_j]$ is 1 if $y_i > y_j$, 0 otherwise.

Listwise LTR using tensorflow-ranking

$ListMLELoss(y, \widehat{y}) = -\log(P(\pi_y \mid \widehat{y}))$

where $P(\pi_y \mid \widehat{y})$ is the Plackett-Luce probability of a permutation and is conditioned on $\widehat{y}$. Here π_y represents a permutation of items ordered by the relevance labels, the ties of which are broken randomly.

$ListwiseSoftmaxLoss(y, \widehat{y}) = \sum_i y_i \cdot \log\left(\frac{exp(\widehat{y_i})}{\sum_j exp(\widehat{y_j})}\right)$

The hyperparameters that are associated with LTR using tensorflow-ranking and were tuned are stated below:

1. learning_rate It specifies how much to change the model in response to the estimated error while updating the weights of the model
2. number of hidden layers
3. number of neurons in each hidden layer
4. non-linear activation function
5. loss function

4.2 Feature Selection Methods

Features are essential in any machine learning model's performance, and LTR methods are feature-based. Hence, it seems necessary to find a good set of features.

Filter Method

Greedy Feature Selection-GAS Algorithm

Let's consider the toy example from the Table 3.2 and calculate the importance score for each feature by setting a threshold of 0.14 (mean of all values)

f_1:

$$\forall f \in f_1 \, if \; f > threshold \; then \, relevancy = 1 \, else \, 0$$

Predicted relevancy=[(1,1,0),(0,1,0,1)] accuracy=0.71

$$\forall f \in f_1 \, if \; f < threshold \; then \, relevancy = 1 \, else \, 0$$

Predicted relevancy=[(0,0,1),(1,0,1,0)] accuracy=0.28

Importance of f_1=0.71

Similarly we calculate the importance of f_2=0.85, f_3=0.85, f_4=0.71, f_5=0.57 and the respective predicted relevancies are as follows:

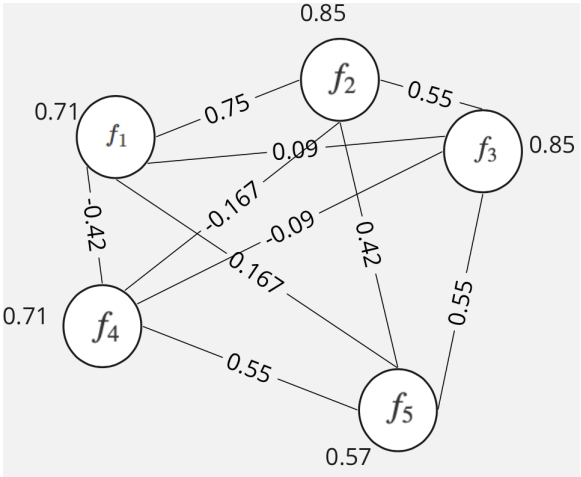
f_2=[(1,1,0),(0,1,0,0)]

f_3=[(1,1,0),(1,1,1,0)]

f_4=[(0,0,1),(0,1,1,0)]

f_5=[(1,1,0),(1,0,0,0)]

The weight between any two nodes is calculated by comparing the ranking result using Kendall tau.This results in the following graph:



Let S be the set of selected features and let the number of features to be selected be t. The weights of each node are updated using the following equation:

$$w_j \leftarrow w_j - e_{k_i,j} * 2c \; where \; j \neq k_i \; and \; k_i \; is \; a \; node \; from \; selected \; nodes$$

The working of the GAS algorithm with t=3 and c=0.1 is explained below.

Step 0:

$$S_0 = \{\}$$

Step 1:

Select the node with the largest weight. f_2 and f_3 both have the same weights. Let's select f_2.

$$S_1 = \{f_2\}$$

Let's update the weights of other nodes based on their similarity to f_2.

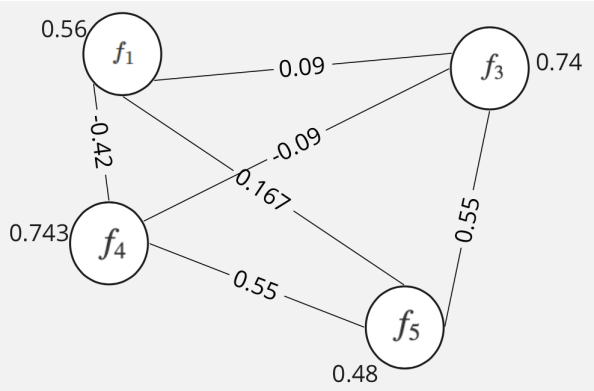
Weight of f_1= 0.71-0.75*0.2=0.56

Weight of f_3=0.85-0.55*0.2=0.74

Weight of f_4=0.71-(-0.167)*0.2=0.7434

Weight of f_5=0.57-0.42*0.2=0.486

We remove f_2 from the graph and all the connections that contain the node f_2 and end up with the following graph:



Step 2:

Select the node with the largest weight i.e .f_4 .

$$S_2 = \{f_2, f_4\}$$

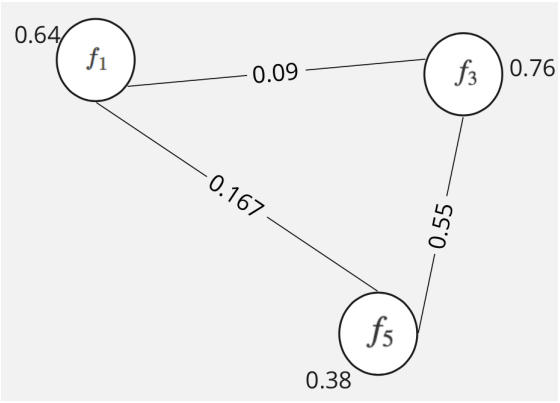
Let's update the weights of other nodes based on their similarity to f_4.

Weight of f_1= 0.56-(-0.42)*0.2=0.64

Weight of f_3=0.74-(-0.09)*0.2=0.758

Weight of f_5=0.486-0.55*0.2=0.376

We remove f_4 from the graph and all the connections that contain the node f_4 and end up with the resulting graph:



Step 3:

Select f_3.

$$S_2 = \{f_2, f_4, f_3\}$$

Stop here as the required number of features are obtained.

Clustering based method

We obtain the similarity weights between features using Kendall Tau, just like the graph based approach. After applying the clustering algorithm k-means [Llo82] with k=2, let's assume that the following two clusters are formed:

Cluster1= f_1,f_2,f_3

Cluster2= f_4,f_5

Now we select features from both the clusters with the highest weight. This weight can be obtained by calculating the mAP or accuracy after using only that feature to perform the ranking. Another way is to use a linear model to use the features with the highest weight. By using the former approach, we choose f_2 and f_3 from the first cluster and f_4 from the second cluster based on the weights assigned in the Graph based approach.

Wrapper Method

'The main difference between this approach and the filter approach is that the selection of features in the wrapper is based on the effectiveness of evaluation measures that will be optimized by the learning procedure' [SK18]. In simple terms, the features that impact the learning approach are selected. For example, in XGBoost Classifier and Random Forest Classifier, a feature importance list can be generated. This generated feature importance list can select the top-n features based on the requirement.

4.3 Sampling Strategies

In our case, the relevant suppliers form the minority class and the irrelevant suppliers form the majority class.

Undersampling for traditional machine learning models- In this technique, for each demand, the minority class suppliers were sampled without replacement with a size equal to that of the majority class suppliers associated with that demand. The demands which had the count of irrelevant suppliers less than the count of relevant suppliers were taken as it is.

Over sampling for traditional machine learning ranking models- In this technique, for each demand, the minority class suppliers were sampled with replacement with a count equal to the majority class suppliers associated with that demand

For deep learning based ranking models, tensorflow-ranking architecture requires an equal number of items associated with a query [PBW+19] denoted by the group_size. Hence slightly different sampling strategies are applied, which are explained below.

Undersampling in deep learning based ranking models- In this technique, group_size of 200 is taken because almost all the demands have the number of relevant suppliers less than 150, which is shown in Fig. 5.6 All the relevant suppliers are taken for each demand, and then 150 irrelevant suppliers are sampled without replacement. The relevant suppliers and the sampled irrelevant suppliers are further sampled without replacement with a size of 200. For the demands, which have more relevant suppliers than irrelevant suppliers, sampling with replacement with a size of 200 is done.

Padding- In this technique, the group_size of 800 is taken as the average number of suppliers associated with the demands is 600 5.2. For each demand, all the relevant suppliers are considered, and the irrelevant suppliers are sampled without replacement with the sample size of (800-the number of relevant suppliers). Suppose the number is not equal to 800. In that case, dummy suppliers with a feature vector of all zeroes and relevancy label as -1 are added to the list of suppliers associated with that demand. tensorflow-ranking framework ignores the instances with label -1.

4.4 Benchmarking

A caveat is that the test data consists of only already marked/labeled suppliers that number up to 600 for each demand in the test set. However, around 5000 suppliers for one demand are directed to the ranking model in production. Hence, the results obtained after evaluating the ranking models on the test set should be considered with a grain of salt. To tackle this issue, before selecting the champion model amongst the competing models, a user study is conducted with the help of domain experts. Four domain experts were requested to provide at least one peculiar demand they worked on after 04-2021, as the data used to build and evaluate the competing ranking models dated till 04-2021. Three domain experts provided one demand each, and one domain expert provided two demands, so a total of 5 demands created in the months of 10-2021 and 11-2021 were used in this human-based evaluation. Each competing model then provides the ranked list of suppliers for each of the five demands. The top 100 suppliers are then given back to their corresponding domain experts to check the Precision@100 for each model.

5 Demand Processing Pipeline; Design, Implementation and Evaluation

5.1 Data

Data is the core part of machine learning tasks. Implying it is a core component of the LTR tasks as well, that this thesis is trying to solve. In the paper, [WS], a document filtering step is applied instead of ranking and retrieving all the documents directly for a query. In this filtering step, documents are filtered w.r.t query. Ranking and retrieval are then performed on this query and the filtered document sets. The total number of suppliers that are dealt with by Scoutbee is approximately 1 billion. Ranking and retrieving these 1 billion suppliers for each demand is not at all a feasible solution. Hence, a filtering step is applied to form a demand-suppliers set on which the ranking algorithms are used.

5.1.1 Raw Data

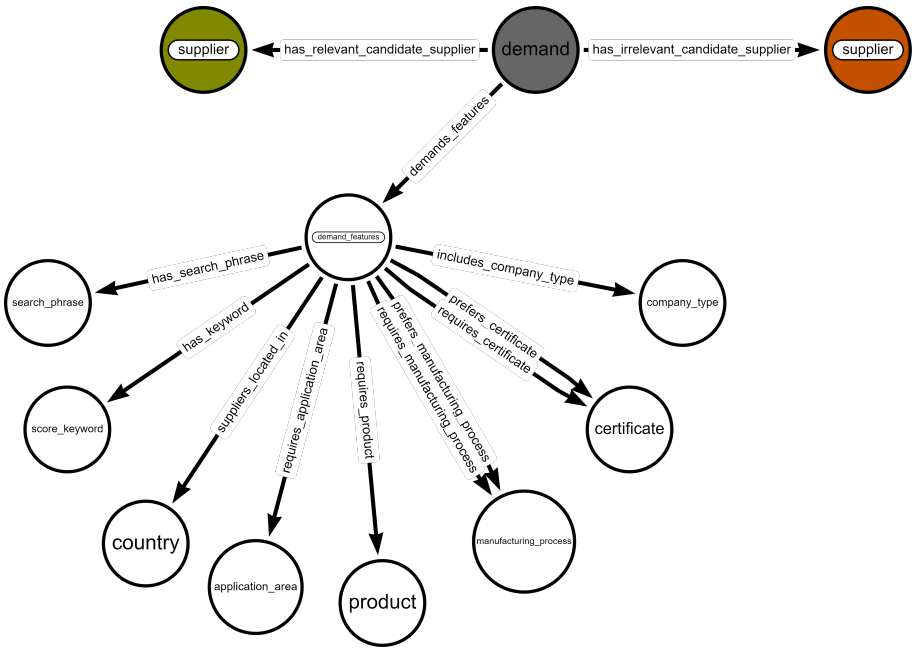


Figure 5.1: Graphical representation of demand data. A demand is associated with properties and a list of potential candidate suppliers. The demand properties are the requirements given by the customer. Suppliers must be ranked based on how much they match the demand's properties.

The demand property `product` consists of the keywords for the products the customer is looking for. `company_type` represents whether the customer is looking for a Manufacturer, Distributor, Online shop, Service provider, or Research facility, or Publisher. The customer may specify the location where the suppliers they are looking for must operate, given by the property `country`. `score_keywords` are the keywords filled by the domain experts, enhancing the `product` property of the demand. They are random adjectives or keywords describing similar products or even keywords that repeatedly appear on product pages of the suppliers that are being looked at. The customers may mention dealing with only suppliers with specific certificates in which the `certificate` property is linked to the demand with requires_certificate. The customers may mention *good-to-have* certificates where the `certificate` property is linked to the demand with prefers_certificate . `application_areas` indicate industrial application the customer is looking for. `manufacturing_processes` include the technical processes used to build the product. Similar to certificates, sometimes `manufacturing_processes` are required, and sometimes they are optional/preferred. `search_phrase` is formed by combining two keywords with AND, OR. `search_phrase` is a superset which includes the keywords from `company_type`, `score_keywords`, `certificate`, `application_areas`, and `manufacturing_process`. The `supplier` node consists of the domains of website content by crawling the domain of the supplier till a certain depth. The supplier can be relevant to the demand which is denoted by the link has_relevant_candidate_supplier or it can be marked as irrelevant which is denoted by the link has_irrelevant_candidate_supplier to the demand. This is used as the binary ground_truth for the models.

`Considering an example demand of "Vacuum Cleaners".`

- `product - Vacuum cleaner, dusting equipment`
- `company_type - Manufacturer`
- `country - Germany`
- `score_keywords - cleaning, dusting, filter suction power`
- `application_areas - Household appliances`
- `manufacturing_processes(required) - Injection moulding`
- `manufacturing_processes(optional) - motor assembly`
- `certificate(required) - ISO 9001`
- `certificate(optional) - ENT 4521`
- `search_phrases - "Vacuum cleaner" AND "manufacturer", "Vacuum cleaner" AND "ISO 9001"`
- `supplier - ['abc.com','xyz.de']`

The aforementioned demand features are used to generate features that will be consumed by the experimented ranking models. The raw data consists of 352 demands till 04-2021. Each demand has a varied number of candidate suppliers. On average, 600 suppliers (only marked ones) are associated with each demand which is shown in Fig. 5.2. The demand-supplier dataset is split into training set, validation set and test set based on the date of creation of the demand. The split is represented in Fig. 5.3. In the Fig. 5.4, the

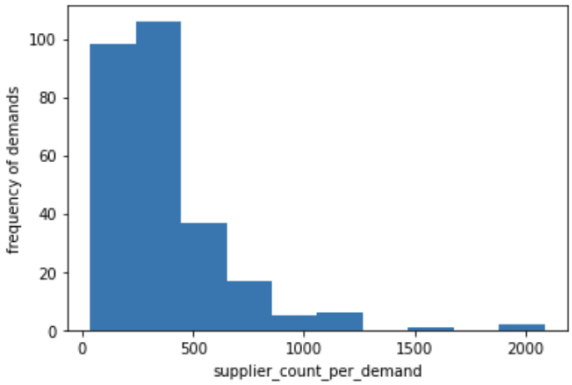


Figure 5.2: Demand-supplier count distribution. 50% of the demands have less than 600 suppliers associated with them. 8% of the demands have more than 1200 suppliers associated with them.

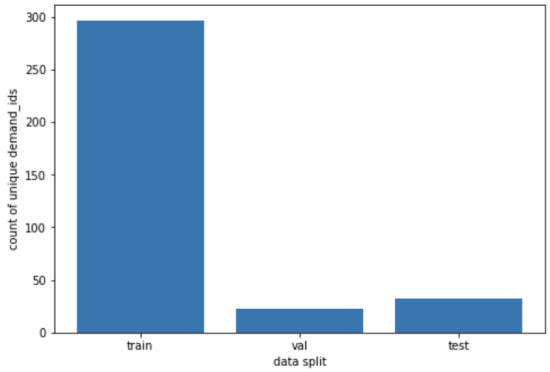


Figure 5.3: Dataset split into train,validation, and test. The train set consists of 290 demands, the validation set consists of 23 demands and the test set consists of 32 demands.

frequency of demands across months is shown. Fig. 5.5, Fig. 5.6, and Fig. 5.7 showcase the class imbalance of the demand-supplier dataset.

5.1.2 Feature Generation using OpenSearch

The supplier website content to a depth of 100 pages is indexed in to OpenSearch. The domain of the supplier, URL of the webpage, title of the webpage, and the webpage's description are considered for indexing. For example a supplier with domain `abc.com` may contain webpages which have URL like `https://abc.com/product` with a description of the `product` served by `abc.com`. It is important to note that websites do not follow a fixed structure; hence, when generating the features, all the webpages indexed for a domain are queried instead of doing a webpage URL-specific query. Elaborating further, for querying the products served by `abc.com`, all the indexed webpages for the domain `abc.com` are queried and not just `https://abc.com/product`.

The two ways in which the features are generated are by a partial match of the keywords and by doing an exact match. In a partial match setting, each space-separated

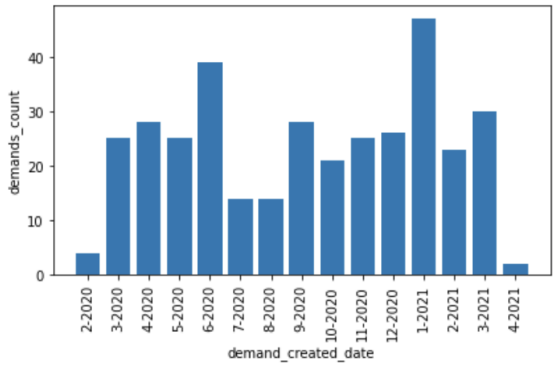


Figure 5.4: Date-wise demand distribution. The demands that were created till 02-2021 are included in the training set, from 02-2021 to 03-2021 are included in the validation set and from 03-2021-04-2021 are included in the test set.

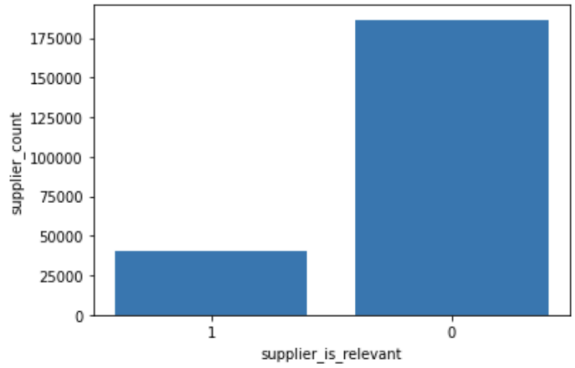


Figure 5.5: Ground truth distribution. Only 18% of the suppliers belong to class 1, i.e, most of the suppliers are irrelevant in the dataset.

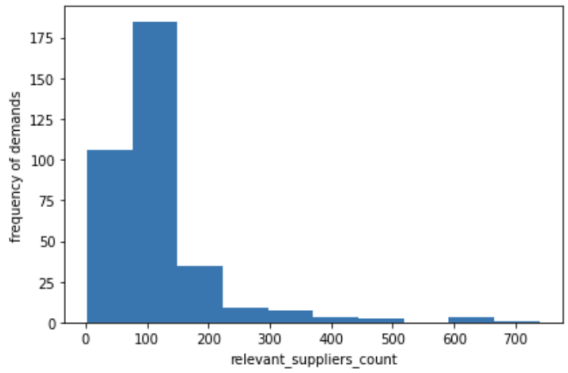


Figure 5.6: Relevant suppliers across demands. 80% of the demands have less than 150 relevant suppliers associated with them.

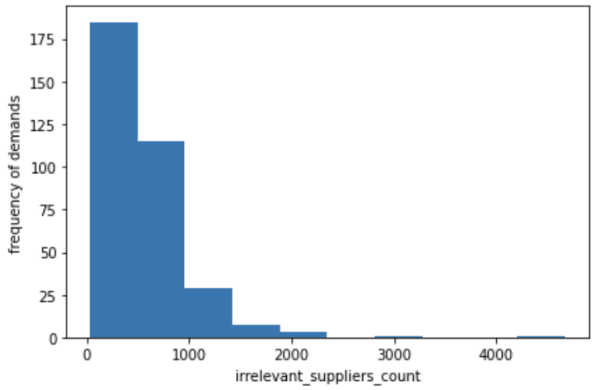


Figure 5.7: Irrelevant suppliers across demands. 86% of the demands have upto 1000 irrelevant suppliers associated with them.

token is considered one keyword. In an exact match setting, the comma-separated token is regarded as one keyword. These space-separated tokens and comma-separated tokens are passed to open search, and for each demand characteristic, an overall total_hits and max_score are obtained. When space-separated tokens are used, they will be addressed as being generated by the **token-based** approach of feature generation and the case of comma-separated tokens as the **phrase-based** approach of feature generation.

total_hits = Total number of hits across all the keywords in one demand feature for a given domain. This gives the total number of pages in which either of the keywords is present.

max_score = Maximum relevance score across all the keywords in one demand feature for a given domain.

OpenSearch is built on top of Lucene [MHGG10]. Scoring function given by OpenSearch is the modified form of the BM25 equation 3.1 which is given by the equation 5.1.

$$\begin{aligned} score(d,p) = & \; queryNorm(d) \\ & * coord(d,p) \\ & * \sum_{i}^{N}(Tf(t_i,p) * IDF(t_i)^2 * t_i.getBoost()) \\ & * norm(t_i,p)) * (t_i,d) \end{aligned} \tag{5.1}$$

where $score(d,p)$ is the relevance score of supplier webpage p for the keywords of the demand d, $queryNorm(d)$ is the square root of sum of squares of IDF of each keyword in the demand keywords, $coord(d,p)$ is the count of number of keywords from demand keywords d that appear in the supplier webpage p, $t_i.getBoost()$ and $norm(t_i,p)$ are used in terms of muti-field. However, in our case, only the description field is queried. The $Tf - IDF$ can be referred from the equation 3.1 The highest score given by any of the supplier's web pages is then considered to be the max_score of that supplier.

Revisiting the example demand of `"Vacuum Cleaners"`, the feature generation for the

`product - Vacuum cleaner, dusting equipment` field using token-based and phrase-based feature generation for the domain `abc.com` is shown below.

The **token-based** feature generation approach is implemented by querying the OpenSearch with a query that semantically looks like `Vacuum OR cleaner OR dusting OR equipment` and programmatically looks as follows.

```
{'query':
    {'bool':
        {'should': [
                    {'bool': {'must': [
                    {'match': {'domain': {'query': 'abc.com'}}},
                    {'match': {'description': {'query': 'Vacuum'}}}]}},
                    {'bool': {'must': [
                    {'match': {'domain': {'query': 'abc.com'}}},
                    {'match': {'description': {'query': 'cleaner'}}}]}},
                    {'bool': {'must': [
                    {'match': {'domain': {'query': 'abc.com'}}},
                    {'match': {'description': {'query': 'dusting'}}}]}},
                    {'bool': {'must': [
                    {'match': {'domain': {'query': 'abc.com'}}},
                    {'match': {'description': {'query': 'equipment'}}}]}}
                    ]
        }
    }
}
```

OpenSearch will give the score for all webpages indexed for the domain `abc.com` and the feature demand_product_keywords_max_score will denote the highest score obtained by either webpages for the domain `abc.com`. In addition, the number of web pages in which either of the keywords occurs will also be given, which will denote the demand_product_keywords_total_hits.

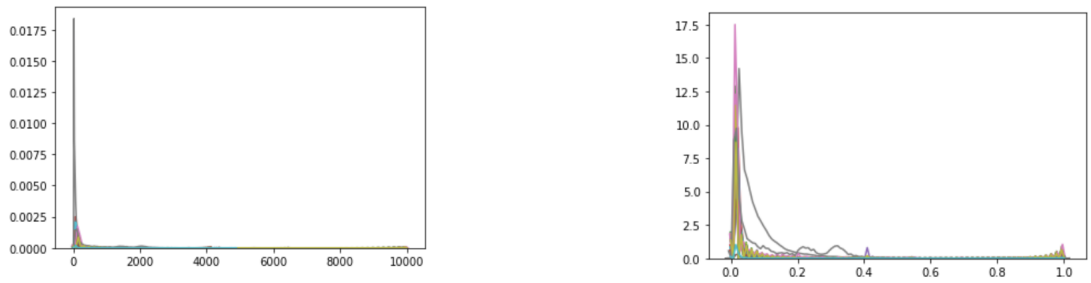
The **phrase-based** approach of feature generation is implemented by querying the OpenSearch with a query that semantically looks like `Vacuum cleaner OR dusting equipment` and programmatically looks as follows

```
{'query':
    {'bool':
        {'should': [
                    {'bool': {'must': [
                    {'match': {'domain': {'query': 'abc.com'}}},
                    {'match_phrase': {'description': {'query': 'Vacuum cleaner'}}}]}},
                    {'bool': {'must': [
                    {'match': {'domain': {'query': 'abc.com'}}},
                    {'match_phrase': {'description': {'query': 'dusting equipment↩
                        '}}}]}}
                    ]
        }
    }
}
```

The feature demand_product_keywords_max_score_phrase for the domain `abc.com` is denoted by the highest score obtained by either webpages. In addition, the feature demand_product_keywords_total_hits_phrase is the the number of web pages containing either of the keywords.

Both the feature generation methods are implemented for all the other demand features mentioned in the graphical representation of a demand represented in the Fig. 5.1. Each demand has ten characteristics; hence in total, 40 features are generated (10*(1+1 from

token based)*(1+1 from phrase based)). All the features are then normalized using min-max scaling. The distribution of features before and after normalization is shown in Fig. 5.8



(a) Before Normalisation

(b) After normalisation

Figure 5.8: Features distribution. From (a), it can be seen that the feature values are widespread from 0 to 10000, and after normalization, it can be seen from (b) that all the feature values have a range between 0 and 1

Analysis of OpenSearch generated data

The two-sample Kolmogorov-Smirnov test for each feature in the train, validation, and test set gave the outcome that rejected all the features denoting they don't follow the same underlying distribution. This can be due to the heavy sparsity of the data.

Apart from the class imbalance problem mentioned above, in Fig. 5.9, it can be seen that most of the handcrafted features are sparse. Sparsity for a feature f is calculated as follows:

$$sparsity(f) = \frac{count\ of\ values = 0\ in\ f}{count\ of\ values = 0\ in\ f + count\ of\ values \neq 0\ in\ f} \tag{5.2}$$

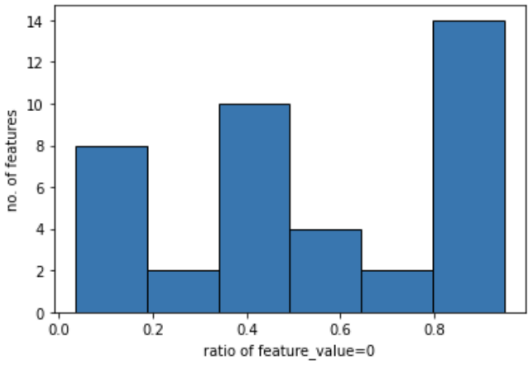


Figure 5.9: Data Sparsity representation. Out of 40 features, 14 features have more than 80% of feature_value as 0.

Table ?? compares the sparsity of features generated using the token-based approach and features generated using phrase based approach. This table validates the fact that, for example, it's easier to get a hit from a webpage for `'Vacuum' OR 'cleaner'` rather than for `'Vacuum cleaner'`.

demand feature	token based feature generation	phrase based feature generation
product	0.15	0.6
company_type	0.38	0.44
score_keyword	0.17	0.47
manufacturing_processes_high_impact	0.93	0.95
manufacturing_processes_low_impact	0.78	0.87
certificate_high_impact	0.91	0.95
certificate_low_impact	0.83	0.88
country	0.12	0.20
application_area	0.40	0.61
search_phrases	0.03	0.43

Table 5.1: Sparsity of features generated using token-based approach and phrase-based approach. Features generated using the phrase-based approach are comparatively more sparse.

5.2 Experimental Setup

All the experiments are performed using Databricks, which provides a plugin for mlflow, making it easier to track all the experiments, from hyperparameter tuning to registering the outperforming model for further deployment. Python is used to code all the implementation of feature engineering to model deployment. The evaluation of all the models is done using mean average precision (mAP) and Precision@k.

5.2.1 Experiments with the handcrafted features

The demand_supplier_raw data was transformed into numerical representation using 79 handcrafted features A.. Pre-trained BERT [DCLT16] was used to embed all the text data of demand features and the landing page of the respective supplier's website. Cosine similarity between the embedded demand features and the embedded supplier's landing page is used to generate the 79 handcrafted features.

`The base model XGBoostClassifier with 79 features gave an mAP of 0.45 on the test set.`

However, after a certain period, the performance given by the model became stagnant. This called for an investigation of the features and the modification of the model. The workflow implemented to resolve this request is shown in Fig. 5.10 and explained below. This is followed by the results and discussions of the experiments.

Pipeline

The data consisted of 352 unique demands with an average of 600 suppliers associated with each one of them. It consisted of 79 handcrafted features. As there were 79 features, a correlation check was done on the features, and one feature from a pair of highly correlated features was taken, which gave a final set of 60 uncorrelated features. The features used to

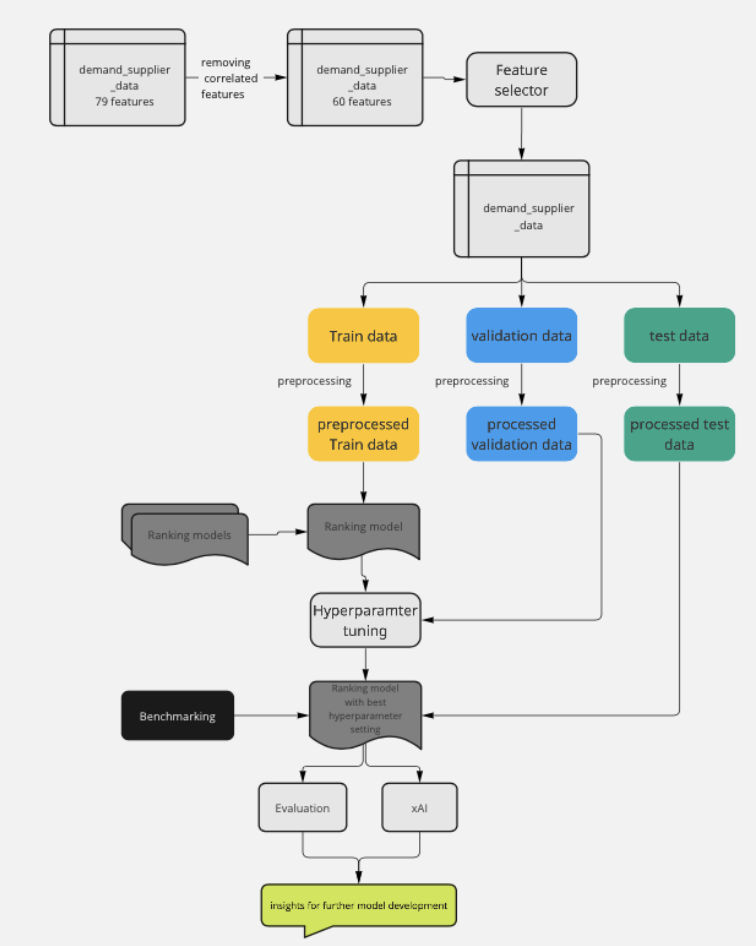


Figure 5.10: Pipeline for experiments with the handcrafted features. Correlated features from the data with 79 features were removed to form the dataset with 60 features. Twenty-five features were selected using feature selection methods. This data with 25 features was split into train, validation, and test set on which preprocessing was done, followed by training the ranking models using training and validation data and evaluating them on the test set and concluded by providing the highly impacting features for each model. This workflow combines feature selection with the three LTR approaches.

build various LTR models were selected using the feature selection approaches explained in Section 4.2. These feature selection approaches were applied to select 25 features, forming demand_supplier_data with selected features. The demand_supplier_data with selected features was then date-wise divided into train, validation, and test set. The mutually exclusive train, validation, and test data were preprocessed by applying min-max scaling and filling all the empty values with 0.

The training data and validation data were then used to train the ranking models one by one. The models used in this workflow were traditional ML based pointwise LTR approach implemented using XGBoost, Deep learning based pairwise, and listwise approaches using the library tensorflow-ranking. The experimented ranking models were evaluated on the test data using mAP. Benchmarking, explained in Section 4.4 was done and SHAP plots were used to gain insights from the features that impacted the best performing model.

Results

Table 5.2 gave the mAP scores obtained by the XGBoostClassifier with the three feature selection method. The highest mAP was given by the XGBoostClassier, which did not use feature selection and considered all the 60 features(79-highly correlated features). As the feature selection methods did not give the expected results, the next set of experiments was performed using deep learning based pairwise and listwise approaches with all the 60 features, shown in Table 5.3. However, neither of the deep learning based LTR approaches outperformed the champion model from Table 5.2.

Feature selection	**mAP**
All features (no feature selection)	**0.46**
Filter Method GAS algorithm	0.39
Clustering based	0.43
Wrapper method	0.42

Table 5.2: Evaluation of traditional ML based Pointwise LTR using XGBoostClassifier with different feature selection methods.

Approach	**Loss function**	**mAP**
Deep learning based pairwise LTR	pairwise_logistic_loss	0.40
	pairwise_hinge_loss	0.40
	pairwise_soft_zero_one_loss	0.41
Deep learning based listwise LTR	softmax_loss	0.40

Table 5.3: Evaluation of Deep learning based Pairwise and Listwise LTR approaches.

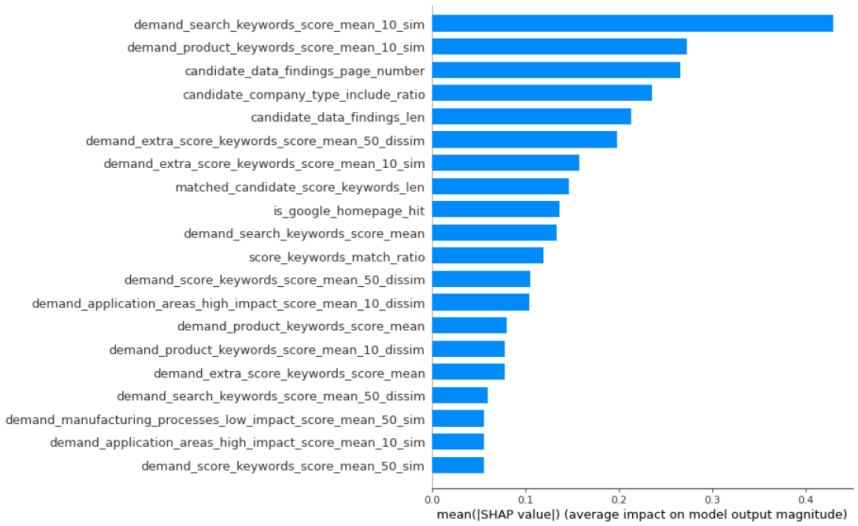


Figure 5.11: Feature importance plot of the top performing model, i.e., traditional ML based pointwise - XGBoostClassifier

Hence, it can be stated that the XGBoost model without any feature selection outperformed the three XGBoost models with their respective feature selection methods and Deep learning based pairwise and listwise models.

The champion model was further investigated using SHAP plots. Going from model-driven machine learning to product-oriented machine learning is the need of the hour. Hence, all the investigations were done by keeping the domain experts, who are also the end-users, in the loop. When the SHAP plots were discussed with the domain experts, the overall feedback was that, firstly, most of the features weren't self-explanatory, and secondly, most of them were redundant. Moreover, the entire pipeline was complicated, and in production, it took two hours per demand. There was a demand for a model with self-explanatory features from the domain experts. Hence, it was considered a sunk cost instead of circling back to improvising the model.

RQ2 hypothesized that the explainability of the ranking model could be leveraged to obtain better insights. The previous investigations played a major role in selecting the features for the new approach to building the ranking model. The domain experts suggested that they would like to have features like the features in the SHAP plots Fig. 5.11 with a suffix '_match_ratio'. These features gave the ratio of the keywords from a particular demand feature matched in the supplier's landing page. This led to the feature generation using OpenSearch. `This positively answers RQ2 as the insights gained from the explainability module of the ranking approach helped to build a simpler approach.`

5.2.2 Intermediate experiments with features generated using OpenSearch

The discussions from the experiments as mentioned earlier and results led to the following workflow shown in Fig. 5.12 to solve the ranking problem. This approach was based on Occam's razor principle. With this approach, a ranking pipeline not only has far fewer features than the previously mentioned model but also self-explanatory features.

Pipeline

The demand_supplier_raw_data raw data 5.1.1 consisted of 10 demand characteristics 5.1 along with their associated domains. The token-based approach, explained in the subsection 5.1.2 where each keyword is split on whitespace and sent as an OR query to AWS_OpenSearch was then used to get the total_hits and max_score for each of the demand characteristics of the raw data. The domains of the suppliers are indexed and stored in the OpenSearch with a page depth of 100.

The token-based approach converted the demand_supplier raw data into data that consisted of 20 features (for each demand characteristic, two features were obtained total_hits and max_score; therefore 2*10) which formed the final demand_supplier_data to conduct the experiments on. This data was then split into train, validation, and test set based on their creation date and further preprocessed by filling the empty values with 0. Traditional machine learning based pointwise ranking model using a random forest classifier was trained using the preprocessed train data, and the hyperparameter tuning was done and validated on the validation data. The trained random forest classifier

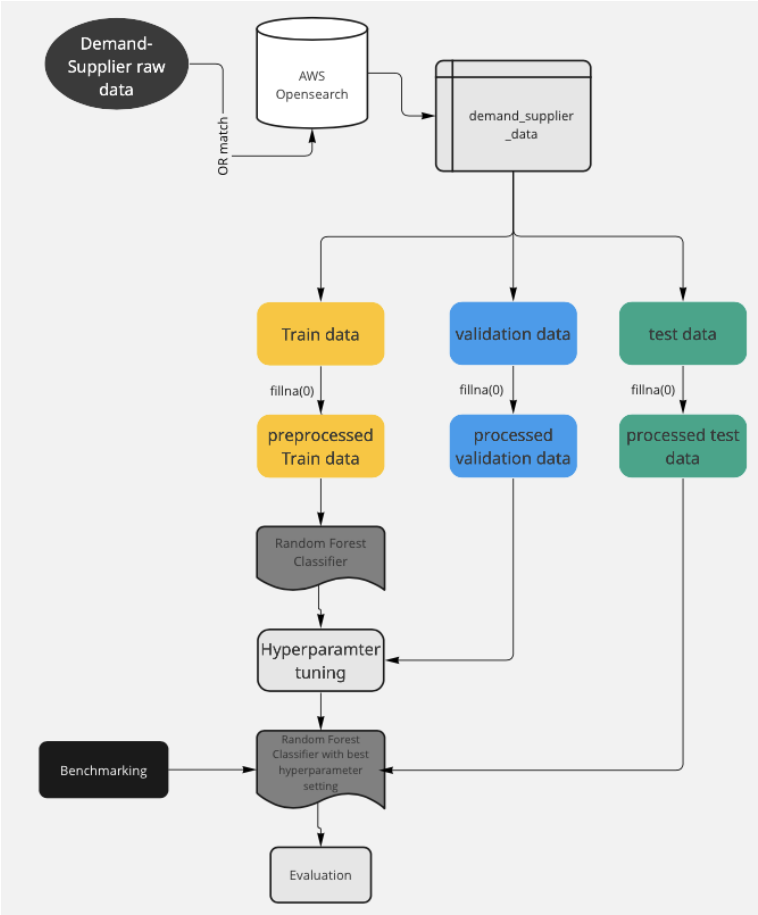


Figure 5.12: Pipeline for experiments with token-based feature generation using OpenSearch. The raw data is transformed into numerical representations. This transformed data is split into train, validation, and test set. The preprocessed train and validation data are used for training the random forest classifier. The trained classifier is evaluated using the preprocessed test data and benchmark demands.

was then evaluated using the preprocessed test set and human-evaluated using the five benchmarking demands mentioned in 4.4.

Results

`mAP of the random forest classifier on the test set was 0.43.` However, as mentioned previously, the test set only consists of approx. 600 suppliers per demand, unlike the real world setting where approx. 5000 suppliers are to be ranked. Hence benchmarking is done with the help of domain experts, which is shown in Table 5.4.

Even though on the test set, the previous workflow had a higher mAP score of 0.46 compared to the intermediate workflow with the mAP score of 0.43, in the human evaluation, the intermediate workflow outperformed in all except one demand. However, compared to the overall spread of Prec@100 for the previous workflow, the overall spread of Prec@100 for the intermediate workflow is uniform, as shown in Fig. 5.13.

demand_id	P@100(Previous Workflow)	P@100(Intermediate Workflow)
783	0.31	**0.38**
1928	0.17	**0.42**
1651	0.28	**0.29**
1976	**0.32**	**0.32**
1817	**0.38**	0.32
overall	0.30	**0.35**

Table 5.4: Performance of the winning XGBoostClassifier from the previous workflow and the random forest classifier from this intermediate workflow.

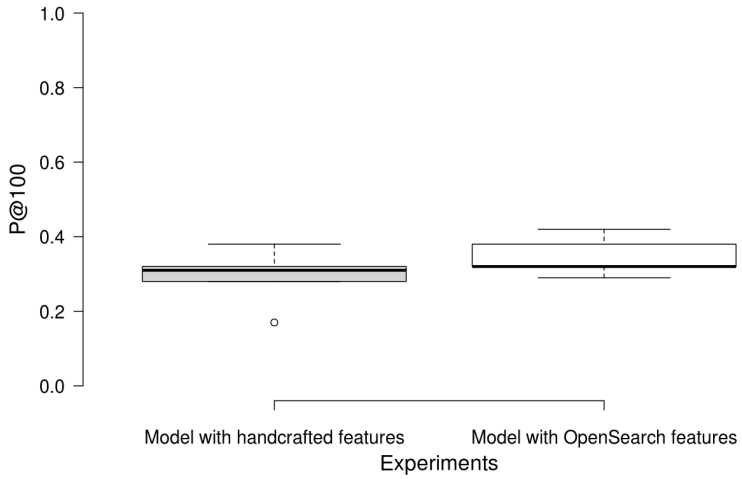


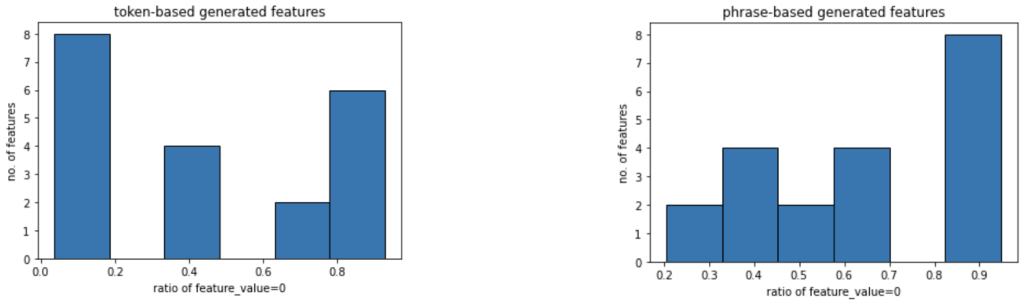
Figure 5.13: Uniformity in the performance of Precision@100 for the champion XGBoostClassifier from the experiments with handcrafted features and the random forest classifier from the intermediate experiments with OpenSearch generated features on the five benchmark demands.

One of the goals of this thesis is to build a product-driven solution. `RQ1 aimed at validating Occam's razor, and based on the results, it can be concluded that a simpler ranking model with just 20 features is better than the complicated winning ranking model from the previous experiments with 60 features.` Even though the performance was better and the features were self-explanatory, the domain experts, who are also the end-users, were skeptical of the token-based approach for feature generation. They proposed a requirement for an exact match, i.e., the phrase-based method for feature generation.

5.2.3 Final experiments with features generated using OpenSearch

Based on the previous discusions, features generated by considering the keywords belonging to each demand feature as a whole without any split was needed. This can be done using the phrase based approach of feature generation using OpenSearch. However, having features generated using only the phrase-based approach made the overall dataset sparse Fig. 5.14.

In Fig. 5.14a, it can be seen that out of the 20 generated features, 8 features have sparsity >



(a) Sparsity of token-based generated features **(b)** Sparsity of phrase-based generated features

Figure 5.14: Features distribution

0.5 whereas in Fig. 5.14b, out of the 20 generated features, 12 features have sparsity > 0.5. This led to the following pipeline shown in Fig. 5.15 that combined both the token-based feature generation and the phrase-based feature generation.

Pipeline

The demand_supplier_raw_data as explained previously, consisted of demand characteristics along with their associated domains. The token-based approach for feature generation was used to get the total_hits and max_score for each of the demand characteristics. Unlike the previous experiments with OpenSearch, the phrase-based approach was also used to generate the features total_hits_phrase and max_score_phrase for each of the demand characteristics. Similar to the previous experiments with OpenSearch generated features, the domains of the suppliers are indexed and stored in the opensearch with a page depth of 100. The token based approach converted the demand_supplier raw data into demand_supplier_token_data that consisted of 20 features. The phrase based approach converted the demand_supplier raw data into the demand_supplier_phrase_data with 20 features. The demand_supplier_token_data and the demand_supplier_phrase_data were combined to form the demand_supplier_data which consisted of 40 features.

This demand_supplier_data was split into train, validation and test set and further preprocessed by applying the preprocessing steps as shown in the Fig. 5.16 forming the pre-processed train data, pre-processed validation data and pre-processed test data. Traditional machine learning based pointwise ranking model using random forest classifier, traditional machine learning based pairwise ranking model using XGBoost, deep learning based point wise ranking model, deep learning based pairwise ranking model and deep learning based list wise ranking models were experimented in combination with various sampling techniques. For random forest, no sampling with class weight parameter as balanced, undersampled train data and oversampled train data were used. For XGBoost pairwise, no sampling, undersampled train data and oversampled train data were used. For Neural Network based pointwise, pairwise and list wise with undersampling and padding technique were used. In combination 12 different ranking approaches were trained using the various sampled train data and hyperparamter tuning was validated using the pre-processed validation set. One trained ranking model with the best hyperparamter setting from each of the 'sampling techniques - LTR approach' combination was further evaluated on the test set and also evaluated by the domain experts using the 5 benchmarking demands.

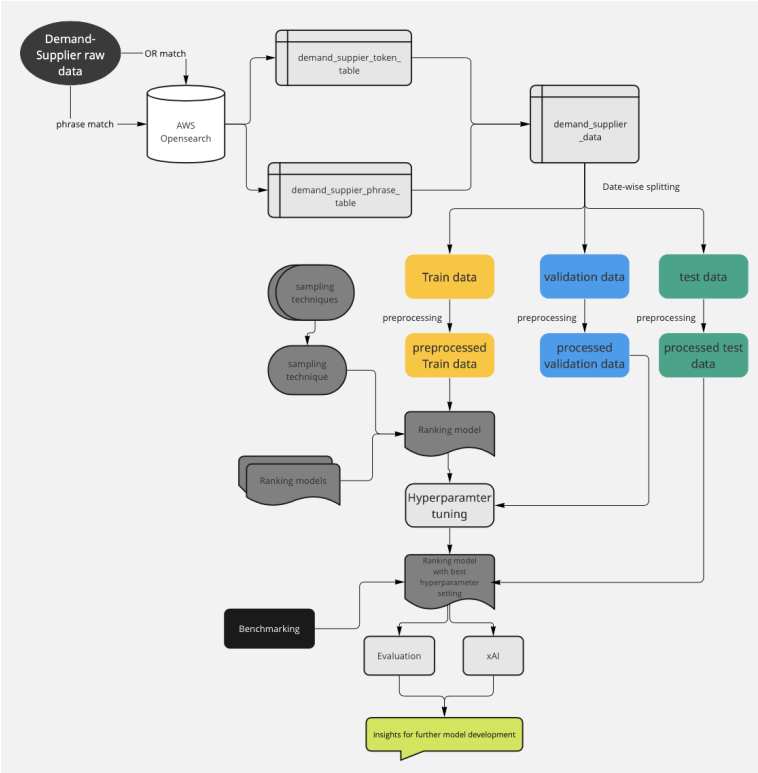


Figure 5.15: Pipeline for experiments with token-based and phrase-based feature generation using OpenSearch. The raw data is transformed numerical representations. This transformed data is split into train, validation and test set. Preprocessed and sampled train and pre-processed validation data are used to train the ranking models and evaluated using test data and benchmark data. Feature importance plots using SHAP are plotted for the best performing ranking models to gain insights.

Results

Table **??** provide the experimentation results of traditional machine learning based point wise and pairwise approach which were trained using various sampling techniques no sampling, under sampling and oversampling and the results of deep learning based LTR approaches which were trained using under sampling and padding. All the traditional machine learning based ranking models provide >0.5 precision@10 and comparatively perform better than that of deep learning based learning to rank models. However, according to the precision@100 the deep learning based models pairwise and listwise and the traditional machine learning based ranking models provide more or less the same performance.

Another interesting finding is that all the deep learning based ranking models have a lower mAP@10 compared to their respective overall mAP. With respect to the sampling techniques, for all the approaches, there isn't any significant difference between the performance of the models when any sampling technique is applied. The least performing model is the deep learning based pointwise approach. Based on previous explanations, considering the reliability of the results on the test and validation set with a warning, a human-based evaluation is conducted which is explained in detail as fol-

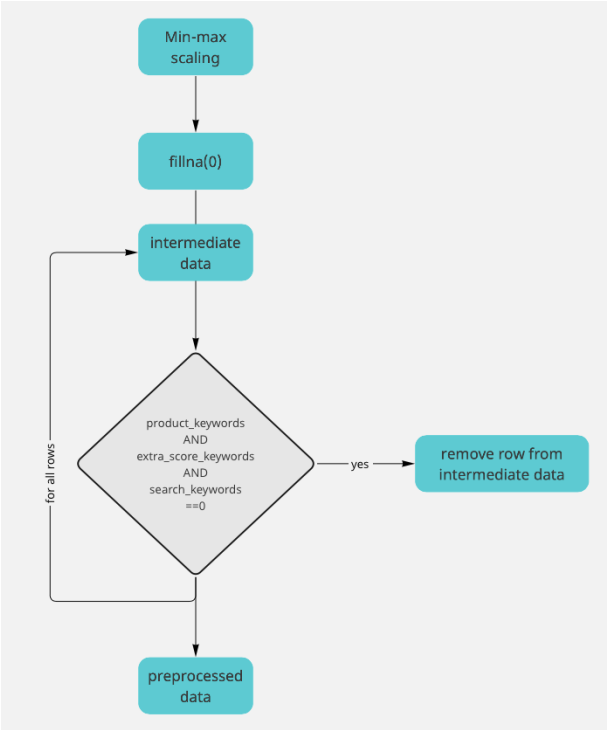


Figure 5.16: Pre-processing steps of train, validation, test, and benchmark data

lows.

In Table 5.5, the performance of various ranking approaches on domain expert evaluated demand_ids is tabulated. The most important observation is that the prec@100 for all the models is less than what was observed on the test set and the most peculiar one being the traditional machine learning based pairwise approach which has shown the most drastic decline in the performance. Reminding that the benchmark demands were from October-2022 and November-2022 and the training, validation and test data consisted of the demands only till April-2022, these results suggests that there might be a data drift which is causing this behaviour. Deep learning based listwise approach with padding of the training data gave the highest overall precision@100 of 0.414, followed by the traditional machine learning based pointwise approach with under-sampled data with a precision@100

Approach	**Model**	**mAP@10**	**mAP@100**	**mAP**	**P@10**	**P@10**
Traditional Machine Learning	pointwise no sampling	0.53	0.33	0.44	0.65	0.44
	pointwise undersampling	0.54	0.34	**0.45**	0.64	0.44
	pointwise oversampling	**0.55**	**0.35**	**0.45**	**0.66**	**0.46**
	pairwise no sampling	0.5	0.32	0.43	0.62	0.42
	pairwise undersampling	**0.56**	**0.35**	0.45	**0.68**	**0.46**
	pairwise oversampling	0.55	**0.35**	**0.46**	0.66	**0.47**
Deep learning	pointwise padding	0.35	0.24	0.38	0.47	**0.38**
	pointwise undersampling	0.31	0.28	0.42	0.49	**0.42**
	pairwise padding	0.38	0.35	0.45	0.56	**0.48**
	pairwise undersampling	0.38	0.35	0.45	0.57	**0.48**
	listwise padding	0.37	0.35	0.46	0.57	**0.48**
	listwise undersampling	**0.40**	0.35	0.45	0.58	0.47

Approach	models/demands	783	1928	1651	1976	1817	overall
Traditional Machine learning	Pointwise no sampling	0.38	0.26	**0.46**	0.47	**0.37**	**0.388**
	Pointwise undersampling	0.44	0.39	0.39	0.44	0.30	**0.392**
	Pointwise oversampling	0.42	0.22	**0.48**	0.48	0.33	**0.386**
	Pairwise no sampling	0.22	0.09	0.16	0.12	0.06	0.13
	Pairwise under sampling	0.42	0.09	0.28	0.25	0.24	0.256
	Pairwise over sampling	0.42	0.07	0.39	0.34	0.23	0.29
Deep Learning	Pointwise undersampling	0.36	0.13	0.26	0.35	0.22	0.264
	Pointwise padding	0.37	0.17	0.21	0.60	0.29	0.328
	Pairwise under sampling	0.40	0.43	0.23	0.56	0.27	0.378
	Pairwise padding	**0.46**	0.42	0.23	0.53	0.22	0.372
	Listwise undersampling	0.42	0.41	0.21	0.50	0.34	0.376
	Listwise padding	**0.45**	**0.53**	0.21	**0.62**	0.26	**0.414**

Table 5.5: benchamrked demands pre@100

of 0.392 and the traditional machine learning based pointwise approach with the balanced class weight with a precision@100 of 0.388. However, it is worth noting that out of the five demands, even though deep learning based list wise approach with a padded training dataset gave precision@100 > 0.5 for two demands also gave precision@100 < 0.3 for two other demands, of which one was pretty well handled by tradition machine learning based pointwise approach with oversampling that gave a precision@100 of 0.48 for the very same demand. This finding raises another question: whether a powerful ranking approach like list wise cannot generalize well over all kinds of demand-supplier data. Another interesting observation is that for the two demands where the winning approach's performance was low, the second runner-up model traditional machine learning pointwise approach with class weight balanced and no sampling performed exceptionally well and vice versa. RQ3 questioned whether one ranking model is sufficient to handle the diverse set of demand-supplier sets. These experiments answer the RQ3 that one ranking model cannot give a consistent performance across the demands.

Based on the performance of the twelve models on the test set and on the benchmarking demands, the top four winning ranking models in order are

1. Deep learning based listwise approach with padded train data.
2. Traditional Machine Learning based pointwise approach with undersampled train data
3. Traditional Machine Learning based pointwise approach with oversampled train data
4. Traditional Machine Learning based pointwise approach with class weight parameter set to balanced

The SHAP plots for these four models on the test set and also on individual benchmark demands are plotted which are shown in the Appendix section to reveal further insights of each of the models and to declare the champion model.

5.3 User Study

A user study was conducted where the domain experts were asked to rank the features as per their importance for considering a particular supplier to be relevant or irrelevant with respect to a particular demand. The consolidated features importance have been shown in the Table 5.6.

Elaborating it further, each domain expert had worked on atleast one demand from the benchmark set. The domain experts were asked to rank the demand characteristics, given in Section 5.1.1 that were considered by them while marking the suppliers as relevant or irrelevant during the benchmarking. These characteristics are given as different data fields. This is represented under the column 'domain experts' in the Table 5.6. Each of the SHAP plots that gave the feature importance for each demand in the benchmark set by each of the above mentioned four winning models is also represented with the column demand_id's being tabulated vertically and and the models being tabulated horizontally.

There are 40 features in the data, however, the demand characteristics are 10 and using token-based approach and phrase-based approach total_hits and max_scores for each of the features was calculated resulting in 10*4=40 features. While consolidating the feature importances, all the features were mapped back to their original demand characteristics. Each cell value represents the ordered set of the demand characteristics (top 5), with the first mentioned one being the most important characteristic for the respective model and the respective demand from the benchmark set.

5.3.1 Observations

1. All the domain experts ranked the data field `product` to be the most important demand characteristic. It can be seen that traditional ML pointwise oversampling model correctly considered product to be the most impacting feature. Also, deep learning based listwise approach with padding as the sampling technique, had `product` as the most important feature for the two of the demands 1976 and 1817. However, it is worth noting that `product` is considered important by all the models if not at rank one but at a lower rank.
2. Four out of five domain experts considered `company_type` as the second most important feature. deep learning based listwise approach with padding as the sampling technique has `company_type` in all of its demand characteristic set. All the other three models have `company_type` in their top five only for one demand 1817.
3. For demand 783, the domain expert considered `certificate_low_impact` important, however, none of the models had this feature in their respective sets. One reason could be the sparsity (approximately 0.8) of this data field. All the models correctly included `application_area_high_impact` and `location` in the important characteristic set except deep learning based listwise padding model that did not include `location`, but it did include `company_type`.
4. For demand 1928, apart from `product`, `location` and `extra_score_keywords` are important as per the domain expert handling this demand and these two have been

taken into account by all the models. The characteristic `manufacturing_processes_low_impact` have been considered as an impacting characteristic by all the models except by the deep learning based listwise padding.

5. For demand 1651, the characteristic `extra_score_keywords` is another important characteritic according to the domain expert and it correctly belongs to the characteristic set of all the models. The characteritics `location` is considered important by all the models except deep learning based listwise padding, however, this model rightly considered `company_type` which other models failed to capture. `manufacturing_processes_high_impact` is another important feature as per the domain expert and it belongs to the demand characteristic set of two models deep learning based listwise padding and the traditional ML pointwise undersampling.
6. For demand 1976, `location`, `manufacturing_processes_low_impact`, and `application_area_high_impact` have made to the list of important characteritics of the domain expert and they are correctly noted by all the models except deep learning based listwise padding which did not have `location` but had `company_type`.
7. For demand 1817, all the characteristics that the domain expert has considered while marking the supplier to be relevant or irrelevant are also correctly included in all the demand characteristic set of all the models in different models.

demand_id	Traditional ML pointwise no sampling	Traditional ML pointwise undersampling	Traditional ML pointwise oversampling	NN Listwise padding	domain experts
783	{extra_score_keywords, product, application_area_high_impact, location, manufacturing_processes_low_impact}	{extra_score_keywords, product, location, application_area_high_impact, manufacturing_processes_low_impact}	{product, extra_score_keywords, application_area_high_impact, location, manufacturing_processes_low_impact}	{application_area_high_impact, extra_score_keywords, company_type, product, manufacturing_processes_low_impact}	{product, company_type, certificate_low_impact, application_area_high_impact, location}
1928	{extra_score_keywords, application_area_high_impact, product, manufacturing_processes_low_impact, location}	{extra_score_keywords, product, application_area_high_impact, location, manufacturing_processes_low_impact }	{product, extra_score_keywords, application_area_high_impact, location, manufacturing_processes_low_impact }	{extra_score_keywords, application_area_high_impact, company_type, product, location }	{product, location, certificate_low_impact, extra_score_keywords, manufacturing_processes_low_impact}
1651	{extra_score_keywords, product, application_area_high_impact, location, manufacturing_processes_low_impact}	{extra_score_keywords, product, location, application_area_high_impact, manufacturing_processes_high_impact}	{product, application_area_high_impact, extra_score_keywords, location, manufacturing_processes_low_impact}	{extra_score_keywords, company_type, application_area_high_impact, manufacturing_processes_high_impact, product}	{product, company_type, location, extra_score_keywords, manufacturing_processes_high_impact}
1976	{extra_score_keywords, application_area_high_impact, product, location, manufacturing_processes_low_impact}	{extra_score_keywords, product, application_area_high_impact, location, manufacturing_processes_low_impact}	{product, extra_score_keywords, application_area_high_impact, location, manufacturing_processes_low_impact}	{product, extra_score_keywords, application_area_high_impact, manufacturing_processes_low_impact, company_type}	{product, company_type, location, manufacturing_processes_low_impact application_area_high_impact}
1817	{extra_score_keywords, product, application_area_high_impact, location, company_type}	{extra_score_keywords, product, location, application_area_high_impact, company_type}	{product, application_area_high_impact, location, extra_score_keywords, company_type}	{product, extra_score_keywords, application_area_high_impact, company_type, location }	{product, company_type, extra_score_keywords, application_area_high_impact, location}

Table 5.6: Demand characteristics importance as per the domain experts and the top 4 winning models.

6 Discussion

Fig. 6.1 plots the performance of the four winning models based on the Table 5.5. It is interesting to see that all the pointwise approaches using traditional machine learning, which are basically classification models made to the list of top 4. This can be due to the binary ground truth. It can be seen that Traditional ML pointwise undersampling performs the most uniformly across all the demands in the benchmark dataset.

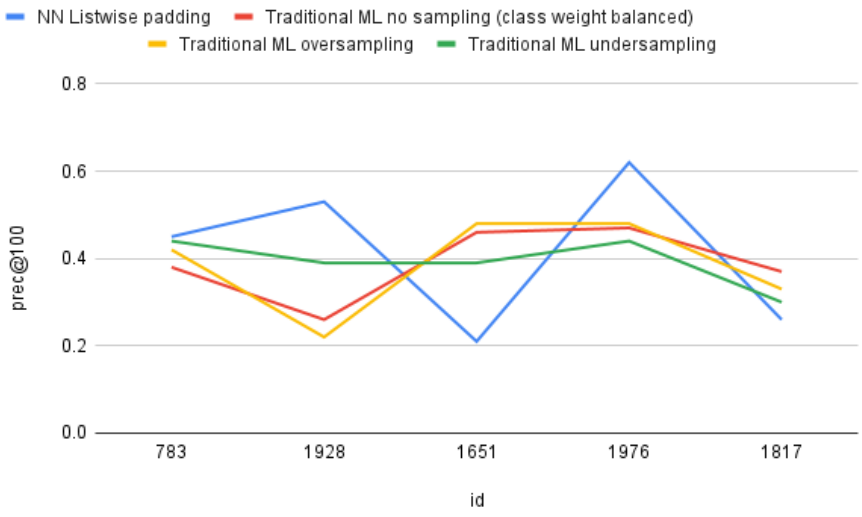


Figure 6.1: Performance of winning model based on Table 5.5

For the demand 783, based on the user study no clear conclusions can be made as all the important demand characteristics as per the domain experts are spread across the models. Hence, based on the P@100 deep learning based listwise padding and traditional ML pointwise undersampling with values 0.46 and 0.45 can be considered winners.

For the demand 1928, based on the user study, deep learning based listwise padding is excluded as the characteristics it considered to be most impacting are different than the characteristics considered important by the domain expert. The characteristic `location` should be before `manufacturing_processes_low_impact` which is very well captured by tradtional ML pointwise undersampling and traditional ML pointwise oversampling. And based on P@100 traditional ML pointwise undersampling is better for this demand as it has the score of 0.39 compared to the score of 0.22 by traditional ML pointwise oversampling.

For the demand 1651, based on the observations from the user study, the demand characteristics sets of traditional ML pointwise undersampling and deep learning based listwise

padding show a close resemblance to the demand characteristics set given by the domain expert. And based on P@100 score, traditional ML pointwise undersampling show a better performance with a score of 0.39 that deep learning based listwise padding with a score of 0.21

For the demand 1976, based on the user study, traditional ML pointwise no sampling model can be excluded as it ranks the characteristic `product` way lower. And for the remaining three models, based on the P100, deep learning based listwise padding gives the highest score of 0.62, followed by a score of 0.48 by traditional ML pointwise oversampling and a score of 0.44 by tradtitional ML pointwise undersampling.

For the demand 1817, no conclusive remarks can be drawn from the user study as all the models have all the important demand characteristics as required by the domain expert. Based on the P@100 score, traditional ML pointwise no sampling gives a score of 0.37, followed by 0.33 by tradtional ML pointwise oversampling, 0.30 by traditional ML pointwise undersampling and 0.26 by deep learning based listwise padding.

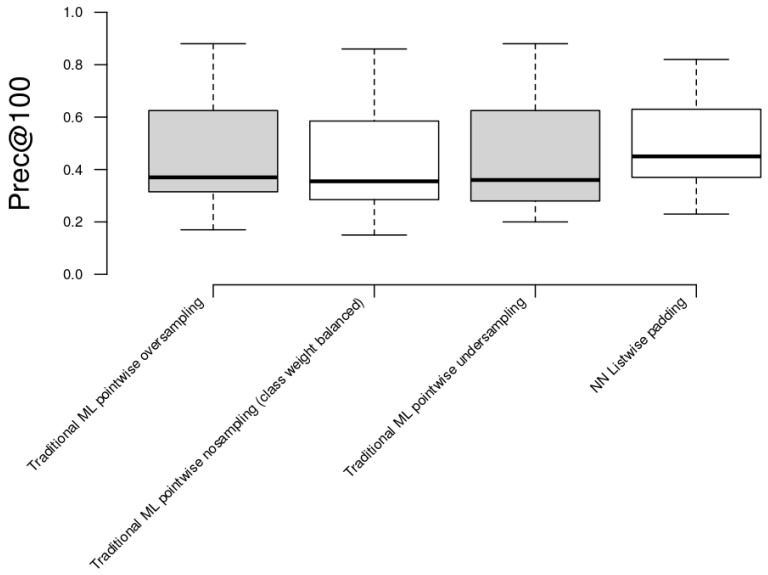


Figure 6.2: P@100 score on the test set of the models

Further the analysis of the performance of the four models on the test set is done. A box plot representing the P@100 score of all the models is shown in the Fig. 6.2. Comparatively, the deep learning based listwise padding had a more uniform performance across all the demands in the test set, followed by traditional ML pointwise undersampling.

Upon further investigation of the test set it was observed that 11 out of 32 demands had less than 100 relevant suppliers so a modified Precision score was calculated such that for the demands with less than 100 relevant suppliers, the precision score was the number of relevant suppliers in top 100 as given by the model in proportion to the total number of relevant suppliers instead of 100. This is shown in the equation below.

$$Modified\ P@k = \frac{\{number\ of\ relevant\ suppliers\ till\ position\ k\}}{min(k,total\ number\ of\ relevant\ suppliers\ associated\ with\ the\ demand)}$$

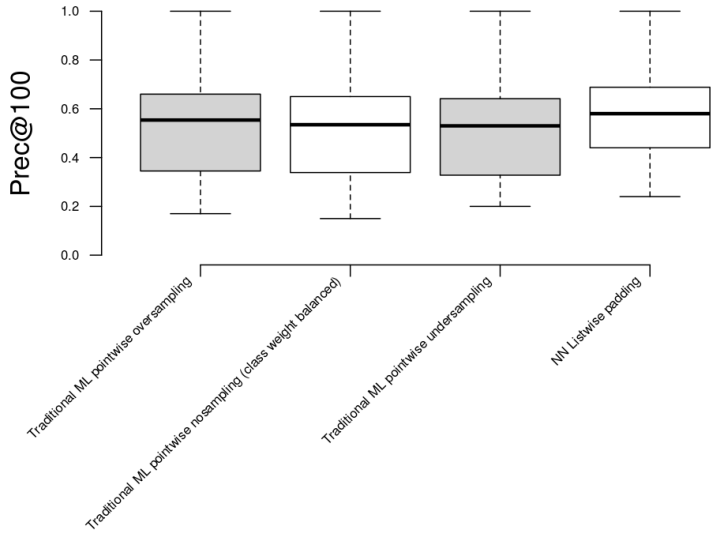


Figure 6.3: Modified Precision score the test set of the models

Another way would be to completely exclude the 11 demands as in the real world setting, it is mostly expected to find at least 100 relevant suppliers. After evaluating the performance of the models based on this modified score a box plot of this modified score is shown in the Fig. 6.3 and the box plot of the demands with greater than 99 relevant suppliers is shown in the Fig. 6.4.

However, the conclusions that can be drawn from these two figures are the same as the ones with the original P@100 score where deep learning based listwise padding model showed a uniformly spread performance followed by traditional ML pointwise undersampling. Considering the discussions above, it can be said that traditional ML pointwise

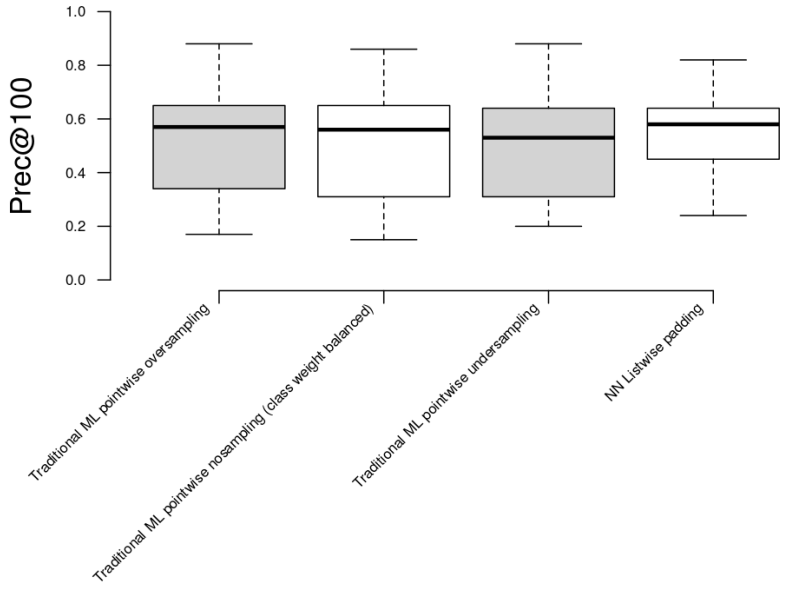


Figure 6.4: P@100 score on the test set of the models where test set only consisted of the demands with greater than 99 relevant suppliers

undersampling is the champion model as it has shown a decent performance in the user study as well on the benchmark and the test set.

7 Conclusion and Future Work

Each demand is different and hence the most important thing that needs to be done is to analyse the demands and group them based on their similarity. There were some demands where some of the models gave near to perfect performance. If the demands are grouped then for each group of demands a model could be assigned.

Another important thing would be try out an ensemble of the models as it can be seen from the Table 5.5 that the performance of some of the models is complementary to each other. A quick check with various loss functions for the Deep learning based ranking models and XGBoost for Traditional ML based pointwise approach can be done. All the demands that were used for training belong to April-2021, retraining the models with new demands may prove beneficial.

Each domain expert had a different set of important demand characteristics, hence models with various combinations of demand characteristics/features could be trained and as per the demand characteristics importance given by the end user/domain expert, the respective model could be chosen for ranking.

The way features are generated is based on naive match of the tokens or the entire keyword on the respective websites. However, this could also add some noise to the generated features. Let's consider the example where the demand is looking for suppliers in Germany. It is possible that the supplier is serving another product in Germany and not the one that the demand is looking for and the word Germany is present in that context. One solution would be to apply Named Entity Recognition (NER) [NS07]. When it comes to the ground truth, instead of having binary labels, it would be interesting to see the performance of various LTR approaches on the demand-supplier data where the suppliers have a relevancy score.

There are also quite a few advancements in the LTR approaches [ZWB+20] which could be tried to solve the problem statement of ranking suppliers based on the demand.

Appendices

A. 79 handcrafted features

demand__score__keywords__score__mean

demand__score__keywords__score__mean__50__sim

demand__score__keywords__score__mean__10__sim

demand__score__keywords__score__mean__50__dissim

demand__score__keywords__score__mean__10__dissim

demand__search__keywords__score__mean

demand__search__keywords__score__mean__50__sim

demand__search__keywords__score__mean__10__sim

demand__search__keywords__score__mean__50__dissim

demand__search__keywords__score__mean__10__dissim

demand__product__keywords__score__mean

demand__product__keywords__score__mean__50__sim

demand__product__keywords__score__mean__10__sim

demand__product__keywords__score__mean__50__dissim

demand__product__keywords__score__mean__10__dissim

demand__application__areas__high__impact__score__mean

demand__application__areas__high__impact__score__mean__50__sim

demand__application__areas__high__impact__score__mean__10__sim

demand__application__areas__high__impact__score__mean__50__dissim

demand__application__areas__high__impact__score__mean__10__dissim

demand__extra__score__keywords__score__mean

demand__extra__score__keywords__score__mean__50__sim

demand__extra__score__keywords__score__mean__10__sim

demand__extra__score__keywords__score__mean__50__dissim

demand__extra__score__keywords__score__mean__10__dissim

demand__manufacturing__processes__high__impact__score__mean

demand__manufacturing__processes__high__impact__score__mean__50__sim

demand__manufacturing__processes__high__impact__score__mean__10__sim

demand__manufacturing__processes__high__impact__score__mean__50__dissim

demand__manufacturing__processes__high__impact__score__mean__10__dissim

demand__manufacturing__processes__low__impact__score__mean

demand__manufacturing__processes__low__impact__score__mean__50__sim

demand__manufacturing__processes__low__impact__score__mean__10__sim

demand__manufacturing__processes__low__impact__score__mean__50__dissim

demand__manufacturing__processes__low__impact__score__mean__10__dissim

demand__business__areas__score__mean

demand__business__areas__score__mean__50__sim

demand__business__areas__score__mean__10__sim

demand__business__areas__score__mean__50__dissim

demand__business__areas__score__mean__10__dissim

demand__application__areas__score__mean

demand__application__areas__score__mean__50__sim

demand__application__areas__score__mean__10__sim

demand__application__areas__score__mean__50__dissim

demand__application__areas__score__mean__10__dissim

is__google__homepage__hit

reachable__for__downloader

candidate__data__findings__len

candidate__data__discovery__findings__len

candidate__data__emis__score

candidate__data__static__score

serpstat__alpha

serpstat__beta

matched__candidate__score__keywords__len

matched__certificates__high__impact

matched__certificates__low__impact

purchaser__names__match__mean

typical_customers_match_ratio

product_keywords_match_ratio

score_keywords_match_ratio

search_keywords_match_ratio

extra_score_keywords_match_ratio

manufacturing_processes_match_ratio_high_impact

application_areas_match_ratio_high_impact

demand_business_areas_similarity_weight_1

demand_business_areas_similarity_weight_2

demand_business_areas_similarity_weight_3

demand_business_areas_similarity_weight_4

demand_business_areas_similarity_weight_5

demand_business_areas_similarity_weight_6

demand_business_areas_similarity_weight_7

demand_business_areas_similarity_weight_8

demand_business_areas_similarity_weight_9

demand_business_areas_similarity_weight_10

business_areas_total_score

application_areas_total_score

products_total_score

candidate_data_findings_page_number

candidate_company_type_include_ratio

Bibliography

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

[AB18] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: a survey on explainable artificial intelligence (xai). *IEEE access*, 6:52138–52160, 2018.

[ADRDS+20] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information fusion*, 58:82–115, 2020.

[Aga05] Shivani Agarwal. *A study of the bipartite ranking problem in machine learning*. University of Illinois at Urbana-Champaign, 2005.

[AMJ18] David Alvarez Melis and Tommi Jaakkola. Towards robust interpretability with self-explaining neural networks. *Advances in neural information processing systems*, 31, 2018.

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[Bur10] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

[BYRN+99] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[CDS18] J Shane Culpepper, Fernando Diaz, and Mark D Smucker. Report from the third strategic workshop on information retrieval (swirl).(2018). 2018.

[CHB+15] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, et al. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4):1–4, 2015.

[CHH+17] Heng-Tze Cheng, Zakaria Haque, Lichan Hong, Mustafa Ispir, Clemens Mewald, Illia Polosukhin, Georgios Roumpos, D Sculley, Jamie Smith, David Soergel, et al. Tensorflow estimators: Managing simplicity vs. flexibility in high-level machine learning frameworks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1763–1771, 2017.

[CQL+07] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136, 2007.

[CZH+12] Fuxing Cheng, Xin Zhang, Ben He, Tiejian Luo, and Wenjie Wang. A survey of learning to rank for real-time twitter search. In *Joint International Conference on Pervasive Computing and the Networked World*, pages 150–164. Springer, 2012.

[DCLT16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Bidirectional encoder representations from transformers. 2016.

[DDF+90] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[DLH19] Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning. *Communications of the ACM*, 63(1):68–77, 2019.

[FM08] Peter Flach and Edson Matsubara. On classification, ranking, and probability estimation. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum fr Informatik, 2008.

[GA19] David Gunning and David Aha. Darpa's explainable artificial intelligence (xai) program. *AI magazine*, 40(2):44–58, 2019.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[GDR16] Artem Grotov and Maarten De Rijke. Online learning to rank for information retrieval: Sigir 2016 tutorial. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 1215–1218, 2016.

[GMR+18] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):1–42, 2018.

[HBLH94] William Hersh, Chris Buckley, TJ Leone, and David Hickam. Ohsumed: An interactive retrieval evaluation and new large test collection for research. In *SIGIR'94*, pages 192–201. Springer, 1994.

[Hey97] Francis Heylighen. Occam's razor. *Principia cybernetica web*, 1997.

[HWZL08] Chuan He, Cong Wang, Yi-Xin Zhong, and Rui-Fan Li. A survey on learning to rank. In *2008 International Conference on Machine Learning and Cybernetics*, volume 3, pages 1734–1739. Ieee, 2008.

[IC14] Muhammad Ibrahim and Mark Carman. Undersampling techniques to rebalance training data for large scale learning-to-rank. In *Asia Information Retrieval Symposium*, pages 444–457. Springer, 2014.

[IC16] Muhammad Ibrahim and Mark Carman. Comparing pointwise and listwise objective functions for random-forest-based learning-to-rank. *ACM Transactions on Information Systems (TOIS)*, 34(4):1–38, 2016.

[Joa06] Thorsten Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226, 2006.

[KJ+13] Max Kuhn, Kjell Johnson, et al. *Applied predictive modeling*, volume 26. Springer, 2013.

[KKH22] Dominik Kreuzberger, Niklas Kühl, and Sebastian Hirschl. Machine learning operations (mlops): Overview, definition, and architecture. *arXiv preprint arXiv:2205.02302*, 2022.

[KWG+18] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, et al. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International conference on machine learning*, pages 2668–2677. PMLR, 2018.

[LEC+20] Scott M Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable ai for trees. *Nature machine intelligence*, 2(1):56–67, 2020.

[Liu11] Tie-Yan Liu. Learning to rank for information retrieval. 2011.

[LL17] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017. URL: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[Llo82] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[LNV+18] Scott M Lundberg, Bala Nair, Monica S Vavilala, Mayumi Horibe, Michael J Eisses, Trevor Adams, David E Liston, Daniel King-Wai Low, Shu-Fang Newman, Jerry Kim, et al. Explainable machine-learning predictions for the prevention of hypoxaemia during surgery. *Nature biomedical engineering*, 2(10):749–760, 2018.

[MHGG10] Michael McCandless, Erik Hatcher, Otis Gospodnetić, and O Gospodnetić. *Lucene in action*, volume 2. Manning Greenwich, 2010.

[Mit99] Tom M Mitchell. Machine learning and data mining. *Communications of the ACM*, 42(11):30–36, 1999.

[MM97] Tom M Mitchell and Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

[Mol18] Christoph Molnar. others. 2018. *Interpretable machine learning: A guide for making black box models explainable. E-book at< https://christophm. github. io/interpretable-ml-book/>, version dated*, 10, 2018.

[Mon21] Robert Munro Monarch. *Human-in-the-Loop Machine Learning: Active learning and annotation for human-centered AI*. Simon and Schuster, 2021.

[MRS10] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. *Natural Language Engineering*, 16(1):100–103, 2010.

[NS07] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[OFG+17] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139*, 2017.

[OGGdR+21] Alexandra Olteanu, Jean Garcia-Gathright, Maarten de Rijke, Michael D Ekstrand, Adam Roegiest, Aldo Lipani, Alex Beutel, Alexandra Olteanu, Ana Lucic, Ana-Andreea Stoica, et al. Facts-ir: fairness, accountability, confidentiality, transparency, and safety in information retrieval. In *ACM SIGIR Forum*, volume 53, pages 20–43. ACM New York, NY, USA, 2021.

[P18] Aruna P. The amazon a9 algorithm. 08 2018.

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[PBW+19] Rama Kumar Pasumarthi, Sebastian Bruch, Xuanhui Wang, Cheng Li, Michael Bendersky, Marc Najork, Jan Pfeifer, Nadav Golbandi, Rohan Anil, and Stephan Wolf. Tf-ranking: Scalable tensorflow library for learning-to-rank. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2970–2978, 2019.

[PHB+18] Alun Preece, Dan Harborne, Dave Braines, Richard Tomsett, and Supriyo Chakraborty. Stakeholders in explainable ai. *arXiv preprint arXiv:1810.00184*, 2018.

[QKF20] Nunung Nurul Qomariyah, Dimitar Kazakov, and Ahmad Fajar. Predicting user preferences with xgboost learning to rank method. pages 123–128, 12 2020. doi:10.1109/ISRITI51436.2020.9315494.

[Rob97] Stephen E Robertson. Overview of the okapi projects. *Journal of documentation*, 1997.

[Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[SHG+15] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, 28, 2015.

[SK18] Mehrnoush Barani Shirzad and Mohammad Reza Keyvanpour. A systematic study of feature selection methods for learning to rank algorithms. *International Journal of Information Retrieval Research (IJIRR)*, 8(3):46–67, 2018.

[SPM19] Amal Saadallah, Florian Priebe, and Katharina Morik. A drift-based dynamic ensemble members selection using clustering for time series forecasting. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 678–694. Springer, 2019.

[SWKA20] Jaspreet Singh, Zhenye Wang, Megha Khosla, and Avishek Anand. Valid explanations for learning to rank models. *arXiv preprint arXiv:2004.13972*, 2020.

[TTJ+07] Liu Tieyan, Qin Tao, Xu Jun, et al. Letor: Benchmark dataset for research on learning to rank for information retrieval. In *Proceedings of the Workshop on Learning to Rank for Information Retrieval*, pages 137–145, 2007.

[VG19] Manisha Verma and Debasis Ganguly. Lirme: locally interpretable ranking model explanation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1281–1284, 2019.

[VRH17] Jan N Van Rijn and Frank Hutter. An empirical study of hyperparameter importance across datasets. In *AutoML@ PKDD/ECML*, pages 91–98, 2017.

[WS] Jingang Wang and Dandan Song. Bit and msra at trec kba ccr track 2013.

[ZCD+18] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, et al. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018.

[ZWB+20] Honglei Zhuang, Xuanhui Wang, Michael Bendersky, Alexander Grushetsky, Yonghui Wu, Petr Mitrichev, Ethan Sterling, Nathan Bell, Walker Ravina, and Hai Qian. Interpretable learning-to-rank with generalized additive models. *arXiv preprint arXiv:2005.02553*, 2020.

Statement of Authorship / Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit selbstständig und außschließlich unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.
Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder einer anderen Prüfungsbehörde vorgelegt oder noch anderweitig veröffentlicht.

UnterschriftDatum