FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Form filling recommendation using machine learning techniques

Alexandre da Silva Lima

U. PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: João Pedro Carvalho Leal Mendes Moreira

March 4, 2019

Form filling recommendation using machine learning techniques

Alexandre da Silva Lima

Mestrado Integrado em Engenharia Informática e Computação

March 4, 2019

Abstract

Nowadays, most enterprise or management software, online or offline, requires filling forms in order to interact with the system, like a new entry, for example. In these type of systems, this is, most of the times, a repetitive activity, which means that the likelihood of a certain user filling the same or an identical form is high. Some forms use already known values to the system, like the most used or most recent value in some field, to facilitate its filling.

In the last years a lot of research has been done in Machine Learning (ML) and how it can be used to facilitate human computer interaction. ML can be used to solve very distinct problems, with ranking problems being one of them. Most ranking approaches are generalizations of classification or regression problems. One area that focuses on solving ranking problems is Preference Learning (PL). PL main objective is to predict the preference of some items in relation to others.

This thesis proposes a suggestion system for these forms, in order to speed up its filling process. The suggestion system is based on multiple models for each field, in order to adapt to different states of the filling process, to rank each possible entry for each unfilled field, trained with past form entries. These models will use as features the currently filled values and also some context variables.

In order to test this hypothesis we used two approaches. First, a PL approach, ranking by pairwise comparison (RPC), that uses the preference knowledge of a label to another and by pairwise comparison obtains the ranking of a set of labels. Then, we used a more simple approach, which we called ranking by probability estimation (RPE), that uses the probability results of a multi-class classification algorithm to generate the ranking of the same set of labels. Both methods were evaluated in regards to the training time, prediction time and the ranking quality. The ranking quality was evaluated by Normalized Discounted Cumulative Gain (NDCG).

Tests show that the RPE approach performs better then RPC in almost all situations, in training and prediction time, while being competitive in ranking quality. Due to the nature of the problem, a real time suggestion system, and also due to the fact that a form usually has multiple entries possible for a certain field, RPC becomes infeasible, because the prediction time scales quadratically with the size of the labels set. We then tested the RPE approach with three different classification algorithms, C5.0, Naive Bayes and Random Forest, to determine which would be the best for the problem at hand. Results show that C5.0 is the most balanced of the three, in the three metrics evaluated. Random Forest also shows competitive NDCG scores, at the cost of training time. Still the overall training time of the whole process was high.

With that in mind, we decided to try a more smart approach, in terms of the models used for each state of form filling. This approach, uses the importance of each feature to split them into relevant and irrelevant features, training only with the relevant ones. Tests show that, this last approach, while maintaining similar NDCG values, reduces to about one forth the time it takes to train the whole system.

Resumo

Atualmente, a maior parte dos programas de gestão, online ou offline, requerem o preenchimento de formulários de forma a interagir com o sistema, como, por exemplo, uma nova entrada. Neste tipo de sistemas, isto é, a maior parte das vezes, uma atividade repetitiva, o que significa que a probabilidade de um utilizador preencher o mesmo ou um formulário idêntico é elevada. Alguns formulários utilizam valores já conhecidos pelo sistema, como o valor mais recente ou o valor mais usado, para facilitar o seu preenchimento.

Nos últimos anos muita pesquisa foi feita na área de Machine Learning (ML). ML pode ser utilizado para resolver problemas distintos, sendo os problemas de ranking um deles. Uma área que se foca em resolver problemas de ranking é o Preference Learning (PL). O principal objetivo de PL é prever a preferência de alguns items em relação a outros.

Esta tese propõe um sistema de sugestões para estes formulários, de forma a acelerar o processo de preenchimento dos mesmos. Este sistema é baseado em múltiplos modelos para cada campo, de forma a adaptar-se a diferentes estados do processo de preenchimento, para ordenar cada entrada possível para cada campo não preenchido. Estes modelos irão utilizar como atributos os valores actualmente preenchidos e também alguns atributos de contexto.

De forma a testar esta hipótese utilizamos duas abordagens. Primeiro, uma abordagem de PL, ranking by pairwise comparison (RPC), que utiliza o conhecimento da preferência de um label por outro e através de comparação entre pares obtém um ranking de um conjunto de labels. Depois, usamos uma abordagem mais simples, que foi chamada de ranking by probability estimation (RPE), que utiliza os resultados de probabilidade de um algoritmo de classificação multi-class para gerar o ranking do mesmo conjunto de labels. Os dois métodos foram avaliados em termos de tempo de treino, tempo de previsão e qualidade do ranking. A qualidade do ranking foi avaliada por Normalized Discounted Cumulative Gain (NDCG).

Os testes mostram que o RPE tem melhor desempenho que o RPC em praticamente todas as situações, em tempo de treino e de previsão, enquanto se mantém competitiva em qualidade de ranking. Devido à natureza do problema, um sistema em tempo real, e também devido ao facto de um formulário geralmente poder ter várias possibilidades de preenchimento para um certo campo, o RPC torna-se inviável, pois o tempo de previsão escala quadraticamente em relação ao tamanho do conjunto de labels possíveis. Depois, testamos a abordagem RPE com 3 algoritmos de classificação, C5.0, Naive Bayes e o Random Forest, para determinar qual seria o melhor para o problema em questão. Os resultados demonstram que o C5.0 é o mais equilibrado dos três, nas três métricas avaliadas. O Random Forest também demonstra resultados competitivos de NDCG, em prejuízo do tempo de treino. Mesmo assim, em geral, o tempo total de treino foi elevado.

Como tal, decidimos tentar uma abordagem mais inteligente, em relação aos modelos utilizados para cada estado de preenchimento do formulário. Esta abordagem, utiliza a importância de cada atributo para os dividir em atributos relevantes e irrelevantes, treinando apenas com os relevantes. Os testes mostram que, esta última abordagem, enquanto mantém valores de NDCG equivalentes, reduz em cerca de um quarto o tempo que demora a treinar o sistema inteiro.

Acknowledgments

In this dissertation I had the support of people to whom I am immensely grateful.

To my supervisor, Prof. João Moreira, I am thankful for his availability, opinions, constructive criticism and ideas that were fundamental to this dissertation.

To the company Infraspeak, for coming up with the idea, for the constant support, for the materials supplied, for giving me a quiet space to work, and above all, for their friendliness. As such, my thanks to Luís Martins and all his team.

To my friends, Pedro Costa, Sérgio Nascimento and Miguel Monteiro, I thank the unconditional friendship.

To my parents, my grandparents and my brother I thank all the support throughout this journey.

And last but not least, to my girlfriend, Diana, and her daughter, Carmen, I thank all the support and understanding, crucial to the success of this work.

Alexandre da Silva Lima

Contents

List of Figures

List of Tables

Abbreviations

ERP	Enterprise Resource Planning
LR	Label Ranking
ML	Machine Learning
NDCG	Normalized Discounted Cumulative Gain
RPC	Ranking by Pairwise Comparisons
PL	Preference Learning
UI	User Interface
RPE	Ranking by Probability Estimation
DFS	Direct Features

Chapter 1

Introduction

In this chapter we will introduce the main concepts, the problem at hand, the main motivation to solve it and the goals of this work.

1.1 Forms in software and web apps

Electronic forms are embedded in many of today's user interfaces (UI), enabling users to engage in online communities, e-commerce, and productivity software (Wroblewski, 2008). This is also true for many enterprise and management software, like Enterprise resource planning (ERP) software. Web forms remain one of the core interaction elements between users and website owners (Seckler et al., 2012). Although web forms are very common, people usually do not like to fill them out (Wroblewski, 2008).

In the case of enterprise software, this filling process can be quite repetitive and time consuming. What can be done to reduce this burden?

This work is being done with the collaboration of Infraspeak. Infraspeak is a company that commercializes a technical operations platform. This software uses forms as the main type of interaction with the system.

1.2 Motivation

There are many aspects to form design (Bargas-Avila et al., 2010). We will be focused in what can be done to speed up the filling process of each field, resulting in improving also the speed of filling the whole form. This also results in users spending less time in the filling process, freeing them to other activities, improving productivity and user experience.

Our approach focuses on State of the Art ML techniques and its application on the referred problem. We expect this approach to be significantly superior to the usual most recently used or most commonly used values that most forms suggest. As ML evolves and sees its use in solving complex problems, sometimes even in simpler problems ML can be a good solution.

1.3 Goals of this work and our approach to the problem

In the recent years a lot of interest was shown in the ML area. These days ML is used in a vast amount of areas. From the human computer interaction field some usages of machine learning can be seen in Recommender systems (Ricci et al., 2015) or Chatbots (Shawar and Atwell, 2007).

In this work we will explore the possibility of using ML techniques, LR and other approaches, to obtain a ranking of each possible input for each unfilled field, in order to generate suggestions. Our approach focuses on the usage of the already obtained knowledge from past user entries on each form. With this knowledge we are able to train multiple ML models, one for each stage of the filling process. Each stage refers to the amount of information, on a specific filling process, we have, meaning that, for instance, if a form has already two fields filled, we have more information than if it had only one. That justifies the reasoning behind the multiple models system.

With this approach we will be able to develop a suggestion system, that can suggest a ranked list of possible entries for each unfilled field and can adapt its suggestions if a user fills a previously unfilled field.

The main goal of this work is to test the viability of the proposed solution. In order to do that we will test some different approaches and ranking algorithms, while evaluating the ranking quality of each, and also the training and prediction time, as this is a real time suggestion system. For this test a real dataset will be used, provided by Infraspeak, from a single form, that encapsulates multiple problems due to its own nature.

1.4 Document structure

Besides the introduction, this document will contain 4 more chapters.

In Chapter 2 we will review the State of the Art, specially related to ML, the algorithms and performance metrics, and also what is usually done in forms.

Chapter 3 will be focused on the careful specification of the problem and how we intend to solve it. It will explore the data preparation, training and evaluation phases of each model for each approach.

Chapter 4 will be centered around the experiments done during this work, its specification and results.

Lastly, in Chapter 5 is the summary of the document, as well as, the enunciation of possible threats and some final thoughts.

Chapter 2

State of the Art

The main focus of this chapter will be in the review of the current state of the art, relating to ML specially, and also comparing our work with other similar works.

2.1 What has been done in form suggestion

This suggestion problem was defined as predictive models for form filling by Ali and Meek (2009). A predictive model of form filling aims to reduce the amount of time a user spends filling out a form by predicting the values of fields on the form and using these predictions to make suggestions to the form filler (Ali and Meek, 2009).

The most common approaches to this problem is suggesting for each field the most commonly used value or the most recently used value (Ali and Meek, 2009). This assumes independence between fields which might result in a bad suggestion.

This paper also presents another solution called Collaborative and Contextually Frequently Used model (CCFU) (Ali and Meek, 2009). While this model assumes dependency between fields it also assumes that a field that is after the predicted field does not affect the prediction. As such it assumes a certain order in filling the form, that may not always be true.

Our approach assumes some degree of dependency between fields and does not assume any special order in the filling process.

It must be noted that the referred paper's forms are not of the same type we are dealing with. It deals with more general forms, like registration forms.

2.2 Machine Learning

The study and computer modeling of learning processes in their multiple manifestations constitutes the subject matter of machine learning (Michalski et al., 2013). It is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories (Bishop, 2006).

ML tasks are typically classified into two categories:

- Unsupervised Learning
- Supervised Learning

Unsupervised machine learning is the machine learning task of inferring a function that describes the structure of unlabeled data. By applying these unsupervised (clustering) algorithms, researchers hope to discover unknown, but useful, classes of items (Jain et al., 1999).

Supervised machine learning or Supervised Learning is the search for algorithms that reason from externally supplied instances to produce general hypotheses, which then make predictions about future instances. In other words, the goal of supervised learning is to build a concise model of the distribution of class labels in terms of predictor features (Kotsiantis et al., 2007).

Classification is a typical supervised learning task. The classification problem is that of learning the structure of a data set of examples, already partitioned into groups. The learning of these categories is typically achieved with a model. This model is used to estimate the group identifiers (or class labels) of one or more previously unseen data examples with unknown labels (Aggarwal, 2015).

Another common supervised learning task is Regression. Regression relates to learning problems where the target variable is expressed in a quantitative scale.

Lastly, another supervised learning task is Preference Learning.

2.3 Preference Learning

Preference learning refers to the task of learning to predict an order relation on a collection of objects (alternatives). Among the problems in the realm of preference learning, the task of "learning to rank" has probably received the most attention in the machine learning literature in recent years (Fürnkranz and Hüllermeier, 2011).

Although still a recent topic, preference learning can be divided into 3 different tasks:

- **Label Ranking** – learning a mapping from instances to rankings over a predefined set of labels (Fürnkranz et al., 2008).
- **Instance Ranking** – produce a ranking in which instances from higher classes precede those from lower classes (Fürnkranz and Hüllermeier, 2011).
- **Object Ranking** – learning functions used to sort objects from example orders (Kamishima et al., 2010), commonly used in information retrieval applications.

This PL task classification was proposed by Fürnkranz and Hüllermeier (2011).

We will now focus on Label Ranking, as it is our main concern.

2.4 Label Ranking

As stated in section 2.3 LR is the task of learning a mapping from instances to rankings over a predefined set of labels (Fürnkranz et al., 2008). Rankings are also sometimes referred as total orders, as in Vembu and Gärtner (2010). The ordering should be performed in accordance with some notion of relevance of the labels. That is, a label deemed relevant to an instance should be ranked higher than a label which is considered less relevant (Dekel et al., 2004). An example of this can be seen in figure 2.1.

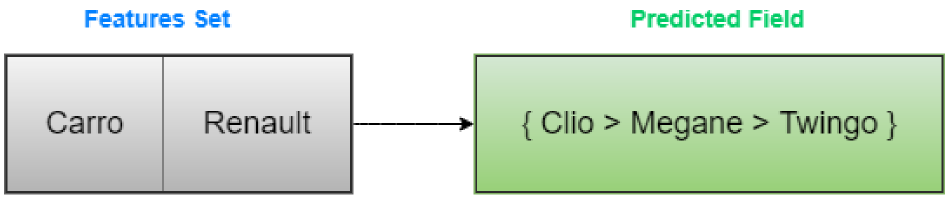


Figure 2.1: Label ranking result example

The training information consists of a set of instances for which (partial) knowledge about the associated preference relation is available (Hüllermeier et al., 2008).

2.4.1 Related Problems

Label ranking is a very interesting problem as it associates with three important supervised learning problems, including multiclass classification, multilabel classification and multilabel ranking (Zhou et al., 2014).

- **Multiclass Classification** – refers to assigning each of the observations into one of k classes (Wu et al., 2004).
- **Multilabel Classification** – supervised learning problem where an instance may be associated with multiple labels (Read et al., 2011).
- **Multilabel Ranking** – refines a multilabel classification in the sense that, while the latter only splits a predefined label set into relevant and irrelevant labels, the former furthermore puts the labels within both parts of this bipartition in a total order (Brinker and Hüllermeier, 2007).

The results of a LR algorithm can be easily used to solve a multiclass classification problem. Multilabel ranking is a mixture of multilabel classification and LR.

2.4.2 Data Preparation

As in any supervised learning problem, the training data is assumed to be labeled. In the case of LR this labeling, as was referenced in 2.4, requires the knowledge of atleast the partial order for each training instance.

It can be obtained explicitly, by asking someone to judge the relevance of each label (Niu et al., 2012), or implicitly, by making judgements based on the available data (Rendle et al., 2009).

2.4.3 Label Ranking Algorithms

There are two main approaches to the problem of LR: methods that transform the ranking problem into multiple binary problems and methods that were developed or adapted to treat the rankings as target objects, without any transformation (de Sá et al., 2017). Other LR algorithm categorization can be seen in Vembu and Gärtner (2010) and Zhou et al. (2014). An example of a transformation into multiple binary problems is ranking by pairwise comparisons (RPC) (Hüllermeier et al., 2008), while an example of ranking as target objects can be seen in Naive Bayes (Aiguzhinov et al., 2010). Almost all algorithms are derived and adapted from other typical ML problems.

Apart from the already mentioned algorithms more can be found such as Decision Trees (Cheng et al., 2009), Ranking Forests (de Sá et al., 2017), Instance based methods (Cheng et al., 2009), Log-linear Models (Dekel et al., 2004), SVM (Shalev-Shwartz and Singer, 2006) and k-Nearest Neighbor (Brazdil et al., 2003).

2.4.3.1 Ranking by Pairwise Comparison

As stated before, one of the classical LR algorithms is the ranking by pairwise comparison (RPC) algorithm. RPC first induces a binary preference relation from suitable training data using a natural extension of pairwise classification. A ranking is then derived from the preference relation thus obtained by means of a ranking procedure, whereby different ranking methods can be used for minimizing different loss functions (Hüllermeier et al., 2008). In essence, RPC for every pair (λ_i, λ_j) of labels of the ranking set, extracts knowledge where a preference relation from one to another is known and trains a model.

The model is trained with all examples for which either $\lambda_i \succ \lambda_j$ or $\lambda_j \succ \lambda_i$ is known. Examples for which nothing is known about the preference between λ_i and λ_j are ignored (Hüllermeier et al., 2008). If we have a tie between ranks no preference knowledge is known and this means those instances should be ignored.

During prediction the final ranking can be obtained by a simple voting strategy where

$$S_i(i,j) = \begin{cases} 1 & if\ \lambda_i \succ \lambda_j \\ 0 & if\ \lambda_j \succ \lambda_i \end{cases} \tag{2.1}$$

then a score for each possible target is obtained and ranked accordingly. Alternatively, one may also employ base classifiers that map into the unit interval [0, 1] instead of 0, 1, and thereby assign a valued preference relation (Hüllermeier et al., 2008). Instead of assuming a win or a loss in every pairwise relation it uses the probability of being of a x class in the scoring function of that class.

The number of pairwise models needed for RPC can be calculated as

$$NModels = \frac{n(n-1)}{2} \tag{2.2}$$

where n is the number of target classes. So, the number of needed models for RPC has quadratic growth.

2.5 Other Machine Learning approaches to Ranking

Many Classification algorithms are also able to generate class probability estimates for each possible label. Some authors defend the idea of exploiting the connection between probability estimation and ranking (Cheng and Hüllermeier, 2012). Indeed, ranking is in a sense in-between classification and probability estimation or, stated differently, can be seen as an intermediate step from classification to probability estimation (Flach, 2007). With the knowledge of the probability estimation of each class we can induce a ranking. For example, (Schapire and Singer, 2000) included an ad hoc extension to multilabel ranking in their experimental setup by ordering labels according to decreasing confidence scores (Fürnkranz et al., 2008).

This approach assumes having a fully labeled dataset, in order to treat the problem as a Classification problem and generating the ranking from class probability estimates.

2.5.1 Probability Estimation

Probability estimation is a metric that represents the likelihood of each class belonging to the test instance. These probabilities have numerous applications, including ranking, expected utility calculations, and classification with unequal costs (Olson and Wyner, 2018).

Probability estimation is calculated in different ways depending on the modeling algorithm used.

For example, when classifying a test example, C4.5 and other decision tree methods assign by default the raw training frequency $p = \frac{k}{n}$ as the score of any example that is assigned to a leaf that contains positive k training examples and n total training examples (Zadrozny and Elkan, 2001). Its capability of representing the real class probability is criticized in some literature, due to high bias and high variance (Zadrozny and Elkan, 2001). Some also introduced some possible changes in order to improve it (Liang and Yan, 2006) (Provost and Domingos, 2003).

Random Forests use a probability forest as in (Malley et al., 2012). In short, each tree predicts class probabilities and these probabilities are averaged for the forest prediction.

Some probability based models already return class probabilities by default, such as Naive Bayes.

2.6 Evaluation Metrics

Different metrics can be used to evaluate the performance of a ranking model.

Spearman's rank correlation coefficient (ρ) (Spearman, 1904) is a non-parametric measure of correlation between two variables. For a pair of rankings π and π' of length k, it is defined as

$$\rho = 1 - \frac{6D(\pi,\pi')}{k(k^2-1)}, \tag{2.3}$$

where

$$D(\pi,\pi') = \sum_{i=1}^{k} (\pi(i) - \pi'(i))^2 \tag{2.4}$$

is the sum of squared rank distances (Vembu and Gärtner, 2010).

Kendall tau correlation coefficient (τ) (Kendall, 1938) is a non-parametric statistic used to measure the degree of correspondence between two rankings. For a pair of rankings π and π', it is defined as

$$\tau = \frac{n_c - n_d}{\frac{1}{2}k(k-1)}, \tag{2.5}$$

where n_c is the number of concordant pairs, and n_d is the number of discordant pairs in π and π' (Vembu and Gärtner, 2010).

These metrics evaluate the quality of the predicted ranking in relation to a ground truth. Although we assume total orders, it may be the case that two labels are tied in the same rank. In this case, a variation of Kendall's τ, the tau - b (Agresti, 2010) can be used (de Sá et al., 2017).

Another interesting metric is the Normalized Discounted Cumulative Gain (NDCG) (Järvelin and Kekäläinen, 2000). For a given query q_i, the results are sorted by decreasing score output by the algorithm, and the NDCG is then computed as

$$NDCG@k \equiv N_i \sum_{j=1}^{k} \frac{rel(j)}{log_2(1+j)}, \tag{2.6}$$

where $rel(j)$ is the relevance of the j'th document, and where the normalization constant N_i is chosen so that a perfect ordering gets NDCG score 1 (Burges et al., 2005). Also, k is the cutoff point or the size of the document list, if smaller than the cutoff.

This formulation can be divided as

$$NDCG@k \equiv \frac{DCG@k}{IDCG@k}, \tag{2.7}$$

where

$$DCG@k \equiv \sum_{j=1}^{k} \frac{rel(j)}{log_2(1+j)}, \tag{2.8}$$

and

$$IDCG@k \equiv \sum_{j=1}^{k} \frac{rel_i(j)}{log_2(1+j)}, \tag{2.9}$$

where $IDCG@k$ is the ideal DCG. As such $rel_i(j)$ is the ideal relevance at position j. $IDCG@k$ is also used for normalization purposes.

There is also a tie aware NDCG approach. NDCG can be adapted fairly easily to deal with ties, as the normalization requires no special attention, and discounted cumulative gain is a simple sum over the returned results. For each position in a tied group, the average gain at that position is the average of the gain function across tied elements. As the discount function is multiplicative, we need only multiply it by this average gain at each position (McSherry and Najork, 2008).

As such, the DCG@k can then be formulated as

$$DCG@k \equiv \sum_{j=1}^{k} \frac{rel_{avg}(j)}{log_2(1+j)}, \tag{2.10}$$

where

$$rel_{avg}(j) \equiv \frac{1}{n} \sum_{t=1}^{n} rel_t(j), \tag{2.11}$$

where $rel_j(t)$ is the relevance of the t element at rank j. n is the number of tied elements at rank j.

NDCG values the document/label at the beginning of the sequence more than the document/label at the end, and that can be useful in some situations. NDCG is common in learning to rank problems.

2.7 Closely related works

Multi-Directional Ensembles of Regression and Classification Trees (Van Wolputte et al., 2018), in short MERCS, achieves, in essence, what this work intends to achieve, but for classification and regression. It intends to solve the following problem: given a dataset D with instances described by a set of variables $A = A1, \ldots, Am$, learn a model M such that, for any subsets $X \subseteq A$ and $Y \subseteq A$, M can be used to predict Y from X (Van Wolputte et al., 2018). They call this type of model a versatile model.

The referred work's approach is to construct an ensemble of trees of which the input and output variables vary, to the extent that the ensemble contains enough trees to allow for predictions of any Y from any X. Additionally, the benefits of decision trees carry over to the MERCS model, ensuring efficient learning and fast predictions (Van Wolputte et al., 2018).

A ranking approach of the enunciated work could prove a good solution to our problem.

2.8 Conclusions

This chapter summarizes the knowledge needed for our work. In order to understand LR we first need to understand ML and PL, as most LR algorithms are based on other ML algorithms. We have also distinguished the LR task from similar tasks and defined the State of the Art on all the stages of a typical supervised learning problem. We also explored one other possible ranking approach in the form of generating the ranks through the class probability estimation. After that, we identified possible metrics to evaluate our models. Lastly, a description of a related work, that could be adapted to ours, was done.

Chapter 3

Problem Definition

This chapter will be focused on specifying and carefully detailing this work's problem and the proposed solutions. We will start by showing exactly what type of forms are being dealt with, followed by stating what type of assumptions this work has. After that, this chapter will be centered around this work's approach to the problem, for each stage of a ML task.

This chapter presents our ranking approaches to solve the form suggestion problem.

3.1 Problem Statement

3.1.1 Forms we are dealing with

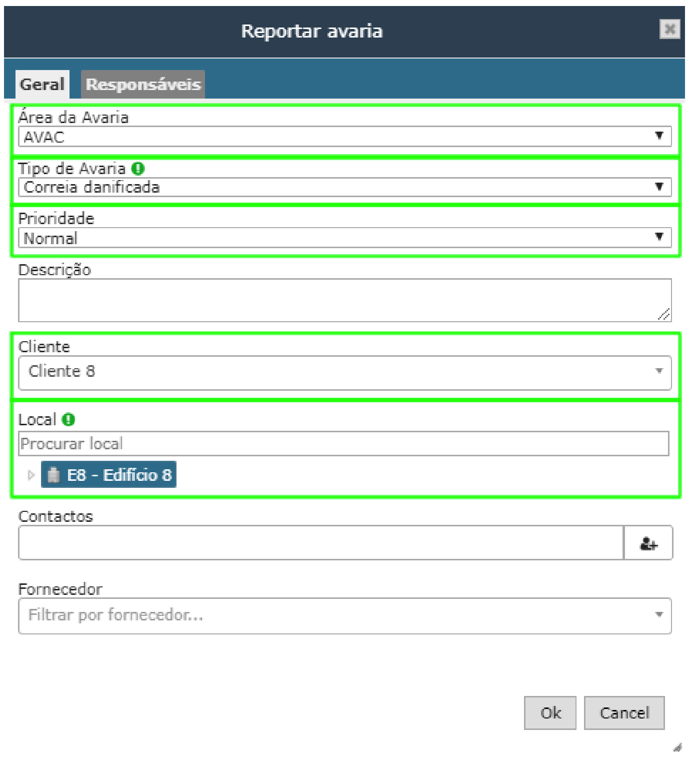
As stated before, this work deals with very specific forms. Enterprise/Management software forms have the characteristic of being used by the same user multiple times for different or the same goals. Contrary to registration forms that only are filled once for each user, the repetitive nature of the former forms allows us to assume that unfilled fields in the filling process can be predicted, based on past user usage.

These forms are composed of a multitude of input types, like textboxes, dropdowns, checkboxes, radio buttons, etc. An example of one of those forms can be seen in figure 3.1.

The green boxes in figure 3.1 are the fields to be suggested.

3.1.2 Assumptions

A couple of assumptions were made to develop this work. Firstly, it is assumed that the same form filled by the same user multiple times follows a certain pattern. If each form is filled in a "random" way this ranking approach makes no sense, since no relevant knowledge can be obtained from past user's data. It is also assumed that there is some degree of dependence between some fields in a form, otherwise using the filled fields as features in a ranking model would have no impact or a negative one. Lastly, it is also assumed that the ranking extracted from past forms is the ground truth (real ranking), in order to train and evaluate the models.


Reportar avaria

Geral | Responsáveis

Área da Avaria	AVAC
Tipo de Avaria	Correia danificada
Prioridade	Normal
Descrição	
Cliente	Cliente 8
Local	Procurar local; E8 - Edifício 8
Contactos	
Fornecedor	Filtrar por fornecedor...

Figure 3.1: Form example (Infraspeak's platform)

3.1.3 Suggestion System requirements

The proposed system has some requirements that need to be taken into account. The system has to:

1. Be user based;
2. Use past forms and currently inputted data for the ranking process;
3. Adapt to the user filling a field;
4. Rank possible entries for each unfilled field;
5. Be able to make predictions in "real time".

Requirement 1 states that the ranking system is independent between users of the software. **Users in this problem are entity (company) and operator combinations.** It makes no sense to use data from company X to rank possible entries for company Y, because they have completely different working contexts. For example, a retail company has completely different filling processes of a certain form compared to a car selling company, even though they can still use the same forms and the same software to manage their business.

Requirement 2 was already stated before. This work intends to extract knowledge from each user's past filled forms to train models to rank a new entry. Each of those models is trained using different field combination in order to allow the currently inputted data to be used for the ranking process.

Requirement 3 means that the system, as soon as a new field is filled, has to adapt the ranking of each unfilled field by making a new prediction. This leads to the necessity to generate multiple models to adapt to each possibility. This will be discussed in further sections.

Requirement 4 is the main goal of the system. We want to rank each possible entry for each unfilled field in order to make a suggestion. To make the suggestion we don't need a full ranking, as not all values need to be used in the suggestion. That will also be discussed in the next sections.

Requirement 5 is the main reason time measurements are being considered. In order to be viable in a real environment, the time it takes to make a ranking prediction must be apparently instant.

3.1.4 Understanding a suggestion system

The purpose of this work is to evaluate the possibility of using ranking algorithms to suggest each possible value in a certain field and extend that solution to all unfilled fields in a form, which leads to a suggestion system.

Although we are only interested in the top values of that ranking, the complete ranking is obtained in order to give whoever implements the visual interface of the suggestion system full liberty. This work only intends to offer the tools and not decide how to use them.

However, it is logical that, for usage purposes, not more than five suggested values will be shown at any given time, usually three even. The usage of too many values can increase the complexity of the filling process, when the main goal is the opposite. This means that, in order to evaluate the quality of the ranking in a realistic way, the selection of those quality metrics is relevant, as it will be discussed in further sections.

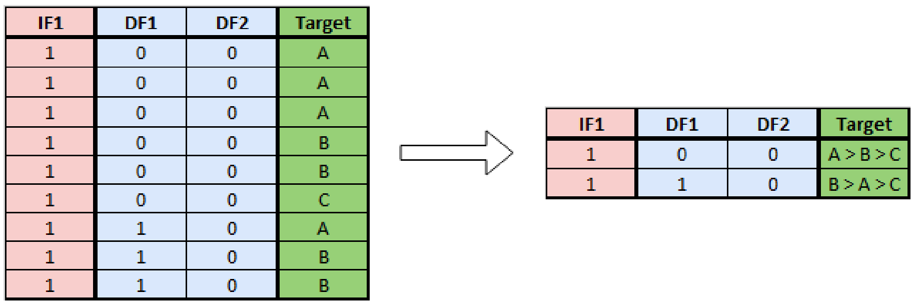
3.2 Proposed solution

3.2.1 Data preparation for model training and validation

As stated in section 2.4.2 there are two main approaches for obtaining a ranked set. This ranked set will be used in RPC's training and also in the ranking validation.

We will take the implicit approach, which means that the ranking information for training will be obtained from the dataset. For this specific problem, in the dataset we only have information on past form entries. This means that we need to find a way to obtain rankings from that information. The proposed solution for that revolves around aggregating instances in the dataset that have the same features and using the count of the target label to order each possible entry for that target. An example of this transformation can be seen in figure 3.2.

We will assume that the ranking obtained from this transformation is the ground truth for model validation purposes.



IF1	DF1	DF2	Target
1	0	0	A
1	0	0	A
1	0	0	A
1	0	0	B
1	0	0	B
1	0	0	C
1	1	0	A
1	1	0	B
1	1	0	B

IF1	DF1	DF2	Target
1	0	0	A > B > C
1	1	0	B > A > C

Figure 3.2: Example of the dataset transformation to obtain an ordered set/rank

Apart from this data transformation we will also be decomposing the date values into three variables, quarter of the year, day of the week and if its weekend or not. This is a simple decomposition, and that is intended. If the date was decomposed into more specific variables, we would end up with a very sparse dataset, which would make the aggregation mentioned in the beginning of this section end up always with a single instance aggregated, and by consequence, a label with rank one and all the others tied with 0 relevance. That is still a normal occurrence in the dataset but we also want to rank more complete ranking data, so we need to control that dataset sparsity by also controlling the features that we are aggregating it by. This also reduces the possibility of overfitting the models.

3.2.2 Model's Features

Features have to be defined in order to train and test the ranking models. This feature selection is crucial to any ML task. Two types of feature sets were defined. They were classified as:

- Direct Features (DFS) – Currently filled fields.
- Indirect Features – Context features.

Direct Features are the obvious features of the system, the currently filled values. These allow the system to adapt to the user interaction with the software, while extracting valuable knowledge to the ranking process. It is important to note that these features will all be treated as nominal or categorical values, even though they can sometimes be plain text. This means that, for the system, a user filling a certain text field as "Porto" is different from filling it as "Oporto". The green boxes in figure 3.1 are an example of these types of features. These features also have the characteristic of being in some cases predictive features and in other cases the target variable we want to suggest.

Indirect Features are context features, for example time or date. Essentially any feature that can be extracted from the dataset, that does not belong to the form fields and that is relevant to the ranking. These features also allow us to control the sparsity of the dataset. For instance, if the dataset is too simple or too bias towards a certain feature set, we can change the granularity of certain indirect features to solve it. A simple example of that is using the day of the week instead of the day of the month.

3.2.3 Multiple models solution

As it was already mentioned before, the suggestion system is based on multiple ranking models, due to the necessity of having to adapt to the other filling a field and to the fact that multiple fields are being predicted for each form. The adaptation of the user filling a field leads to variables that were being ranked before becoming part of the DFS set. A simple example of that can be seen in figure 3.3.

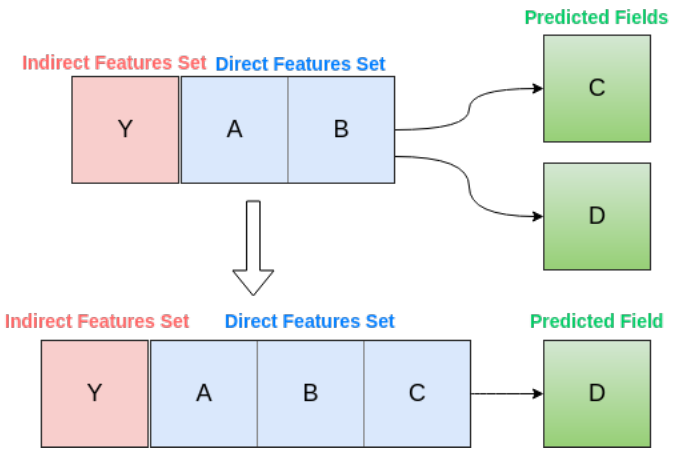


Figure 3.3: Example of a variable passing from the predicted set to the DFS set

This fact leads to the formulation of multiple models, and the amount of models to be generated depends on the number of fields each form has. This can be calculated as

$$2^{n-1}n, \tag{3.1}$$

where n is the number of fields in the form. This formula essentially is the number of values in subsets that can be achieved using a superset, in this case the DFS set minus each target variable, times the number of target variables. With all these models we are able to predict and suggest any unfilled field for any possible form filling state.

There is the threat of the number of models scaling, due to the exponential growth in relation to the number of fields, to an impracticable amount, due to high training time. As such, the amount of fields in a form this system can deal with has to be limited. For example, the form shown in the figure 3.1 leads to $2^{5-1} * 5 = 80$ needed models for each User, 16 for each target variable. An example of the needed models can be seen in the table 3.1.

This assumes that every Direct Feature is relevant for each target variable. Another possible approach to reduce this issue, that this work will attempt, is to identify, prior to training the models for a certain target variable, which DFS are relevant. Our hypothesis is that, it is possible to do it by training a model, for a certain target variable, with all the other DFS. From that model, we can then extract each feature's importance and identify which are irrelevant. If a feature is irrelevant,

Target	Feature Combinations
A	{}, {B},{C},{D},{E},{B,C},{B,D},{B,E},{C,D},{C,E},{D,E},{B,C,D},{B,C,E},{B,D,E}, {C,D,E}, {B,C,D,E}
B	{}, {A},{C},{D},{E},{A,C},{A,D},{A,E},{C,D},{C,E},{D,E},{A,C,D},{A,C,E},{A,D,E}, {C,D,E}, {A,C,D,E}
C	{}, {A},{B},{D},{E},{A,B},{A,D},{A,E},{B,D},{B,E},{D,E},{A,B,D},{A,B,E},{A,D,E}, {B,D,E}, {A,B,D,E}
D	{}, {A},{B},{C},{E},{A,B},{A,C},{A,E},{B,C},{B,E},{C,E},{A,B,C},{A,B,E},{A,C,E}, {B,C,E}, {A,B,C,E}
E	{}, {A},{B},{C},{D},{A,B},{A,C},{A,D},{B,C},{B,D},{C,D},{A,B,C},{A,B,D},{A,C,D}, {B,C,D}, {A,B,C,D}

Table 3.1: Feature combinations needed for the DFS set $\{A, B, C, D, E\}$

for a certain target, models trained with and without it are approximately the same. That means that it is not needed to train all models, only the ones without it. Even if that field is filled, we can use the most adequate model for that filling state while ignoring that Direct Feature. For example, from the 16 calculated models for a certain target, if one Direct Feature is considered irrelevant we would end up with only $2^{4-1} = 8$ models. An example of this model pruning method can be seen in table 3.2.

Generated models for Target A	It also works for
{}	{E}
{B}	{B,E}
{C}	{C,E}
{D}	{D,E}
{B,C}	{B,C,E}
{B,D}	{B,D,E}
{C,D}	{C,D,E}
{B,C,D}	{B,C,D,E}

Table 3.2: Example of generated models for target A with the irrelevant feature E

In essence, we generate all possible combinations of the considered relevant features, and, apart from working for a that feature combination, it should also be used for all subsets of irrelevant features, including the superset, merged with the original feature combination.

This approach will be compared to the more brute force approach of generating models for each DFS combination.

3.2.4 Dataset Analysis

As stated before we will be working with a dataset provided by Infraspeak. This dataset is the result of entries in a failures form, figure 3.1. A sample of that dataset can be seen in table 3.3.

entity_id	operator_id	start_date	problem_parent_id	problem_id	local_id	client_id	priority
24	1136	2018-05-02 20:36:20+00	486	2269	90579	3155	3
12	826	2018-04-11 23:56:57+00	151	739	55099	5507	2
66	836	2018-04-21 16:10:44+00	2962	3015	66720	6016	2
87	2511	2018-07-03 12:40:27+00	5580	5583	141521	6389	3
24	536	2018-05-13 11:09:07+00	462	463	34866	3155	3

Table 3.3: Example of failures dataset in its original state

Most of the columns are categorical variables. They refer to:

- **entity_id** – Id of the entity responsible for reporting the failure, in essence the company using Infraspeak's platform – *Categorical Variable*.
- **operator_id** – Id of the operator (platform user) reporting the failure – *Categorical Variable*.
- **start_date** – Time and date of the failure report – *Continuous Variable*.
- **problem_parent_id** – Id of the area in which the problem can be included – *Categorical Variable*.
- **problem_id** – Id of the problem – *Categorical Variable*.
- **local_id** – Id of the local in which the failure took place – *Categorical Variable*.
- **client_id** – Id of a client of the entity, owner of the failing item – *Categorical Variable*.
- **priority** – Priority of the reported failure, from 1 to 4, where 4 is the highest priority and 1 the lowest – *Discrete variable*.

As stated in section 3.1.3, users in our approach are entity_id and operator_id combinations. The complete dataset has **363339** instances divided by **1571** entity/operator combinations.

The other variables, although mostly categorical, vary in the numbers of possible classes in different ways, depending of the entity/operator combination. A certain id only matters for an entity/operator combination if it shows at in a instance within that combination, otherwise it does not exist within the scope of that combination. As such, and assuming that the number of possible classes to rank has an effect in both quality metric and time metrics, an evaluation of it is needed, in order to understand how they vary. An analysis of the distribution of the number of classes for each variable, grouped by entity/operator combinations, can be seen in figure 3.4.

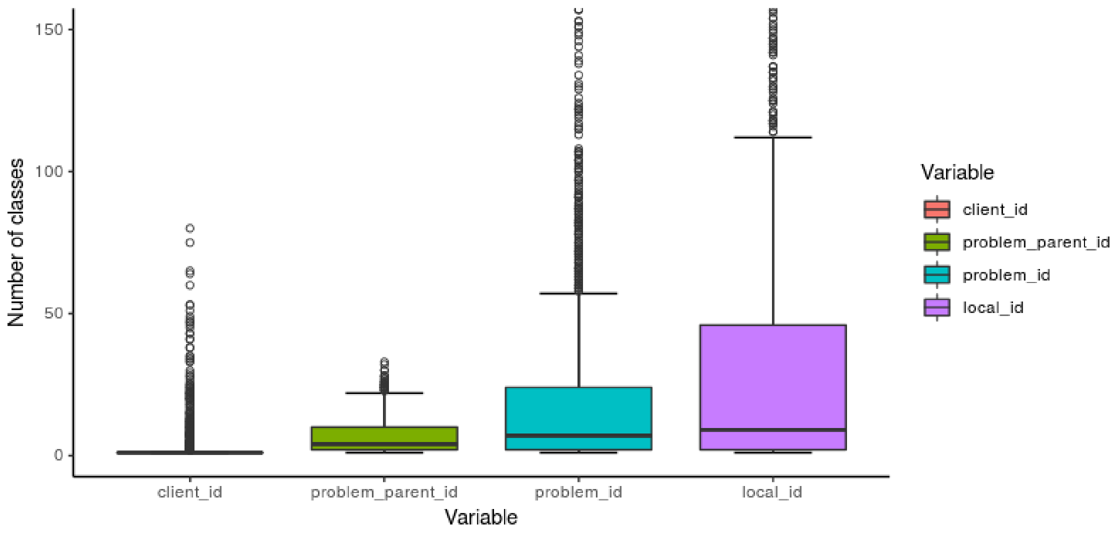


Figure 3.4: Number of classes by entity/operator distribution

As it can be seen, client_id, problem_id and local_id have a high amount of outliers. It can also be added that the maximum possible value of the number of classes of each is 310, 280 and 1105 respectively. From a brief analysis of the plot in figure 3.4, we can observe that the variable that usually has the higher number of classes or possibilities is local_id, while also having the higher possible amount at 1105. This identifies the number of possible local_ids as a relevant value for the ML process. Apart from that value, we also know for a fact that the number of instances in each entity/operator combination is also an important factor in both quality and training time. While the higher amount of instances usually correlates to more knowledge, it should also lead to an increase of the training time. On average, the increase in the number of instances leads to an increase in the number of possible local_id classes, but that is not always true as we can see in figure 3.5.

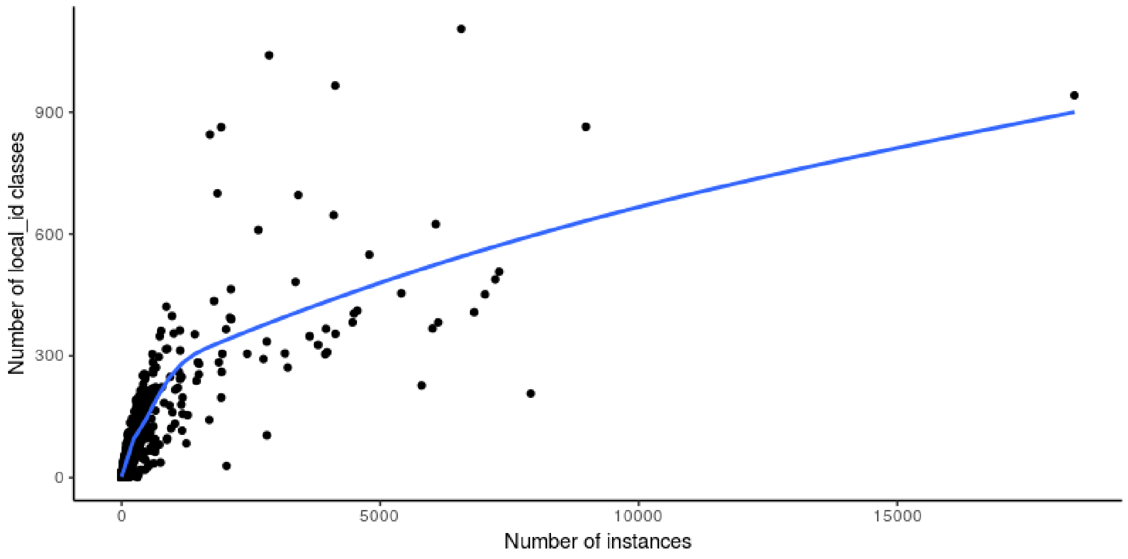


Figure 3.5: Number of instances vs the number of local_id classes

It should be noted that client_id is a fringe case within this dataset. Since it refers in theory to a client of a client, from Infraspeak's perspective, in reality it usually is a different id for its own entity. That leads to, on average, having only one client_id within an entity/operator scope, although there are some cases where it is higher than that, representing companies that really offer a maintenance service to others.

Other than this evaluation, it is hard to make a precise analysis of each variable, although they are always the same for each entity_id/operator_id combination, their usage varies wildly.

3.2.5 Ranking Approaches and Algorithms

Three ranking approaches will be tested in this work, RPC (2.4.3.1), ranking by probability estimation (2.5), and the later approach with pruned models (3.2.3).

In regards to the RPC implementation, the pairwise comparison models are trained with rpart trees. Also, for the pairwise comparison, only instances where knowledge of a preference between one of the values in the pair in relation to the other is known, otherwise, that instance is removed. This means that if the pair is tied in a certain instance, that instance is not used to train that pair's model.

Ranking by probability estimation (RPE) requires data in a labeled format, not in a ranking one. This means the data used for RPE will be close to the original dataset. It will still be tested with ranked instances, for NDCG purposes. In order to test different probability estimation methods from different algorithms, RPE's method, both the pruned models and brute force approach, will be tested with C5.0 trees, Random Forest and the Naive Bayes classification techniques. SVM and xgBoost were also considered, during development, but, due to the need of specific parameter tuning from problem to problem and due to high training times their usage possibility was discarded.

The experiments and comparisons between each approach will be further explained in the later chapters.

3.2.6 Model evaluation/validation

In order to evaluate the ranking performance of each model, NDCG with ties, as defined in section 2.6, will be used. This means that not all the dataset can be used for training, as part of it will be used for validation. For that we will use 5-fold cross validation strategy.

NDCG measures the relevance of the ranked labels according to their rank divided by the ideal rank, for normalization purposes. As stated before, it gives more importance to the ranks at the top. It is also possible to use a cutoff point k, that means that ranks higher thank k are not evaluated. We will be using a cutoff of 5, mostly because it is assumed that we will never suggest more than 5 values, section 3.1.4. Since NDCG uses the relevance of each label, we also need to classify the dataset labels, for any given instance, in terms of relevance. Relevance will be the times a certain label showed up in a given instance, using the aggregation method described in section 3.2.1. This means if it shows zero times it is considered an irrelevant label for NDCG purposes and the more times it appears the more relevant it is.

This metric will give us a good idea of the quality of the obtained ranking.

It should be noted that, there are other possible gain functions that value even more the relevant values. Due to the fact that we are using an aggregation count, that can achieve already high values, we felt that it already differentiates and values relevant from irrelevant values, so there no need for an even more punishing gain function.

3.2.7 Other metrics

Apart from a pure ranking quality metric described in section 3.2.6, two performance metrics will be evaluated.

Training time for each approach and algorithm will be measured in order to evaluate how long it takes, for any given form, to train. This is relevant because in order to keep the models updated, a certain training frequency is assumed, for example each day. Since the training process is long and complex, due to all combinations needed to train for multiple Users, keeping the training time of each model low, and by consequence keeping the complete form training time low, is important.

Prediction time will also be taken into consideration, since this is a real time system. In real time, multiple predictions, one for each unfilled field, will be made in order to suggest possible entries. As such, the smaller it is, the faster will the system be able to give its suggestions. There is also a limit to which the prediction time cannot pass, because at that point it loses its real time characteristic.

3.3 Conclusions

Defining the form suggestion problem as a ranking problem has many facets. First what type of forms are we dealing with. It is important to note the context of these forms and what a user is from the perspective of the system, a company/operator combination. From this point on, every time a certain feature is referenced is always within this scope.

The feature separation into two different types allows a much needed flexibility to solve some typical ML problems, such as bias datasets or lack of information, while allowing the multiple model system to adapt to the user filling a form. While the most obvious approach is to model every possible filling state to solve these issues, we also presented the possibility of avoiding to train redundant models by identifying the relevant DFS.

Data preparation is also an important stage in our work, as it is from it that we can get the rankings needed to train with the RPC algorithm and also to test with NDCG. NDCG will be the main metric used to evaluate the quality of the ranking models, mainly because it values the top ranked labels, that are the most important for the suggestion system.

Understanding the dataset allows us to take some assumptions. In this case we assumed that the number of local_id classes and also the number of instances, within a company/operator scope, are distinctive characteristics, that will influence directly, at least, the total training time. On average, the higher the number of possible classes any given target has, the lower NDCG will become.

Chapter 4

Experiments

In this chapter we will be detailing and explaining the three experiments done. In summary, we will be doing an experiment in order to compare the RPE against the RPC label ranking algorithm. Experiment 2 and 3 will be based around testing RPE, with three algorithms, C5.0, Random Forest and Naive Bayes, with the brute force approach, testing every combination of DFS, and with the feature pruning by importance approach, respectively. In all experiments we will be comparing NDCG, training time and testing time.

Some sampling strategies were used due to the sheer size of the dataset. It should be noted that, for sampling purposes we removed all entity/operator combinations that did have less than 100 instances, in all experiments, as we considered that as an arbitrary number to consider that, with less than 100 we did not have enough knowledge. This number is merely arbitrary, and was chosen mainly because we had 420 observations higher than that.

4.1 Experimental Setup

As stated before, we will use different sampling strategies, depending on the experiment, to select which entity_id/operator_id combinations will be used. The optimal way would be to use the whole dataset, but due to time constraints that is impossible.

The basic experimental setup for each entity_id/operator_id combination can be seen in figure 4.1. In summary, for each sampled entity_id/operator_id combination, a validation process (1) is applied.

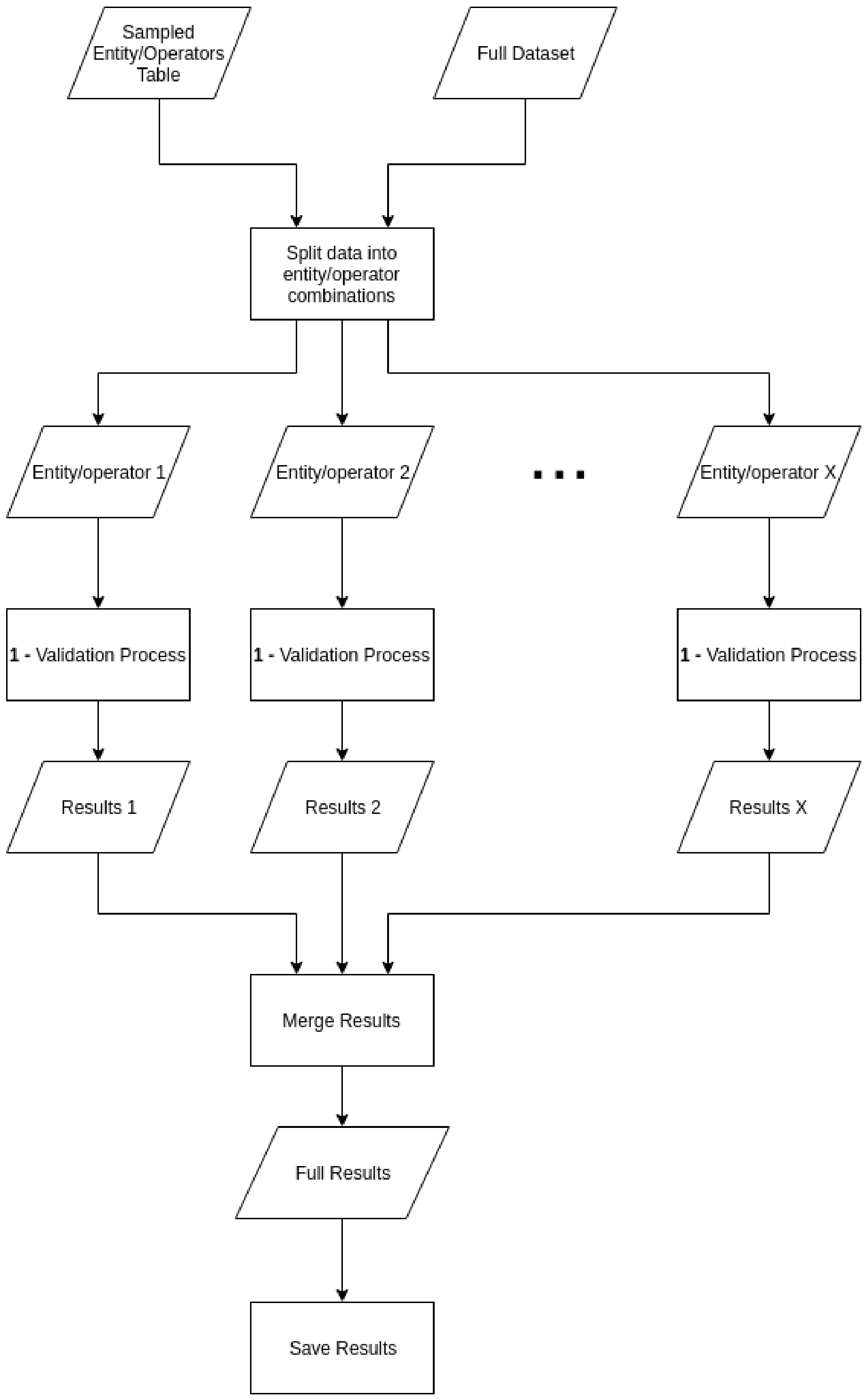


Figure 4.1: Complete form validation and results

That validation process, see figure 4.2, takes the sampled dataset and a list of DFS. Then, for each target (each element in the DFS list), generates the ranking sets (1), as explained in section 3.2.3, and a list of all possible combinations of the rest of the DFS (2). The result of the first operation is then used to generate the 5-fold cross-validation folds. The later operation uses bitwise operations and lexicographic ordering to generate all possible subsets of the DFS minus the target set. This includes the empty set and the complete set.

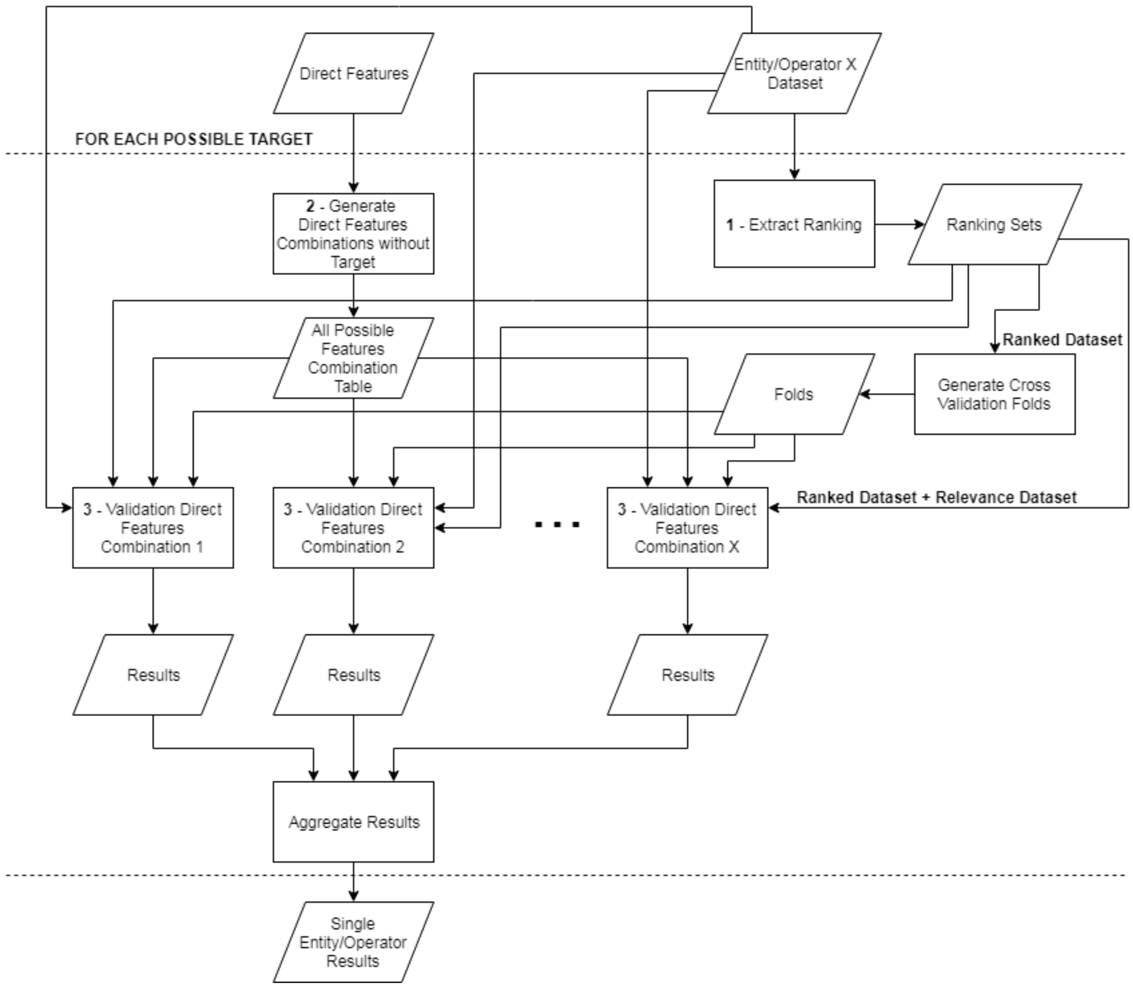

Direct Features
Entity/Operator X Dataset
FOR EACH POSSIBLE TARGET
2 - Generate Direct Features Combinations without Target
1 - Extract Ranking
Ranking Sets
All Possible Features Combination Table
Ranked Dataset
Folds
Generate Cross Validation Folds
3 - Validation Direct Features Combination 1
3 - Validation Direct Features Combination 2
3 - Validation Direct Features Combination X
Ranked Dataset + Relevance Dataset
Results
Results
Results
Aggregate Results
Single Entity/Operator Results

Figure 4.2: Validation Process for each entity_id/operator_id

In the pruned models approach, before generating all possible combinations of DFS (2), a C5.0 model is trained with all DFS. C5.0 importance values are in the range 0 to 100. We then, consider as relevant all DFS that have an importance higher than 10, using only those to generate the possible combinations. Then, after each model is trained, it is also assigned to that DFS combination associated with all possible subsets of irrelevant features.

We used 5-fold cross-validation because it felt as the perfect balance between computation time and results credibility. This cross-validation is generated through a stratified strategy, in which, it is considered the label that is on top of the rank for a certain instance, the distinctive attribute for that same instance, for stratified cross-validation purposes only.

It is important to note that the result of Extract Ranking (1) is, in reality, two sets. One with the labels identified by ranking, see table 4.1, with 1 being the top rank, and another dataset with the relevance for each label, see table 4.2, which is, as stated before, the number of times that label appears in that specific instance. The last is used only for NDCG purposes.

client_id	local_id	problem_id	problem_parent_id	SeasonYear	WeekDay	WeekEnd	1	2	3	4
3155	90474	543	541	4	Qua	0	3	3	1	3
3155	90474	543	541	4	Qui	0	3.5	2	1	3.5
3155	90474	2269	486	1	Qua	0	3	1	3	3

Table 4.1: Example of the ranked dataset for target priority

client_id	local_id	problem_id	problem_parent_id	SeasonYear	WeekDay	WeekEnd	1	2	3	4
3155	90474	543	541	4	Qua	0	0	0	1	0
3155	90474	543	541	4	Qui	0	0	1	2	0
3155	90474	2269	486	1	Qua	0	0	1	0	0

Table 4.2: Example of the relevance dataset for target priority

Then, the ranked set, original set and the generated folds are used into a single DFS combination validation process (3).

A pseudocode representation of the whole process can be seen in algorithm 1. The original dataset refers to the selected entity_id/operator_id combination.

```
input  : $directFeaturesSet, originalDataset = entity\_id\_operator\_idX\_Dataset$
output: $resultsSet$
foreach $target \in directFeaturesSet$ do
    $directFeaturesCombSet \leftarrow$ generateDFSComb ($directFeaturesSet$ - $target$);
    $rankingSets \leftarrow$ extractRanking ($originalDataset, target$);
    $rankedDataset \leftarrow rankingSets[0]$;
    $relevanceDataset \leftarrow rankingSets[1]$;
    $foldsSet \leftarrow$ getCrossValidationFolds ($rankedDataset$);
    foreach $dfs \in directFeaturesCombSet$ do
        $results \leftarrow$ validate ($originalDataset, rankedDataset, relevanceDataset,$
          $foldsSet, dfs$);
        $resultsSet \leftarrow resultsSet + results$;
    end
end
```

Algorithm 1: Validation Process for each entity_id/operator_id

We can observe a sketch of the single combination validation process in figure 4.3. In this process, first a fold is obtained from the ranked set (1), generating a ranked training set. The same process is repeated, in the relevance set, to obtain the test instances of that same fold, generating a test set. Afterwards, from the original dataset, we extract all instances that correspond to the ranked training set (2), for the RPE approach, in order to guarantee the integrity of the results, by using the same instances in both RPE and RPC. We then fit the data into 2 types of models, RPC (4) and RPE (5). For the RPC approach, multiple models are trained, one for each possible pair. These models are generated through rpart trees. For RPE, for each algorithm being tested, only one model is trained. In both approaches, we apply the models to the test set, and, using the relevance knowledge obtained in the transformation of the original set into the ranked set, compare the predicted ranking with the ideal ranking, using NDCG.

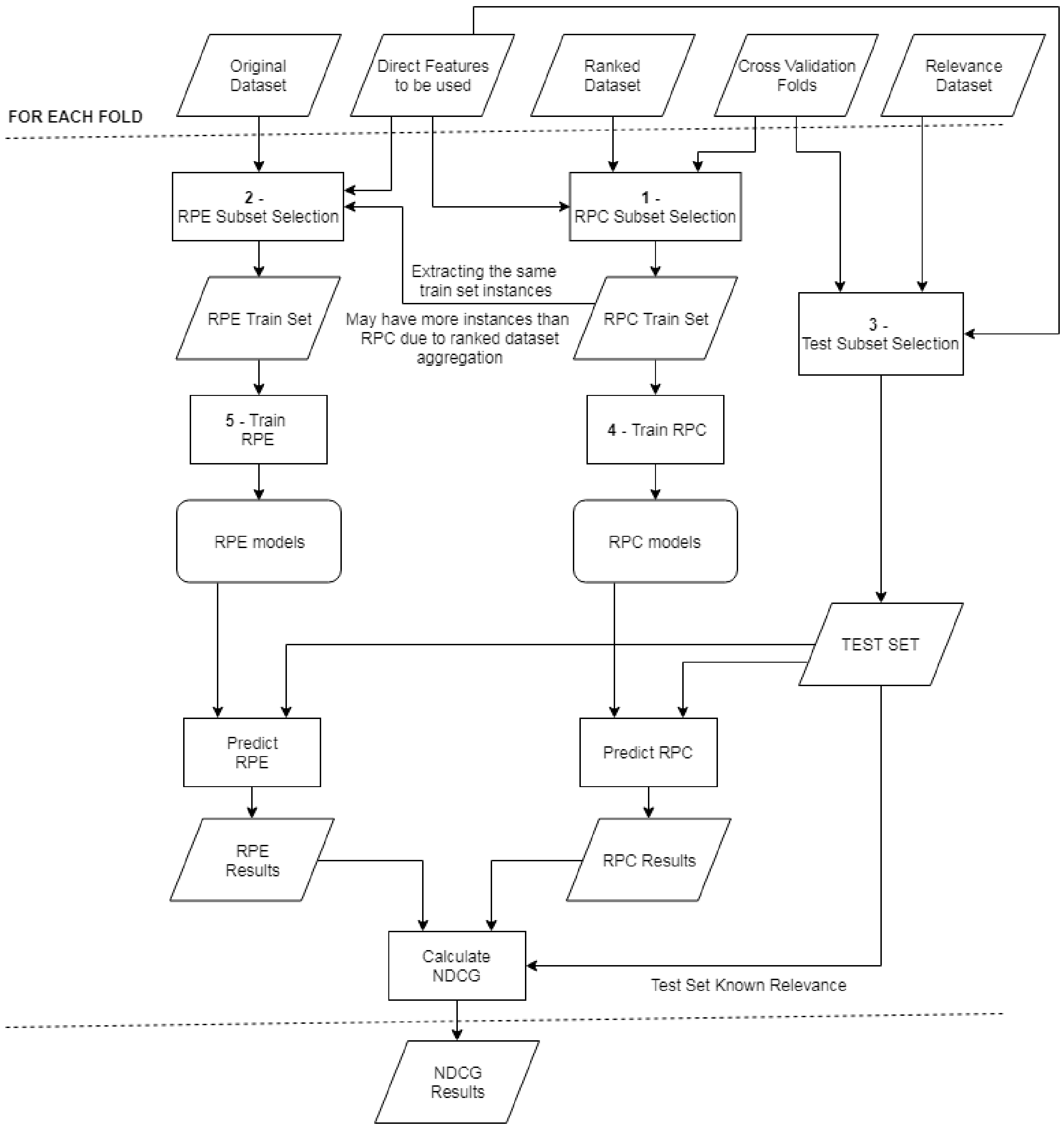


Figure 4.3: Single Direct Features combination Validation Process

When validating only the RPE approach, which will happen in experiment 2 and 3, the RPC Train Set is still produced and used to obtain the RPE train instances, because it is necessary to guarantee consistency between the training set and the test set. Mainly, because the test set has to be a ranked set too. All RPC operations after that are then ignored for this case.

A pseudocode representation of the last validation process can be seen in algorithm 2.

```
input  : $originalDataset, rankedDataset, relevanceDataset, foldsSet, dfs$
output: $results$
foreach $fold \in foldsSet$ do
    $trainSetRPC \leftarrow$ selectTrainFolds ($rankedDataset, fold$);
    $trainSetRPE \leftarrow$ extractInstances ($originalDataset, trainSetRPC$);
    $testSet \leftarrow$ selectTestFolds ($relevanceDataset, fold$);
    $trainSetRPC \leftarrow$ selectFeatures ($trainSetRPC, dfs$);
    $trainSetRPE \leftarrow$ selectFeatures ($trainSetRPE, dfs$);
    $testSet \leftarrow$ selectFeatures ($testSet, dfs$);
    if $doRPC == TRUE$ then
        $modelRPC \leftarrow$ trainRPC ($trainSetRPC$);
        $predictionsRPC \leftarrow$ predictRPC ($testSet, modelRPC$);
        $resultsRPC \leftarrow$ NDCG ($testSet, predictions$);
        $results \leftarrow results + resultsRPC$;
    end
    $modelRPE \leftarrow$ trainRPE ($trainSetRPE$);
    $predictionsRPE \leftarrow$ predictRPE ($testSet, modelRPC$);
    $resultsRPE \leftarrow$ NDCG ($testSet, predictionsRPE$);
    $results \leftarrow results + resultsRPE$;
end
```

Algorithm 2: Single Direct Features combination Validation Process

4.2 Implementation and Hardware Details

The validation process was tested on a Intel Core i5-6200U CPU 2.30GHz, with two physical cores, and on 8 gigabytes of memory. This specification is relevant because the measured times obtained would be different on a different machine.

In regards to the implementation, it was done in R, using for most algorithms already existing packages. It is also important to note the following for each algorithm:

- RPC – Pairwise modeling used a self-made implementation, using rpart trees from the rpart package, with the default values. Rpart is pretty much parameter free.

- C5.0 – From C50 package, with 10 boosting iterations, although with early stopping if no gain is achieved. The class probability estimation reflects the distribution of the classes in the terminal nodes.

- Random Forest – From the ranger package, with 500 trees and working on two threads. The last one is relevant because it affects how we visualize the timings obtained on Random Forest. While the other implementation use only one cpu core, Random Forest is using two. The class confidence values were already explained in 2.5.1

- Naive Bayes – From the e1071 package. Naive Bayes is also parameter free.

There are some exceptions also implemented, that may occur in any given dataset. First, for any given feature, if its value is always the same, that feature is removed from the training process, as some implementations fail when that its present. Also there is no knowledge to obtain from a feature that only has one possible value. That may also happen with the target variable. If the target variable has always the same value, no model is trained, regardless of the algorithm being used. We can assume that value as the right response always.

The whole process is seeded so that the results can be reproducible.

4.3 Experiment 1 - RPE vs RPC

Experiment 1, in essence, was made in order to compare RPC and RPE. We expect the training and prediction time of RPC to grow in relation to the number of possible targets to be ranked, due to the nature of pairwise comparison. We will be detailing the sampling strategy and the results obtained, as well as, an analysis of the later.

4.3.1 Sampling Strategy

It is assumed that RPC's training and prediction time will have a quadratic growth, in relation to the number of possible target classes, due to the number of needed models having the same growth. This means that, in order to be able to collect results in a reasonable time, the sample set used cannot exceed a certain value. It is also known that the local_id variable is the one which fluctuates the most in terms of possible classes, while being the one that reaches the higher amount. As such, we obtained entity_id/operator_id combinations that follow the next rules:

- Has more than 100 instances.
- Has more than 20 local_id classes.
- Has less than 90 local_id classes.

Following those rules we then extracted 5% of the resulting dataset and ended up with 8 entity_id/operator_id combinations. A summary of that sample dataset can be seen in table 4.3, where the number shown in the id's variables is the number of classes for that combination, apart from entity_id and operator_id.

An interesting detail in this sample is that the number of client_id classes varies more, on average, than in the original dataset. Other than that, this sample is a good representation of the dataset, given the limitations forced by RPC. It is also very heterogeneous in terms of the number of instances of each combination, ranging from 134 instances to 2032. Although priority ranges from 1 to 4, there are some cases where no observations of a certain value are seen, as such that value is also not considered for training purposes.

entity_id	operator_id	NumberOfInstances	Nclient_id	Nlocal_id	Nproblem_id	Nproblem_parent_id	Npriority
12	44	556	8	33	26	9	4
20	586	314	9	71	7	7	4
24	227	211	1	39	37	15	4
24	296	161	1	45	43	13	4
24	1080	134	1	63	59	12	3
62	874	355	25	77	14	7	4
62	2281	147	12	52	9	6	3
110	1372	2032	1	29	9	8	4

Table 4.3: Sample summary for RPE vs RPC

4.3.2 Results

The complete form validation of all the sampled data took *9070 seconds* (151 minutes) while on average each entity/operator combination took *1133 seconds* (18.9 minutes).

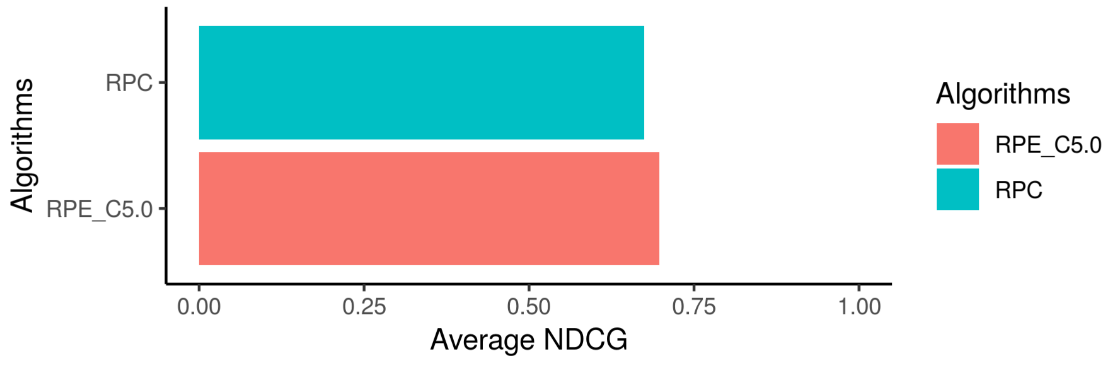


Figure 4.4: RPE vs RPC - Average NDCG

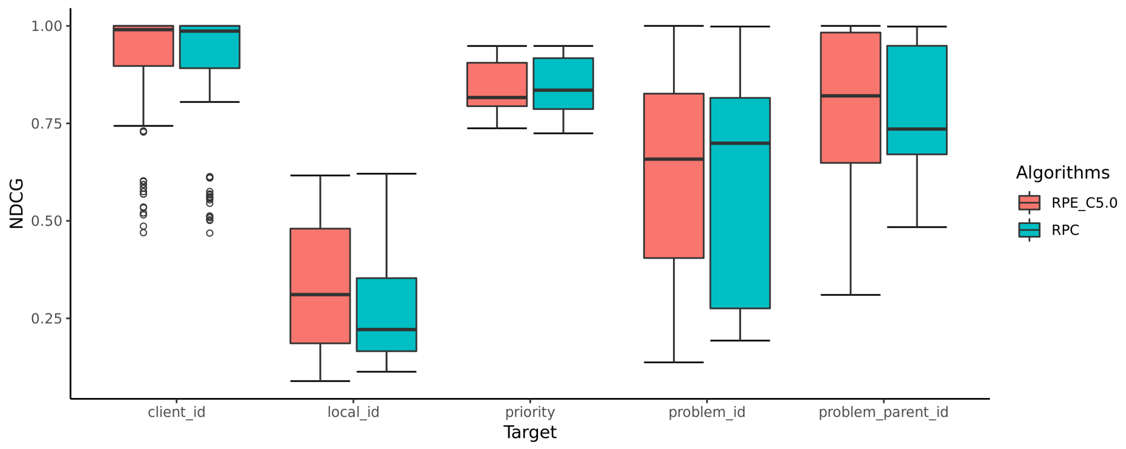


Figure 4.5: RPE vs RPC - Average NDCG for each Target

The calculated average NDCG score for RPE and RPC, see figure 4.4, was *0.697* and *0.674*, respectively. It is also possible to observe how each target behaves in regard to NDCG in figure 4.5.

We can see how NDCG behaves, for each state of the form filling process for each target variable in Appendix A. In those observations, we can also see how the existence or not of each direct feature affects the quality of the ranking.

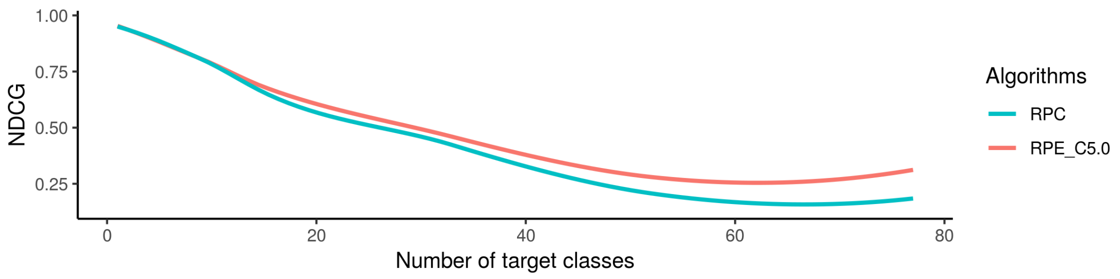


Figure 4.6: RPE vs RPC - NDCG by Number of Classes

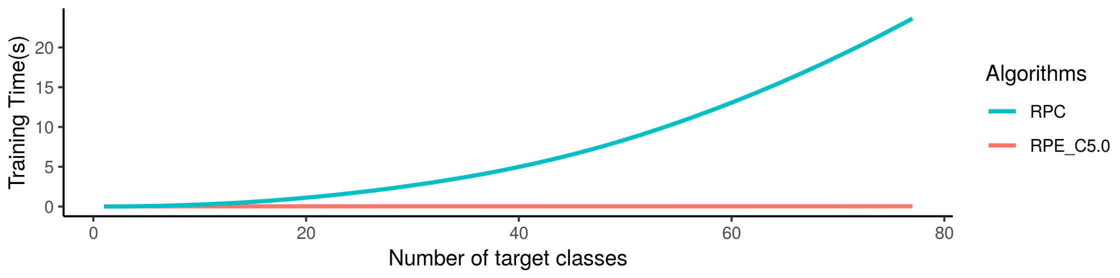


Figure 4.7: RPE vs RPC - Training Time by Number of Classes

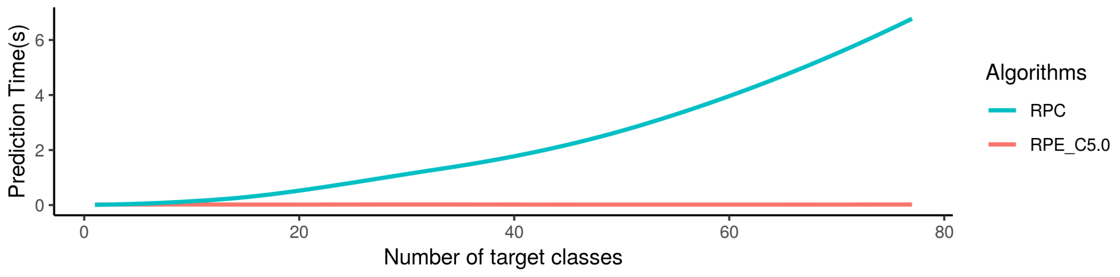


Figure 4.8: RPE vs RPC - Prediction Time by Number of Classes

Since the number of classes or possible targets was identified as a relevant value for NDCG, training time and prediction time, an analysis of how those values behave when this number of classes increases was also done. While NDCG decreases with this increase (figure 4.6), for both RPE and RPC approaches, the training time (figure 4.7) and the prediction time (figure 4.8) of RPC increases. RPC reaches, at 77 classes, *24.8 seconds* of training time and *7.6 seconds* of prediction time.

Knowing these values, it is also possible to calculate how much time it takes, for each approach, to generate all models for any given entity_id/operator_id combination, see figure 4.9. While RPE ended up with an average training time of *1.8 seconds*, for all 80 models of any combination, RPC was *227 seconds*, as can be seen by the value disparity in figure 4.9.

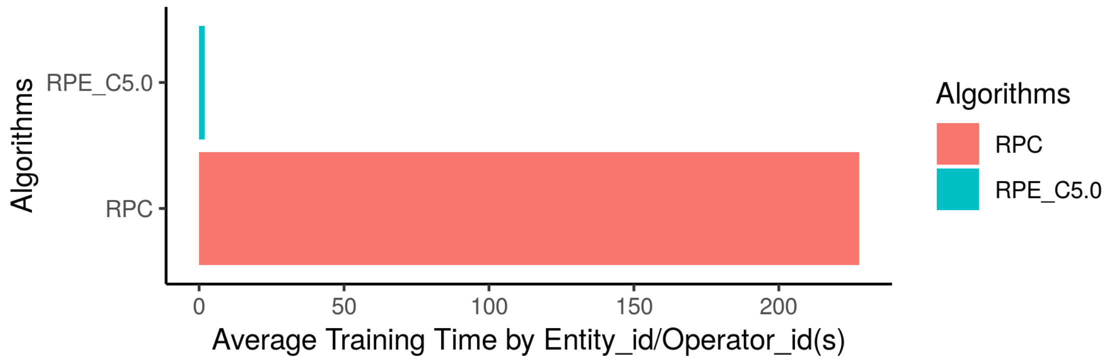


Figure 4.9: RPE vs RPC - Average Training Time for an entity_id/operator_id combination

4.3.3 Results Analysis

Results show that RPC, while competitive in terms of NDCG score, albeit slightly lower, both the high values of training time and prediction time make it infeasible for this suggestion system. As we saw before, this sample does not represent entirely the original dataset, as the later can have DFS that have way more possible classes than the sampled one. We can also see that, the measured times have a quadratic growth, in relation to the number of target classes, as expected. Training time influences how frequently these models can be updated. High values force this update frequency to be really low. On another hand, high prediction time takes the real time capabilities off the system. With all that in mind, we discarded RPC for the next experiments and focused solely on RPE.

From the plots in Appendix A we can perceive how certain features interact with a certain target. For instance, whenever problem_parent_id is present in the features set, the problem_id's NDCG increases, when it is the target variable, and vice versa. The same is also true, to some degree, between client_id and local_id.

One last thing to note, is the fact that NDCG score drops, for both RPE and RPC, with the increase of the number of possible targets, but this was also expected.

4.4 Experiment 2 - RPE multiple algorithms

Experiment 2 focuses on expanding the RPE approach to other typical classification algorithms. In this experiment, we will test how C5.0, Random Forest and Naive Bayes behave, with this approach.

These algorithms were selected for three reasons. First, they are mostly hyper parameter free, although Random Forest as some possible tuning options, specially in the number of generated trees. Second, they are generally fast at training and prediction. And lastly, because they are able to return class probability estimations.

4.4.1 Sampling Strategy

For this experiment, since the limitations of RPC are not present, a more broad sampling strategy can be applied.

We wanted to test a more precise representation of the dataset, while also testing the more extreme cases. As such, two sampling strategies were defined.

After determining that the most defining characteristics of each entity_id/operator_id combination would be the number of instances and the number of local_id classes (3.2.4), if we apply a clustering algorithm and then proceed to do a stratified sampling from it, a good representation of the original dataset should be achieved. A visual description of the resulting clusters of a kmeans algorithm applied to the number of instances and the number of local_id classes can be seen in figure 4.10.

After that we did a stratified sampling, by kmeans clusters, of 5% of the original dataset. A summary of the result entity_id/operator_id combinations is in table 4.4.

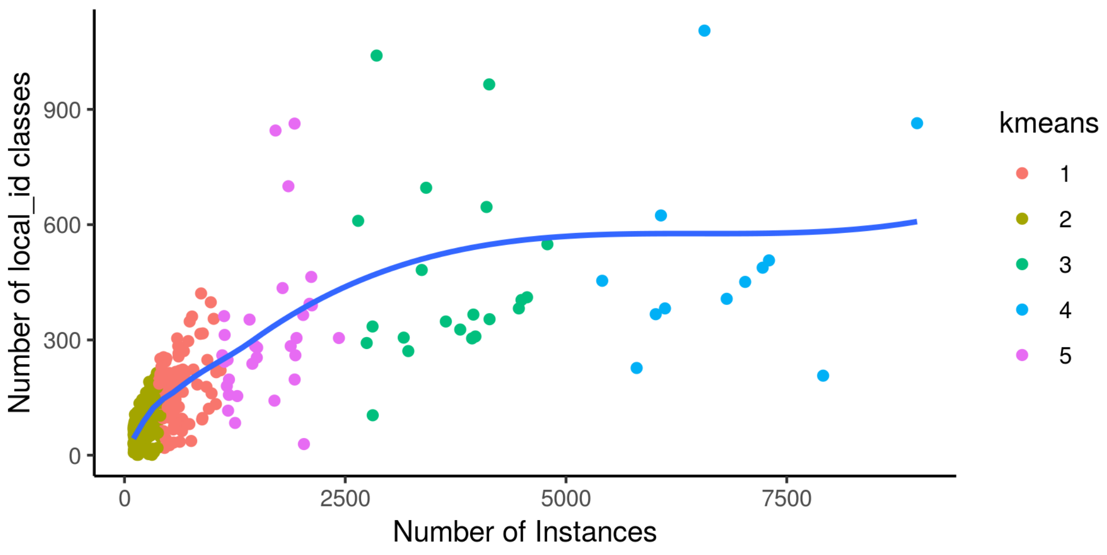


Figure 4.10: Kmeans sampling by number of instances and number of local_id classes

We ended up with 24 combinations. Sample combinations range from 126 to 5800 number of instances and from 1 to 610 number of local_id classes, while having a good diversity in the other DFS. The odd case of the number of client_id classes remains, being mostly 1, apart from 3 cases. We can expect NDCG values of client_id to be skewed to 1, but that’s the reality of the dataset.

entity_id	operator_id	NumberOfInstances	Nclient_id	Nlocal_id	Nproblem_id	Nproblem_parent_id	Npriority
10	37	867	43	421	47	13	4
24	228	272	1	97	58	15	4
24	308	1252	1	84	64	20	4
24	328	399	1	130	115	26	4
24	355	188	1	66	57	16	4
24	386	612	1	257	123	19	4
24	760	133	1	10	12	5	2
24	1107	406	1	206	58	14	4
24	1136	3951	1	366	151	18	4
24	1988	142	1	37	56	15	4
24	2175	142	1	1	44	11	3
27	193	429	19	218	32	12	4
39	395	190	1	109	16	8	4
45	1593	168	1	105	12	12	3
62	589	126	6	19	11	5	4
67	701	247	1	98	56	11	4
72	762	510	1	26	43	13	4
75	828	626	1	35	8	8	4
75	829	173	1	26	8	8	4
82	1864	366	1	214	93	22	4
90	982	390	1	125	54	16	4
94	1027	2646	1	610	87	15	4
110	1372	2032	1	29	9	8	4
134	1633	5800	1	227	151	23	4

Table 4.4: Sample 1 summary for RPE algorithm tests

We also wanted to test the most extreme cases, so another sampling strategy was used. Both samples will be tested separately. In the extreme case, it was decided to select all combinations that had more than 4000 instances, producing the sample set that can be observed in table 4.5.

entity_id	operator_id	NumberOfInstances	Nclient_id	Nlocal_id	Nproblem_id	Nproblem_parent_id	Npriority
24	402	6016	1	367	229	24	4
24	504	6121	1	382	210	24	4
24	565	7029	1	451	252	22	4
24	1137	4559	1	411	184	21	4
24	1138	4496	1	404	157	19	4
40	338	7912	1	207	280	33	4
41	339	4466	1	382	59	10	4
61	571	7299	3	507	56	13	4
64	636	18424	1	941	91	15	4
66	659	6820	1	407	113	16	4
66	836	4134	1	354	81	13	4
67	697	6075	1	624	116	13	4
82	876	5410	1	454	225	30	4
84	892	4789	1	549	149	13	4
86	1067	4130	1	965	34	10	2
87	910	8976	1	864	262	17	4
103	1176	6568	1	1105	159	22	4
104	1177	7227	1	488	86	15	4
126	1537	4100	1	646	57	13	4
134	1633	5800	1	227	151	23	4

Table 4.5: Sample 2 summary for RPE algorithm tests

While it is usually considered that the number of instances has a direct correlation with the amount of knowledge that can be obtained, the increase of instances also makes the average number of classes for any target increase. In this case, not only the local_id reaches high values (1105) but also the problem_id reaches unseen values so far (280). It is expected really high validation times for this sample set, as well as, a general reduction in average NDCG score. This sample has 20 combinations.

4.4.2 Results Sample 1

The complete form validation of all the sampled data took *13813 seconds* (230.2 minutes) while on average each entity/operator combination took *576 seconds* (9.6 minutes).

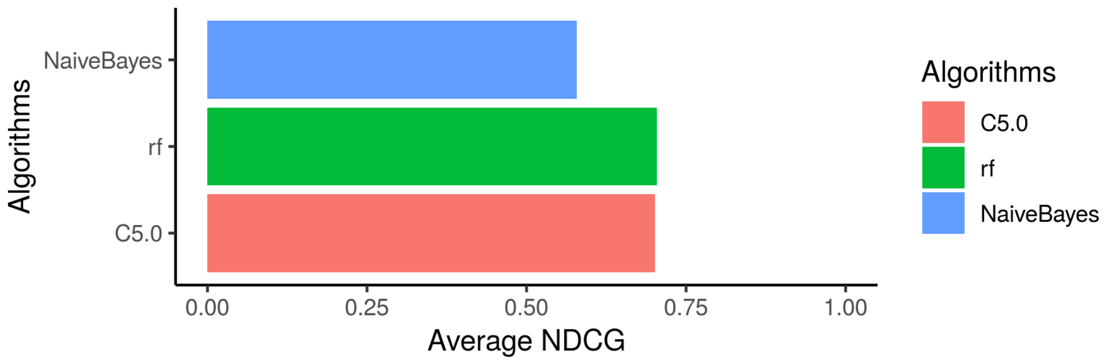


Figure 4.11: RPE Sample 1 - Average NDCG

The average NDCG score for C5.0, Random Forest and Naive Bayes was *0.701*, *0.704* and *0.579*, respectively. A plot of that result can be observed in figure 4.11. Figure 4.12 shows NDCG values distribution, for each possible target attribute.

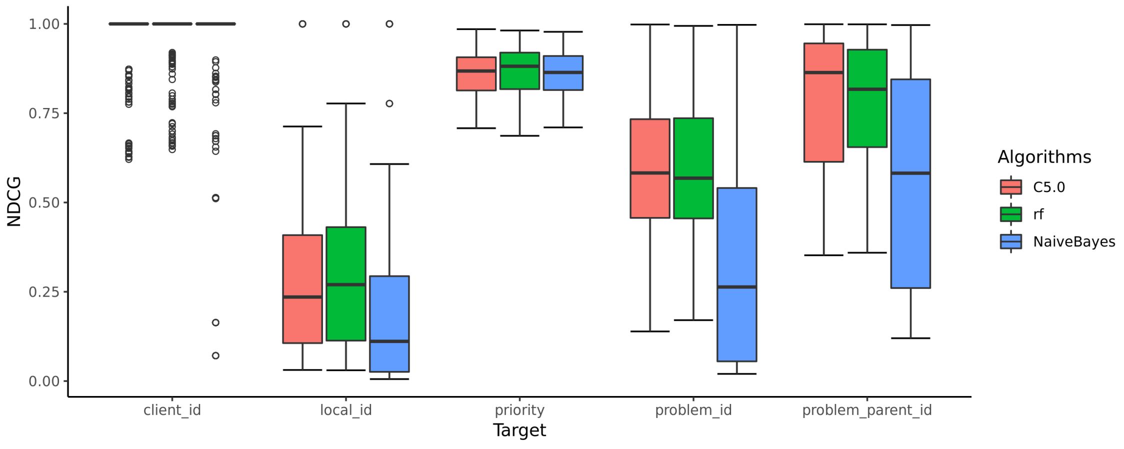


Figure 4.12: RPE Sample 1 - Average NDCG for each Target

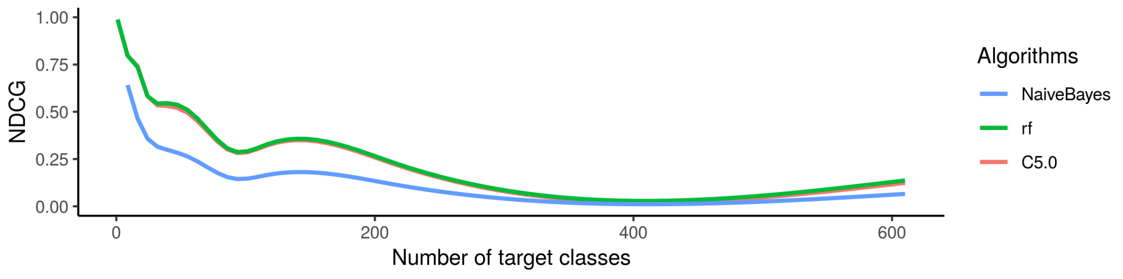


Figure 4.13: RPE Sample 1 - NDCG by Number of Classes

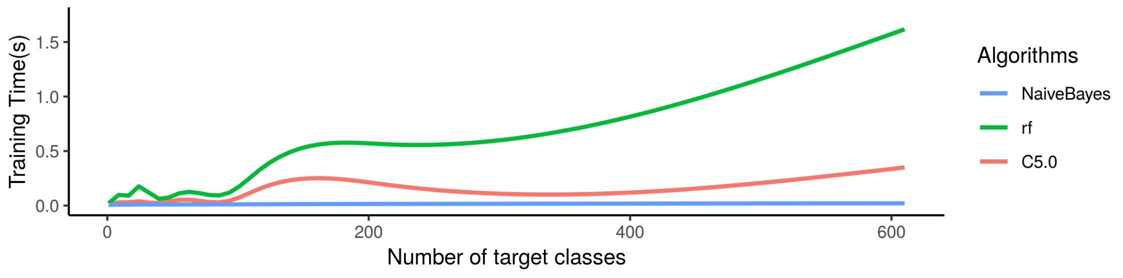


Figure 4.14: RPE Sample 1 - Training Time by Number of Classes

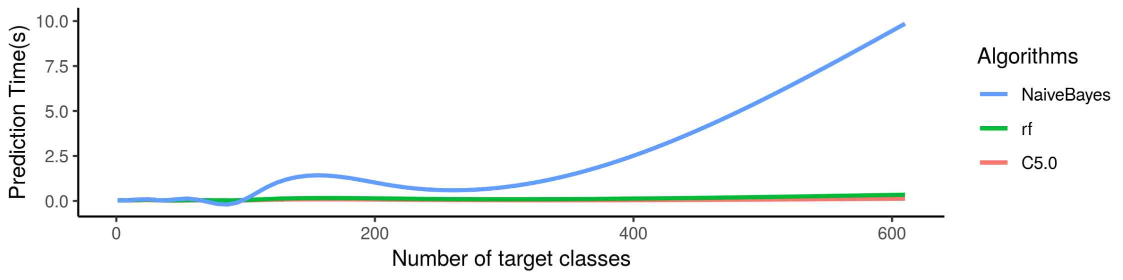


Figure 4.15: RPE Sample 1 - Prediction Time by Number of Classes

We can again, observe a decrease in NDCG with the increase of the number of possible target classes, see figure 4.19.

In figure 4.14, we see Random Forest being the most affected by the number of classes, with *2.7 seconds* of training time, at 610 classes. C5.0 reaches *2.3 seconds*, at 151 classes, while decreasing in higher number of classes. Another factor must have influenced this value. Naive Bayes's training time is close to zero, independently of the number of classes. It is important to note that, the plotted curve exhibits the average values at a certain value of number of classes, and not the top values.

In terms of prediction time, figure 4.15, Naive Bayes reaches *16.8 seconds*, at 610 classes, while for the same number of classes also has values at around 3 seconds, which means that the addition of certain features influences the prediction time. C5.0 and Random Forest have negligible prediction times throughout the whole sample.

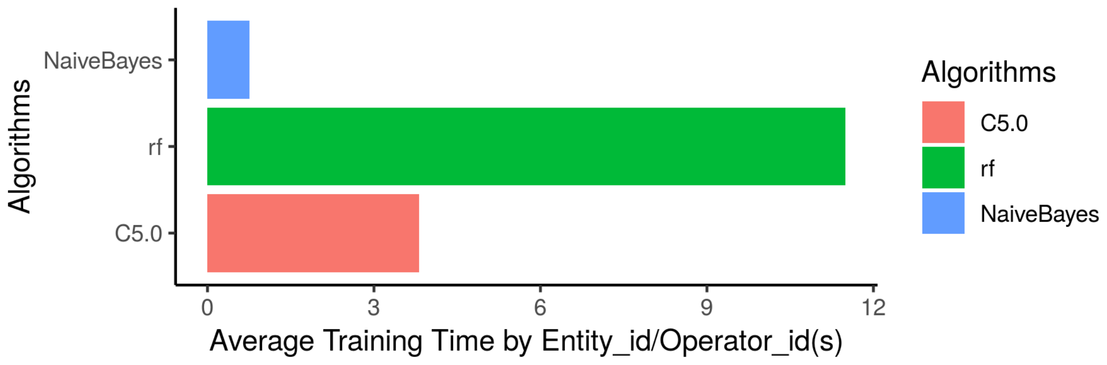


Figure 4.16: RPE Sample 1 - Average Training Time for an entity_id/operator_id combination

The average training time for a single entity_id/operator_id combination, see figure 4.16, is *3.8*, *11.5* and *0.76 seconds* for C5.0, Random Forest and Naive Bayes, respectively.

4.4.3 Results Sample 2

For this sample with more extreme cases, the complete form validation of all the sampled data took *53392 seconds* (889.9 minutes) while on average each entity/operator combination took *2669 seconds* (44.5 minutes).

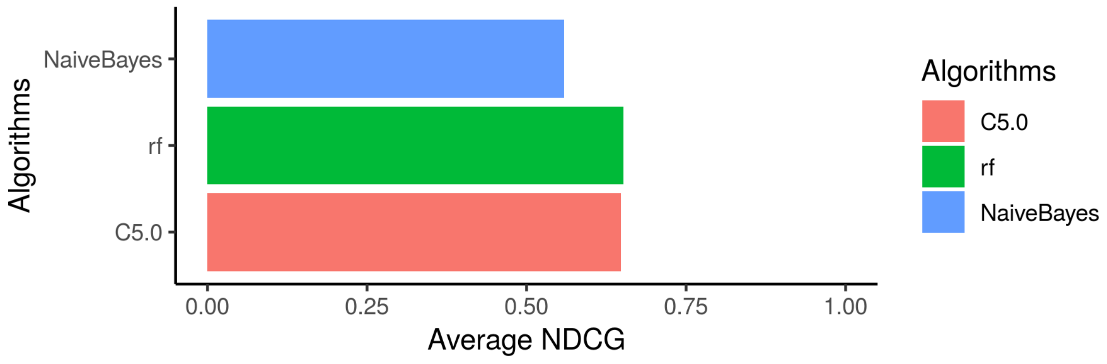


Figure 4.17: RPE Sample 2 - Average NDCG

The average measured NDCG scores, see figure 4.17, were *0.648*, *0.652* and *0.559* for C5.0, Random Forest and Naive Bayes, accordingly. Regarding each target variable, which can also be observed in figure 4.12, although almost all dropped a bit, the local_id variable takes the biggest dive in terms of NDCG, due to the consistent high amount of possible classes it has throughout all the sample set.

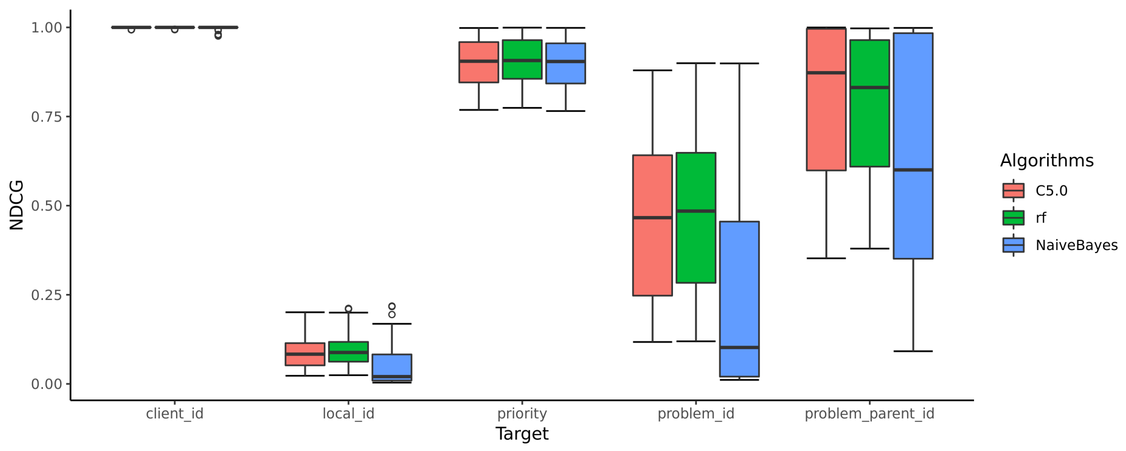


Figure 4.18: RPE Sample 2 - Average NDCG for each Target

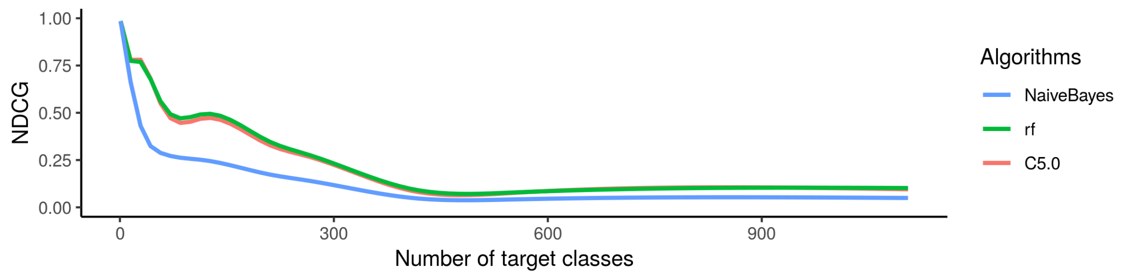


Figure 4.19: RPE Sample 2 - NDCG by Number of Classes

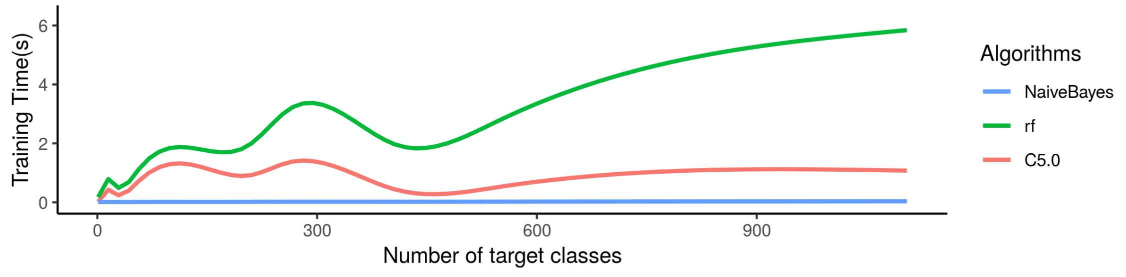


Figure 4.20: RPE Sample 2 - Training Time by Number of Classes

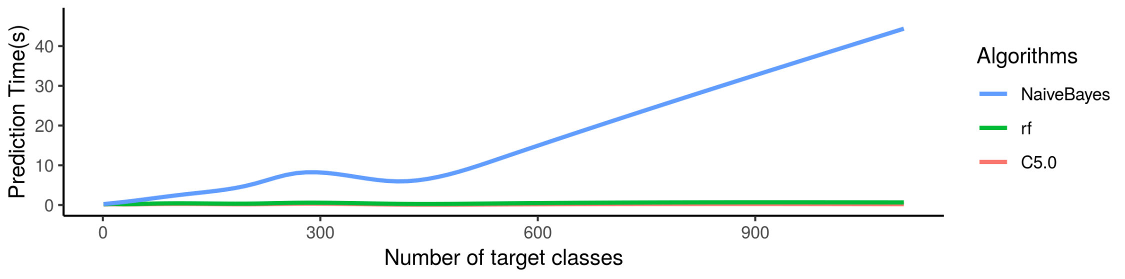


Figure 4.21: RPE Sample 2 - Prediction Time by Number of Classes

Figure 4.19 exhibits the already observed behaviour of NDCG in relation to the number of possible target classes.

The training time, see figure 4.20, also follows the same behaviour as before, albeit having higher values overall. For Random Forest, it reaches, at 864 classes, *13 seconds*, while C5.0 reaches, at 91 classes, *9 seconds*, again, probably motivated by other factors.

In terms of prediction time, as can be seen in figure 4.21, Naive Bayes stays the loser, reaching *92.6 seconds*, at 941 classes.

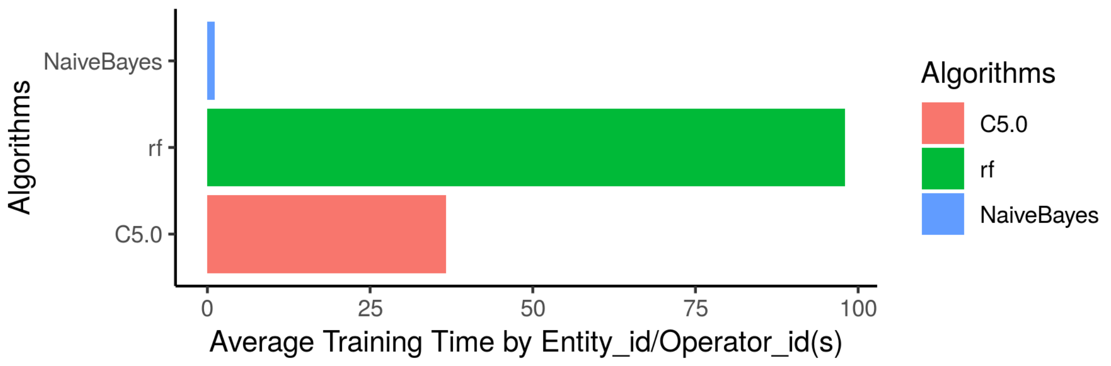


Figure 4.22: RPE Sample 2 - Average Training Time for an entity_id/operator_id combination

The average training time for each entity_id/operator_id combination, in summary, is a scaled version of the previous sample, reaching *36.7*, *98* and *1.16 seconds* for C5.0, Random Forest and Naive Bayes, respectively, which can be observed in figure 4.22.

4.4.4 Results Analysis

Both samples show really interesting average NDCG values, specially for C5.0 and Random Forest. With NDCG score above 0.5 we can assume that, on average, the first three ranks will contain relevant values. Sample 2, considered more extreme in both local_id classes and in the number of instances, shows a small drop in average NDCG values, due to really low values for local_id as target, mainly. In reality, all algorithms show a huge decrease, and eventual stagnation at really low values of NDCG, at high number of classes. Still Naive Bayes NDCG drops faster, as seen in figure 4.13 and figure 4.19. Naive Bayes is the clear loser regarding NDCG.

In terms of pure ranking quality, and specially for C5.0 and Random Forest, this experiment proves the viability of this approach. We can see how the NDCG behaves for each of the DFS combination in Appendix B. This behaviour is similar to what was observed in 4.3.2, while the client_id and local_id have less interaction due to the simplicity of the first, for both samples.

In the matter of training time, Random Forest reveals a flaw. It exhibits around three times more training time than C5.0, for a single entity_id/operator_id combination. It also shows a higher growth of this value at high number of classes. These values, although high, do not reveal the impossibility of using Random Forest, as it depends on the problem and on the intended update rate of the models, although it should still be taken into account. They can also be lowered by reducing the number of trees the algorithm uses. Naive Bayes simple training process shows really fast training times.

On another hand, while fast at training, Naive Bayes shows concerning prediction times, specially at high number of classes, observed in figures 4.15 and 4.21, reaching values at which the system cannot be considered in real time.

Although considerably higher in sample 2, the times measured in both samples reveal a consistent behaviour of all algorithms, that can be observed by the similar appearance of the plots in each sample set.

C5.0 appears to be the most balanced of the three. While it has competitive NDCG scores, it also exhibits good training and prediction times.

Lastly, and specially on sample 2, the total training time for each entity_id/operator_id and also the complete validation process time show extremely high values. Sample 2 results took around 14.8 hours to be obtained. While in a real scenario only one algorithm and one pass through the data is needed, compared to the 5 passes for 3 algorithms done in this experiment, these values are still alarming. Which leads us to Experiment 3, in which we will attempt to reduce these same values.

4.5 Experiment 3 - RPE multiple algorithms with feature pruning

The main goal of this experiment is to test an hypothetical possibility of reducing the number of models, and by consequence, reducing the total training time for the complete form. This approach was already explained in section 3.2.3, and focuses on identifying the relevant and irrelevant DFS of each target, by training a model for any given target with all the other DFS and extracting the feature importance. This can be seen as a smart approach compared to the more brute force approach of the experiments 1 and 2.

If we can maintain similar NDCG results while reducing the total training time by entity_id/ operator_id combination this experiment will be a success.

It is also important to note that, for comparison purposes, the pruned models with the irrelevant features are still produced in the results, by copying the results of the corresponding model with only the relevant features, while identifying it as fake, for training time purposes. This also simulates the real world scenario, where we would end up only using models with the relevant features with this approach. An example of how models with irrelevant features are assigned to models with only relevant ones can be seen in table 3.2.

Lastly, and also for comparison reasons, the same samples and the same folds, within each target of each sample, of experiment 2 will be used.

4.5.1 Results Sample 1

The complete form validation of all the sampled data took *3693 seconds* (61.6 minutes) while on average each entity/operator combination took *154 seconds* (2.6 minutes). These results were produced in about 26.7% of the time it took in section 4.4.2 to be obtained.

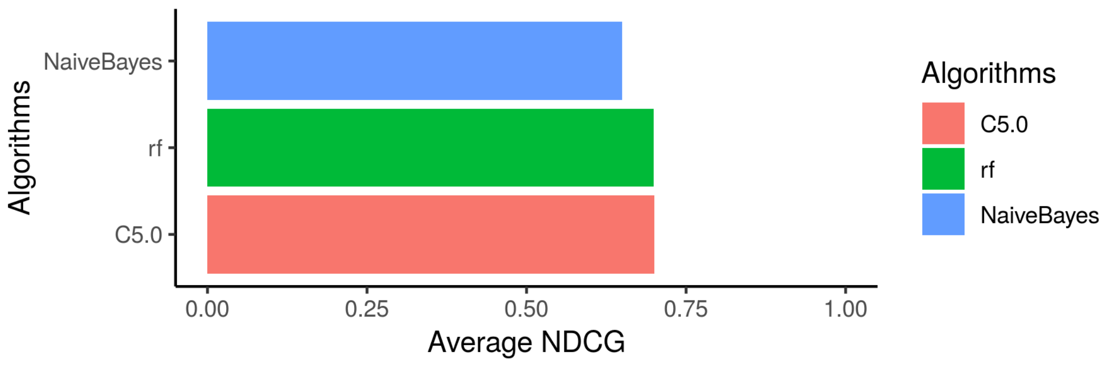


Figure 4.23: Smart RPE Sample 1 - Average NDCG

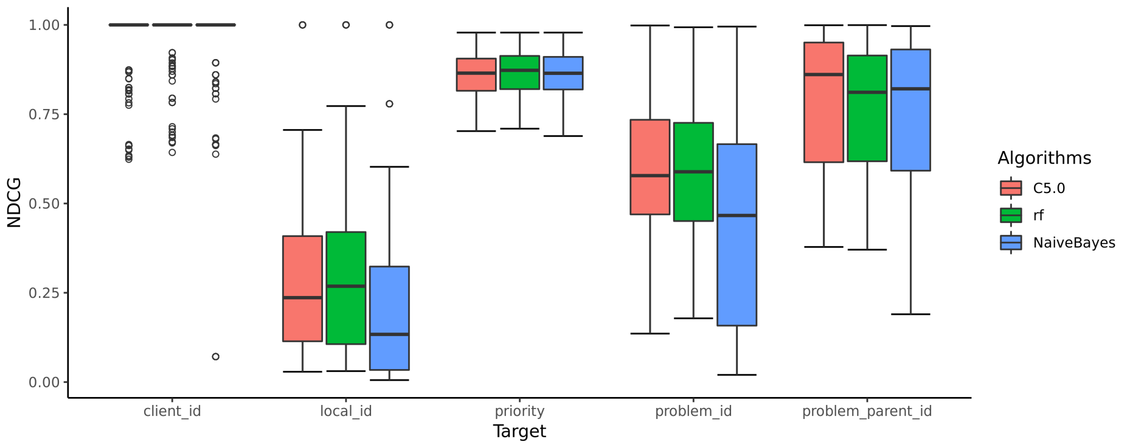


Figure 4.24: Smart RPE Sample 1 - Average NDCG for each Target

The average measured NDCG score, seen in figure 4.23, was *0.7*, *0.699* and *0.65* for C5.0, Random Forest and Naive Bayes, respectively. Pretty close results to what was seen in section 4.4.2, with the scores *0.701*, *0.704* and *0.579*. Naive Bayes actually shows considerably better results with this approach.

In figure 4.24, we can observe that Naive Bayes, compared to the results from section 4.4.2, shown in figure 4.12, has an increase in NDCG score for targets problem_id and problem_parent_id. For the other algorithms the results are close to identical for each target.

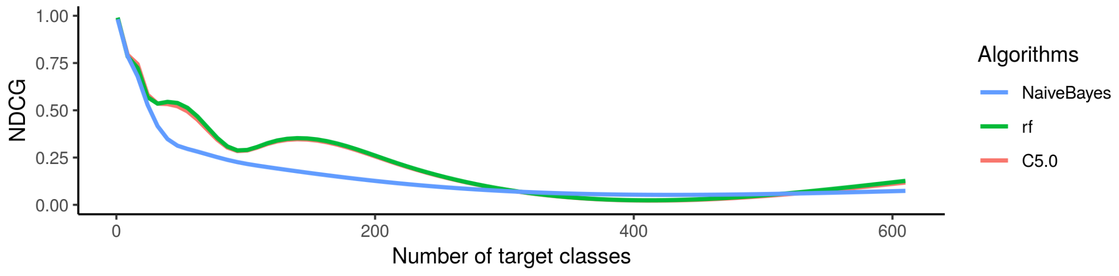


Figure 4.25: Smart RPE Sample 1 - NDCG by Number of Classes

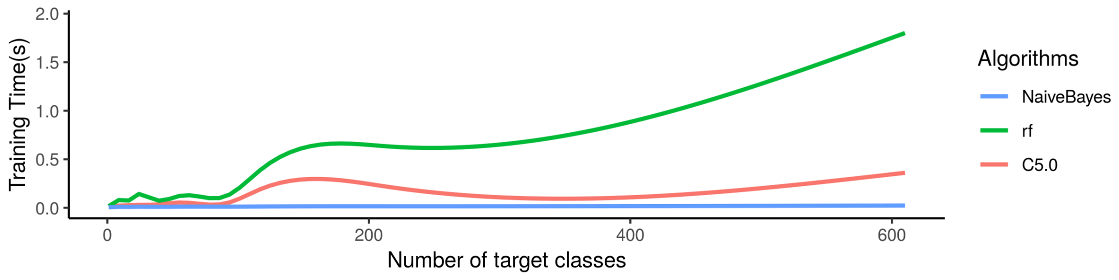


Figure 4.26: Smart RPE Sample 1 - Training Time by Number of Classes

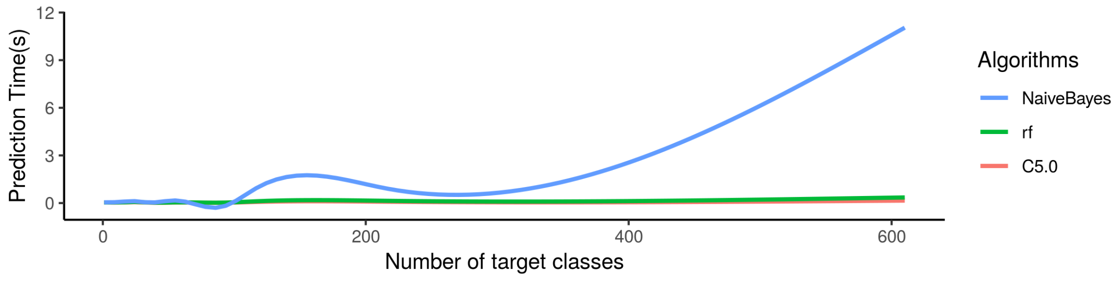


Figure 4.27: Smart RPE Sample 1 - Prediction Time by Number of Classes

The behaviour of NDCG with the increase of the number of possible target classes, seen in figure 4.25, is almost an identical replica of the one shown in figure 4.13.

The training time of each model, see figure 4.26, stays pretty much that same, peaking, at 151 classes, *2.7 seconds* for C5.0 and, at 610 classes, *2.88 seconds* for Random Forest. There is no reason for these values to change much, as this sample is exactly the same sample that was used in section 4.4.2. C5.0 probably reached higher values due to some outside interference.

The same is true for prediction time, in figure 4.27, having Naive Bayes reach *18.7 seconds*, at 610 classes. Again, slightly higher peak values, but probably due to variations in the processing power.

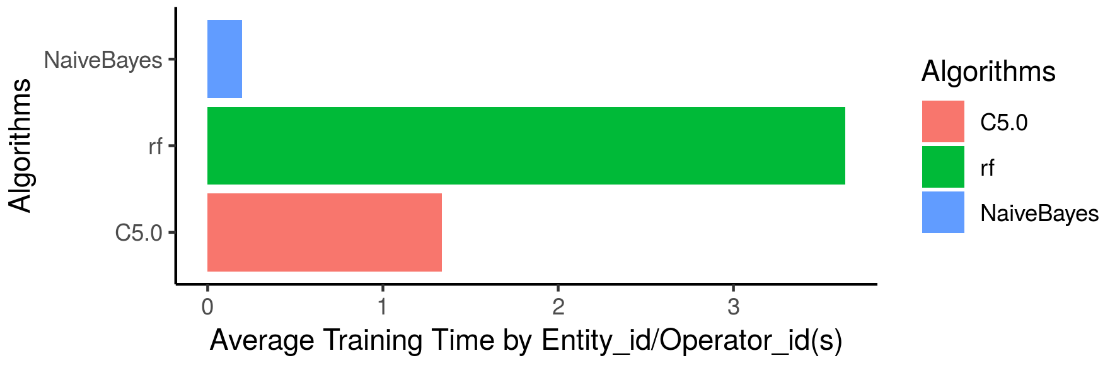


Figure 4.28: Smart RPE Sample 1 - Average Training Time for an entity_id/operator_id combination

In the average training time for each entity_id/operator_id combination, seen in figure 4.28, is where the possible gain can be achieved. Although the figures in 4.16 and 4.28 look identical, the scale is completely different. In this case, the C5.0, Random Forest and Naive Bayes measured times were *1.34*, *3.64* and *0.197 seconds*, respectively, being about 35%, 32% and 26% of the values obtained in section 4.4.2.

We can observe a comparison of the average training time for each entity_id/operator_id combination for both experiments in figure 4.29.

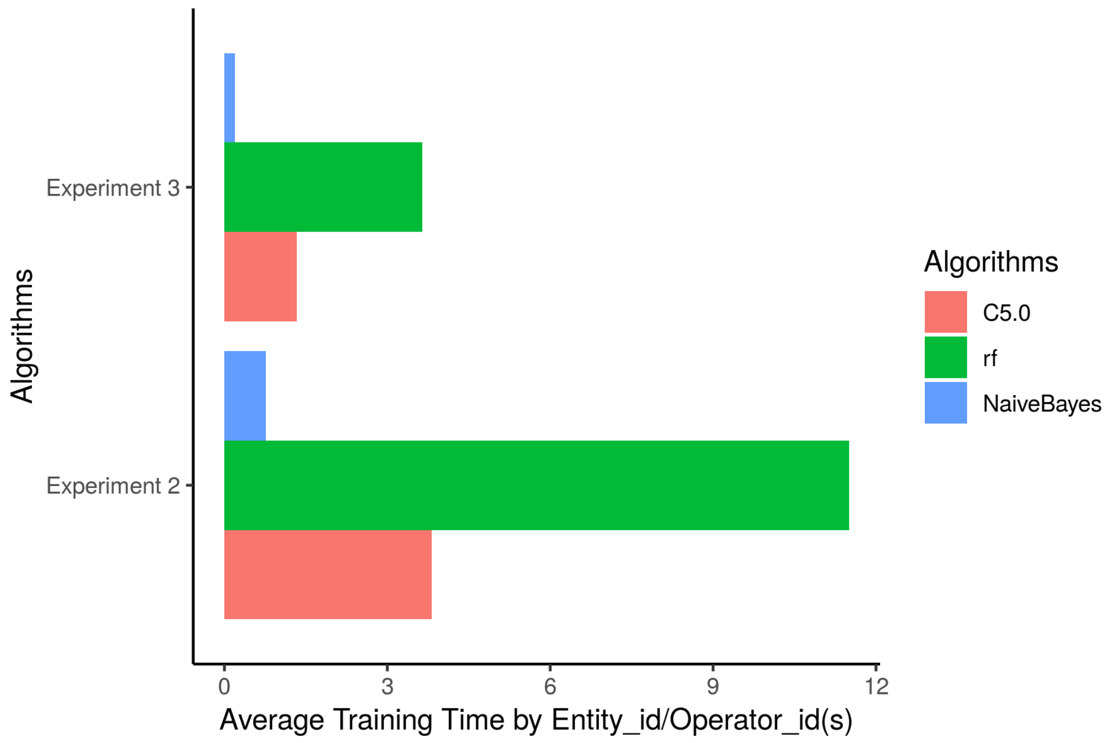


Figure 4.29: Sample 1 - Average Training Time for an entity_id/operator_id comparison

In this experiment, is also possible to evaluate the number of generated models. An average of this value for each target and for an entity_id/operator_id combination can be seen in figures 4.30 and 4.31, respectively. The dashed line shows the static values for the other experiments.

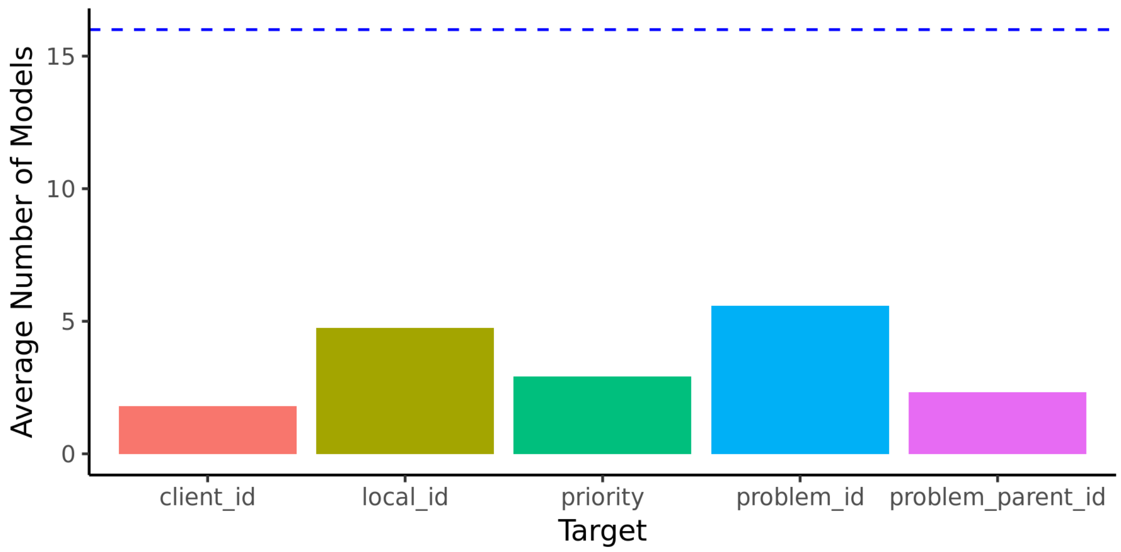


Figure 4.30: Smart RPE Sample 1 - Average number of models per Target

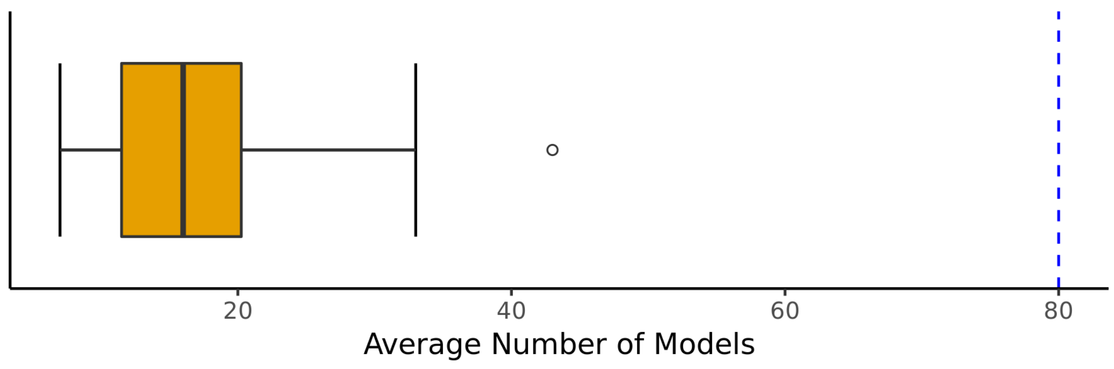


Figure 4.31: Smart RPE Sample 1 - Average number of models for an entity_id/operator_id combination

On average, *17.4 models* were generated, for each combination, 21.75% of the models used in the other two experiments.

4.5.2 Results Sample 2

The complete form validation of all the sampled data took *12056 seconds* (200.9 minutes) while on average each entity/operator combination took *603 seconds* (10.1 minutes). These results were produced in about 22.6% of the time it took on experiment 2 to be obtained.

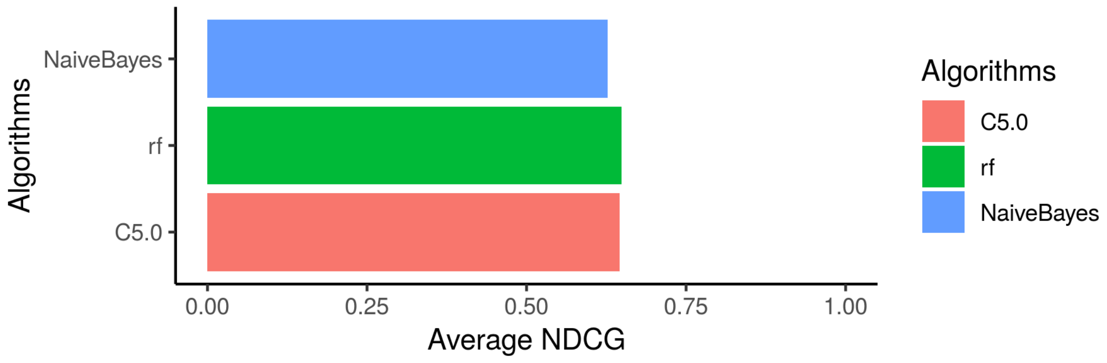


Figure 4.32: Smart RPE Sample 2 - Average NDCG

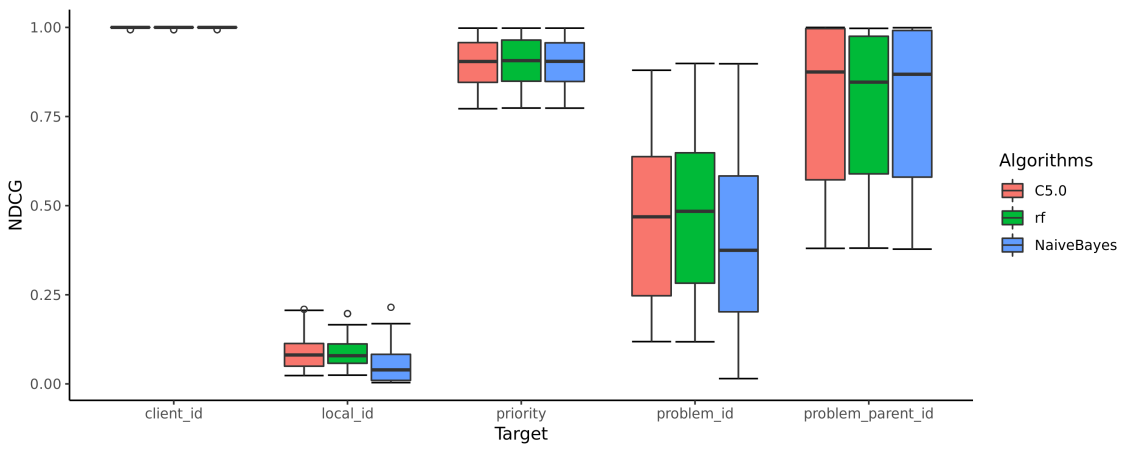


Figure 4.33: Smart RPE Sample 2 - Average NDCG for each Target

In resemblance to section 4.5.1, the measured average NDCG values for C5.0 and Random Forest, that can be seen in figure 4.32, have an insignificant variation, scoring *0.646* on the first and *0.649* on the second, compared to the *0.648* and *0.652* values observed in section 4.4.3. Naive Bayes also follows the same behaviour seen in section 4.5.1, increasing its score to *0.627* compared to *0.559* observed in section 4.4.3.

The relation of average NDCG for each Target follows the same behaviour exhibited in section 4.5.1.

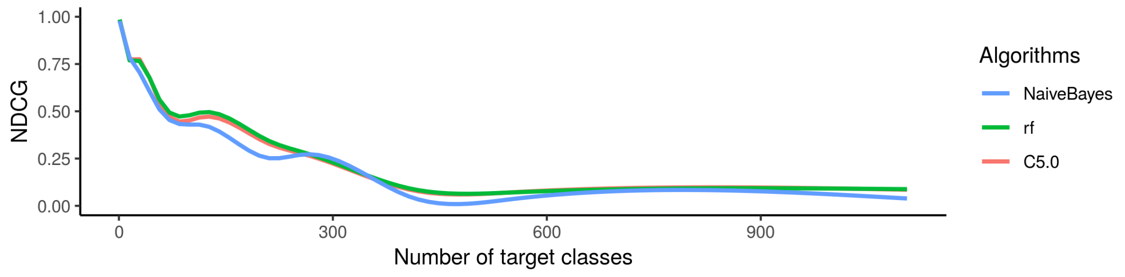


Figure 4.34: Smart RPE Sample 2 - NDCG by Number of Classes

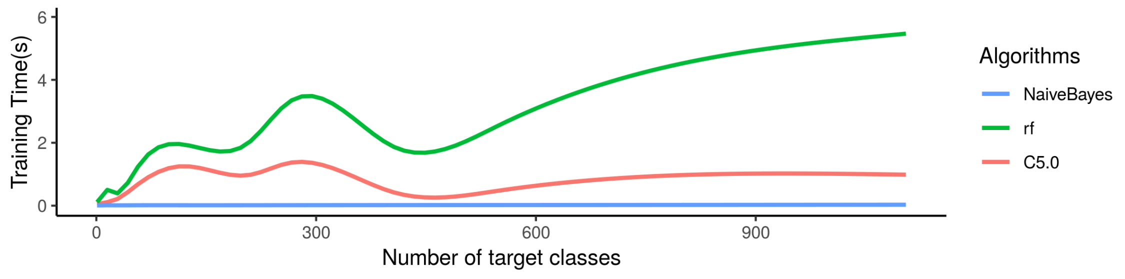


Figure 4.35: Smart RPE Sample 2 - Training Time by Number of Classes

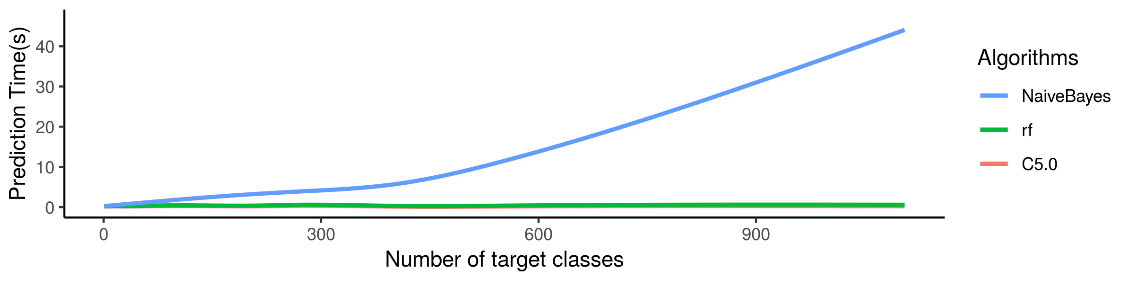


Figure 4.36: Smart RPE Sample 2 - Prediction Time by Number of Classes

Compared to section 4.4.3, the NDCG score, training time and prediction time in relation to the number of target classes, also follows the same behaviour observed in section 4.5.1.

The training time, seen in figure 4.35, reaches *7.4 seconds*, at 91 classes, for C5.0 and *11.6 seconds*, at 864 classes, for Random Forest.

The prediction time (figure 4.36) for Naive Bayes peaks at *91.5 seconds*, at 941 classes.

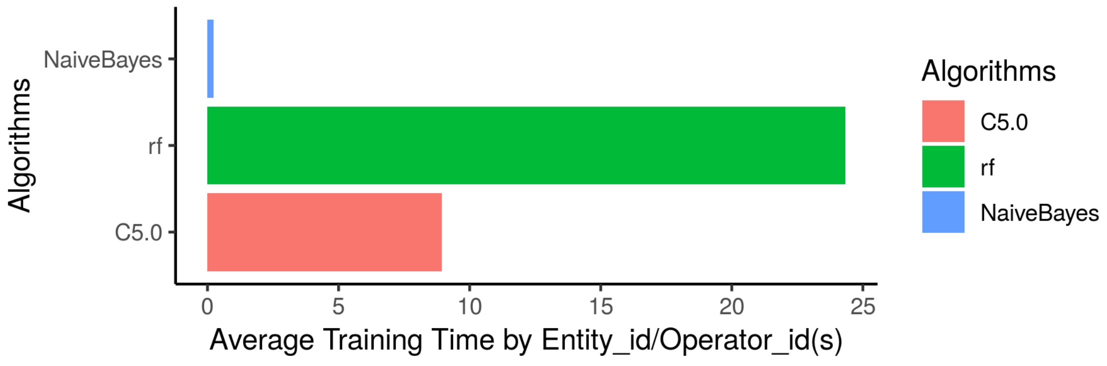


Figure 4.37: Smart RPE Sample 2 - Average Training Time for an entity_id/operator_id combination

Again, the average training time for each entity_id/operator_id combination, see figure 4.37, seems like a replica from section 4.4.3 but with a completely different scale. C5.0, Random Forest and Naive Bayes scored average values of *8.9*, *24.3*, and *0.237 seconds*, respectively, being about 24.3%, 24.8% and 20.4% of the values observed in the last experiment.

We can see a comparison between the two plots in figure 4.38.

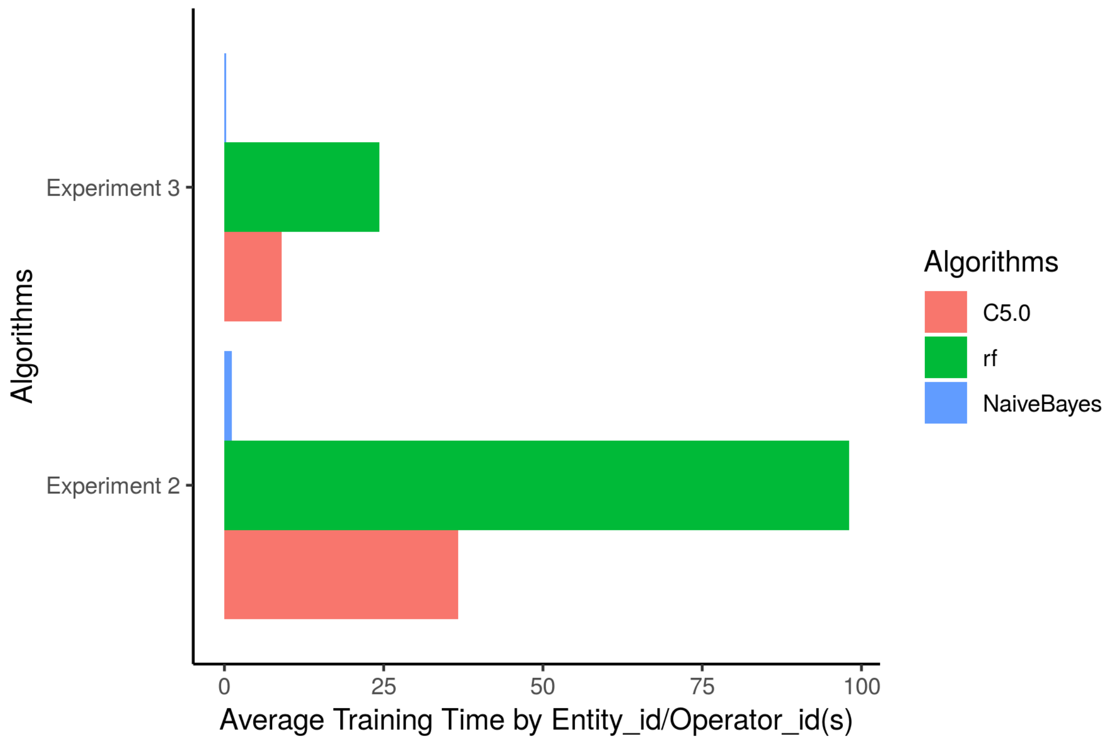


Figure 4.38: Sample 2 - Average Training Time for an entity_id/operator_id comparison

In resemblance to section 4.5.1, is possible to evaluate the number of generated models. An average of this value for each target and for an entity_id/operator_id combination can be observed in figures 4.39 and 4.40, accordingly.

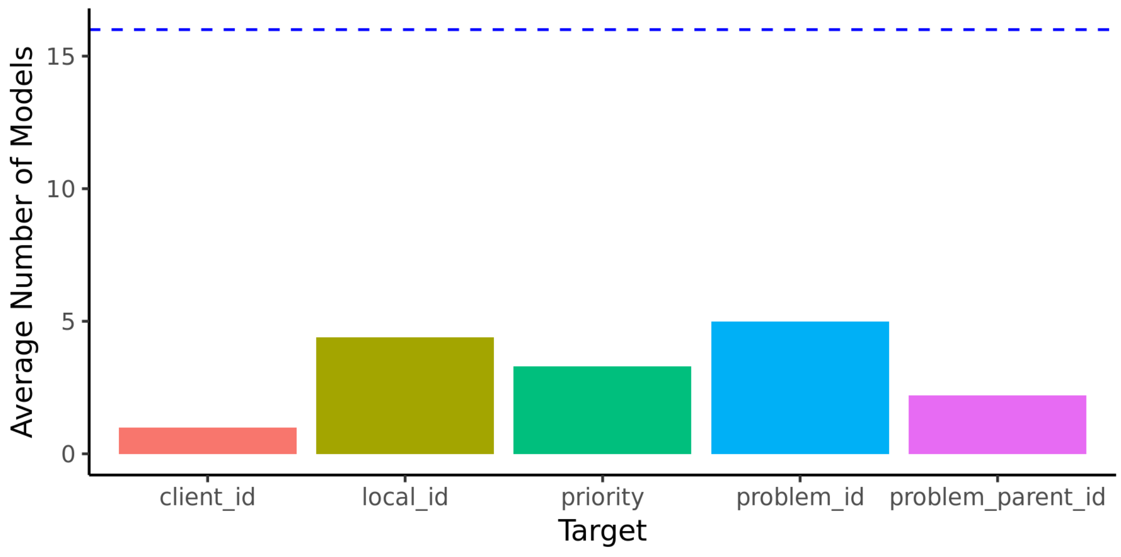


Figure 4.39: Smart RPE Sample 2 - Average number of models by Target

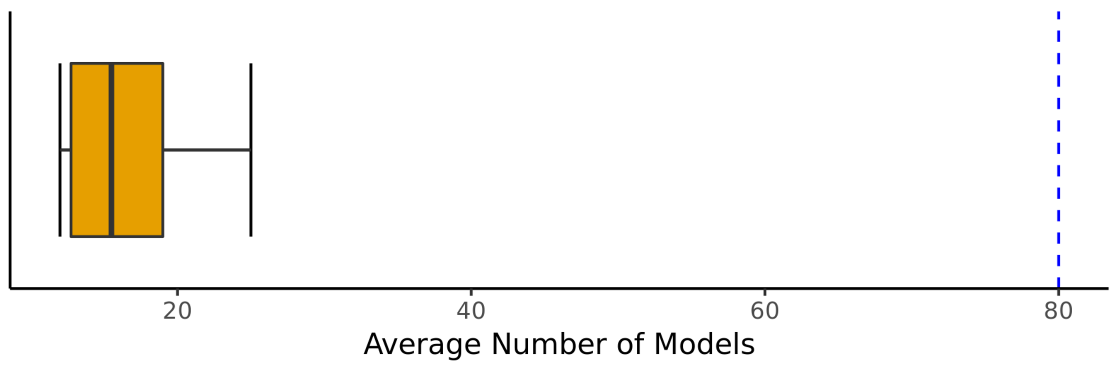


Figure 4.40: Smart RPE Sample 2 - Average number of models for an entity_id/operator_id combination

On average, *15.9 models* were generated, for each combination, 19.88% of the models used in the other two experiments.

4.5.3 Results Analysis

This experiment shows that the model pruning approach has, on average, close NDCG results to the ones observed previously. Actually, for Naive Bayes, it improves its score. This can happen because we are removing highly correlated features, by removing irrelevant features from training. We know for a fact that Naive Bayes is susceptible to it.

In terms of the average training time for one entity_id/operator_id combination, which is directly proportional to training the whole dataset, we have gains in between 65% and 80% by reducing in the number of generated models around 80% too. This is huge, as it will allow to increase the update frequency of the models, at little to no cost of NDCG.

This experiment was successful in what it intended to achieve.

4.6 Conclusions

This chapter was focused on the actual work done and implemented, and its corresponding results.

The experimental setup, shown in section 4.1, guarantees the integrity of the whole validation process. Although not the same sampling strategy is used for all experiments, when comparing algorithms and approaches, the same training and test sets are used. It is also guaranteed that we are not making predictions with models trained with the instance being predicted.

It is important to note, also, the different processing power available to each algorithm. While Random Forest, albeit producing 500 trees, is running on two cores, the rest is running on only one. This is important to know, mainly, to compare the training times of each algorithm and approach.

Different sampling strategies were used for two reasons. The first being the assumption that RPC will take a huge amount of time, to train and predict, when the quantity of target classes is high. The second reason is that we wanted to test, for RPE, both the general cases and the more extreme ones.

Experiment 1, see section 4.3, reveals the flaw, already assumed before, of RPC. While it is competitive in terms of NDCG, its training and prediction times, specially on high amount of target classes, are shown to be infeasible for this real time system. As such, it was decided to expand the RPE

Experiment 2, see section 4.4, extends the RPE approach to other algorithms. C5.0 appears to be the most balanced of the three algorithms tested, in all three metrics. Random Forest, albeit having very competitive NDCG score, suffers from high training times. This can be decreased by reducing the amount of produced trees, at a probable NDCG cost. Naive Bayes is shown to be the worst of the three. It has inconsistent NDCG values, making its overall score lower than the rest. Still, the main issue with Naive Bayes is the huge prediction time it has at an high number of target classes. All algorithms are shown to have bad NDCG scores at really high values too, with Naive Bayes being the one more affected by it.

Experiment 3, see section 4.5, wanted to reduce the total training time, by reducing the number of needed models. It proved to be successful, reducing the total training time to approximately one fourth of the original time, while maintaining close NDCG values to the ones obtained in experiment 2. Naive Bayes actually scores higher in this experiment.

Chapter 5

Conclusions

This chapter will be centered around a summary of the full document as well as a final exploration of the problem, our solution, its main threats and future work.

5.1 Using ranking algorithms for a form suggestion system

This work is focused on expanding the already used solutions in form suggestion, to further adapt to each user needs. ML has been invading all areas of computation, improving them.

We propose another possible application of ML, in form suggestion. This approach uses several models to predict and rank each possible entry for each field, adapting its prediction during the filling process. It is expected to have better results than traditional methods, as it takes into account the past knowledge from previously filled forms, while being able to adapt to new scenarios. This solution is targeted to very specific types of forms, as it was explained in section 3.1.1.

The complexity of the training process is directly related to the number of fields it has. The number of needed models grows exponentially in relation to the number of fields. This led us to develop one possible approach that selects which fields it will use for a certain target, considerably reducing the number of needed models.

Two ranking approaches were tested in this work, the RPC algorithm and RPE approach. The first is a proved and tested LR algorithm. The second is the hypothetical approach of converting the probability estimation results of a multi-class classification algorithm into rankings. This is only possible because we have a labeled dataset.

Mostly hyper parameter free algorithms were used, for generalization purposes.

Experiment 1 shows that, while both have really interesting NDCG results, RPC exhibits really inefficient training and test times. As such, we decided to expand the RPE approach to other multi-class classification algorithms.

Experiment 2 compares C5.0, Random Forest and Naive Bayes algorithms, for the RPE approach. C5.0 comes out as being the most balanced of the three.

Experiment 3 shows that the model pruning approach works, at little to no cost of NDCG.

The experiments done show the viability of the system, from a NDCG standpoint. While viable, we cannot expect the system to always suggest the correct ranking, as there is no perfect prediction algorithm. For instance, at high number of possible target values, all algorithms tested reveal really low NDCG values. Remember also, that for suggestion purposes, we are only using a selection of the first ranked elements.

5.2 Main threats

One of the main threats of this work, is the fact that the system was not tested in a real environment. System analysis was done only on validation results.

Testing the system on only one form can also be threatening. Although this form encapsulates multiple problems, as each entity working on the form has completely different working contexts, there's no guarantee that for any other form the system still proves viable.

The system is also not able to deal properly with pure text fields, as that would lead to innumerous possible target classes, which already proved to show bad results. It would only be able to deal with it, if the user filled that text field in a finite number of ways.

Lack of data is a typical threat in ML problems. If we are confronted with that issue it is always possible to revert to typical suggestion approaches, like the most used values for a certain field, resorting to old methods to fix the problem. In our case, we removed from testing the entity_id/operator_id combinations that exhibited a low number of instances, although they are still real users.

Lastly, the system training process, even with the pruned models approach, uses alot of computation resources. This is due to the scope in which each model is trained, in our case a entity_id/operator_id combination.

5.3 Future Work

The most important work left to be done was already enunciated in section 5.2. Those are:

- To test the system in a real environment.
- To validate the system in other types of forms.

In order to validate the reduction of the form filling time, testing the system in a real environment has to be done. This means, implementing the suggestion in the visual interface of that form, and measuring the results. If the time it takes for a user to fill a form is recorded, before and after the suggestion is implemented, it is possible to do a real evaluation of the system. Another way to do it, is to count the number of hits it has. By number of hits we mean the number of times a value suggested is used.

Also, in order to prove that this solution can be generalized for any type of form, multiple forms must be tested. It can, also, allow us to perceive other behaviours, not seen in the experiments done.

Although we assumed, in chapter 4, that if an entity_id/operator_id combination had less than 100 instances we wouldn't have enough knowledge to get a good ranking, that value was not studied. It should be possible to obtain a more precise value, by studying the behaviour of NDCG on combinations with a lower amount of instances.

Other than the already referred work, it is also possible to extend the proposed system to other LR algorithms, see section 2.4.3, or other multi-class classification algorithms, in the case of a RPE approach. Also, a ranking implementation of MERCS, see section 2.7, could prove to be a viable approach, in terms of NDCG, training time and prediction time, of the problem at hand.

5.4 Final thoughts

The ML field has infinite applications in the computing field. The possibility proposed in this document proves that.

Form suggestion is not a typical theme these days. The form users are used to the way forms normally behave and the techniques used to help the form filling. This thesis can lead to very interesting results in the field. While on one hand, it attempts to use ranking in a way it was never used before and, on another hand, because form design does not usually focuses on suggestion.

To finish, we expect that our approach can lead to a new paradigm in form suggestion.

Appendix A

Experiment 1 Detailed NDCG Results

In this appendix we can find the detailed results, for each Direct Features combination for each target, for experiment 1, allowing us to perceive how the presence of certain features influence the quality of prediction.

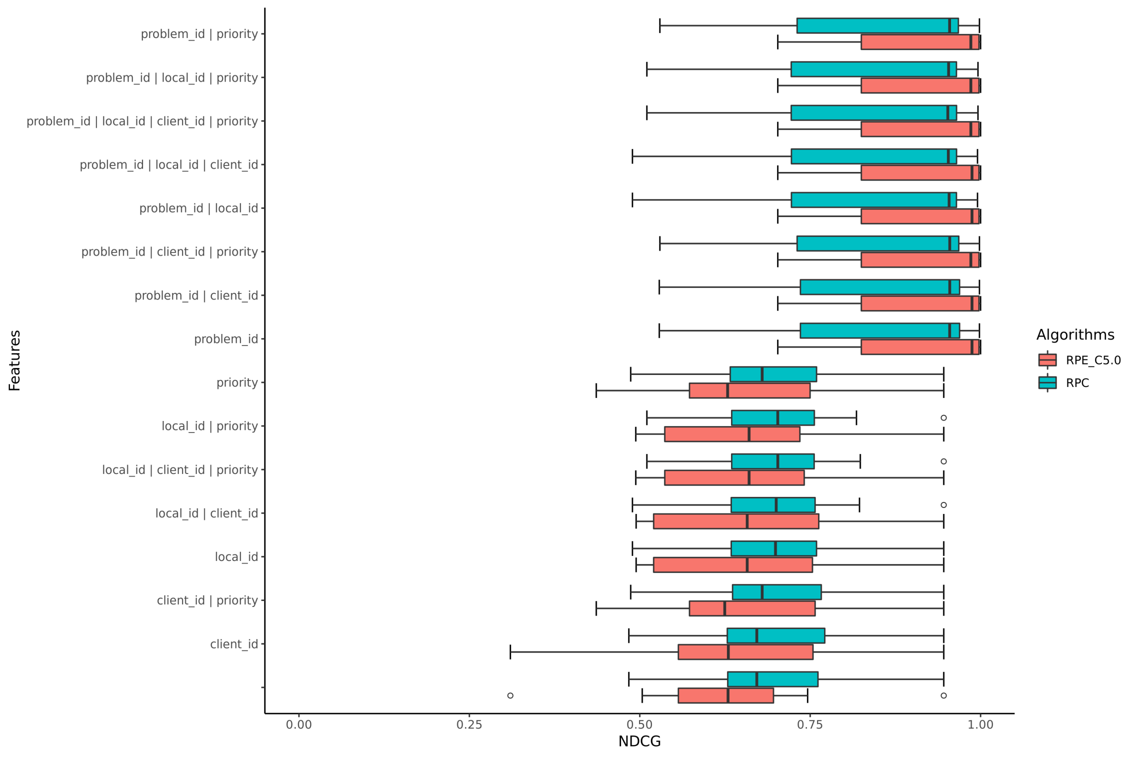


Figure A.1: RPE vs RPC - Complete NDCG for Target problem_parent_id

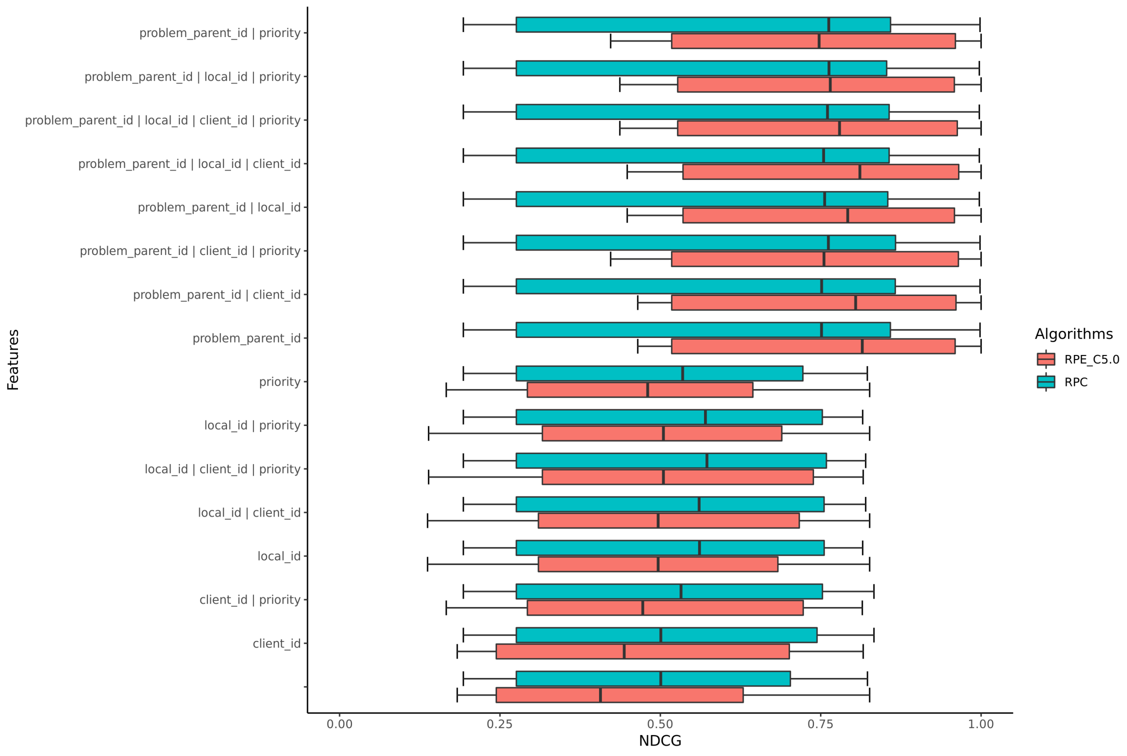


Figure A.2: RPE vs RPC - Complete NDCG for Target problem_id

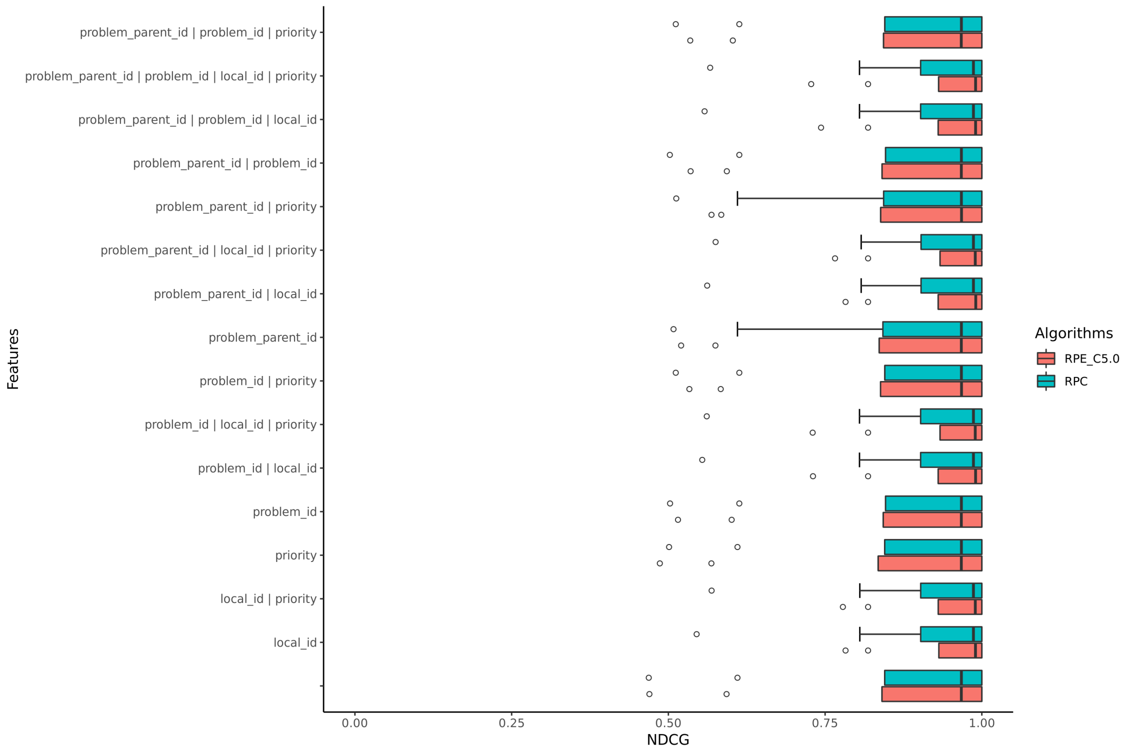


Figure A.3: RPE vs RPC - Complete NDCG for Target client_id

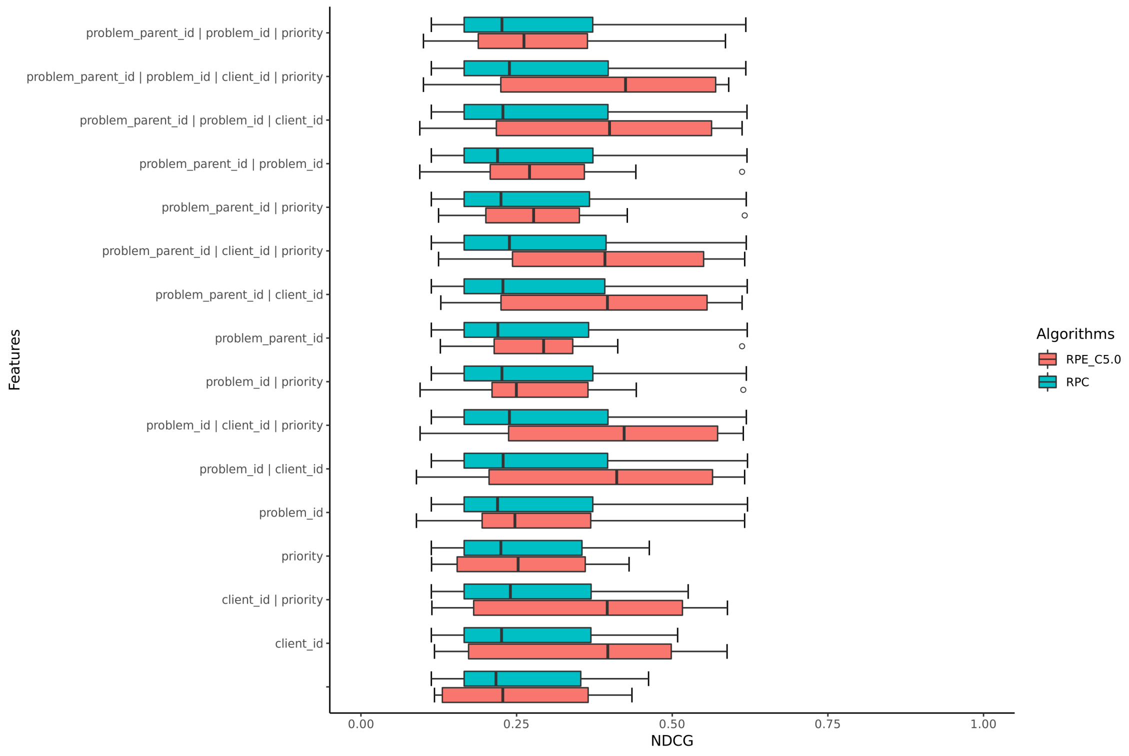


Figure A.4: RPE vs RPC - Complete NDCG for Target local_id

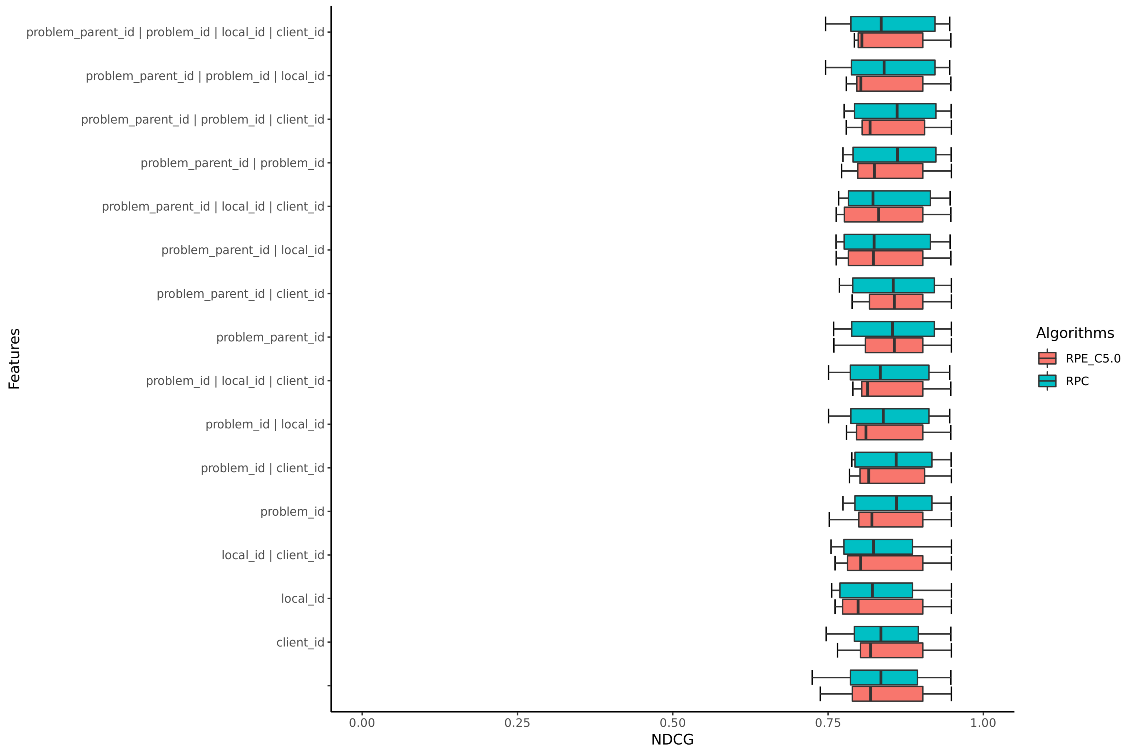


Figure A.5: RPE vs RPC - Complete NDCG for Target priority

Appendix B

Experiment 2 Detailed NDCG Results

Same as Appendix A but for Experiment 2, for both sampling strategies.

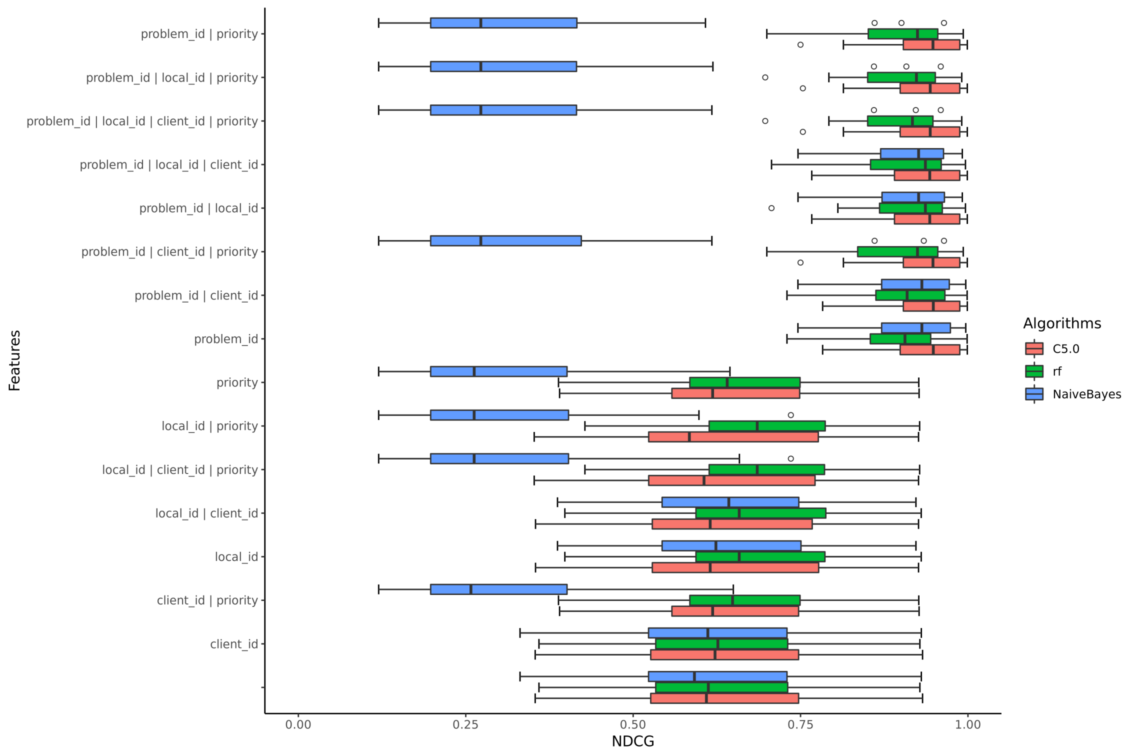


Figure B.1: RPE Sample 1 - Complete NDCG for Target problem_parent_id

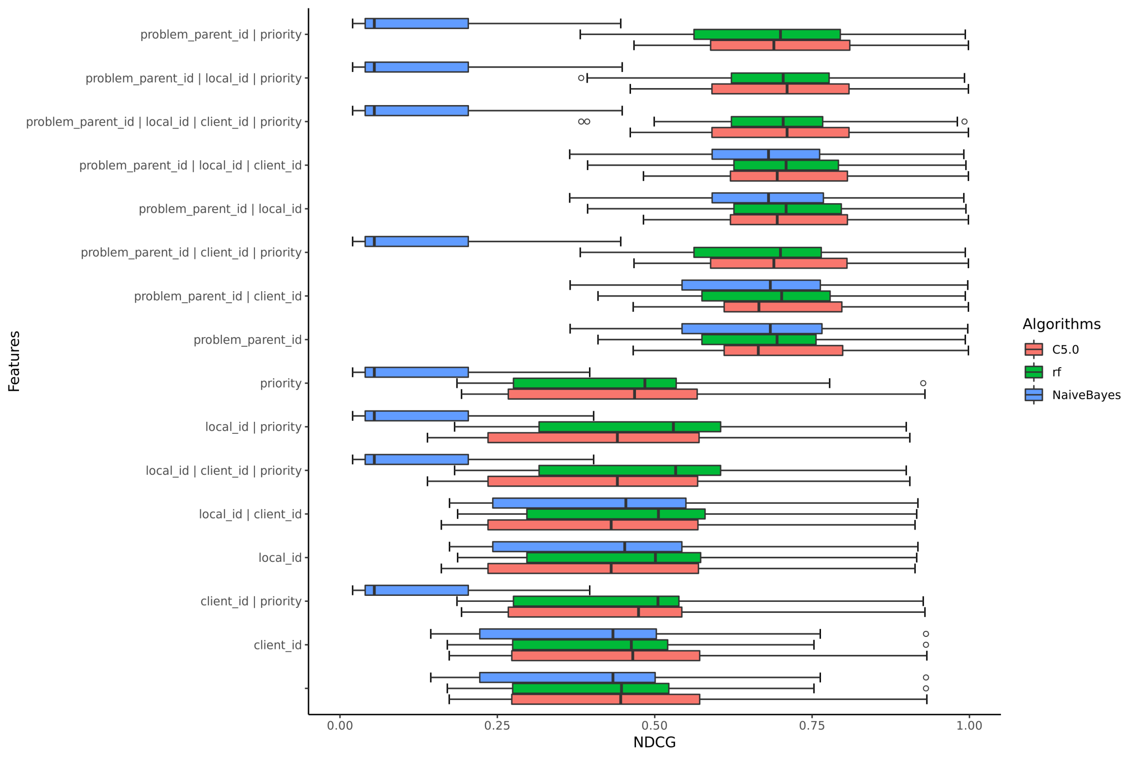


Figure B.2: RPE Sample 1 - Complete NDCG for Target problem_id

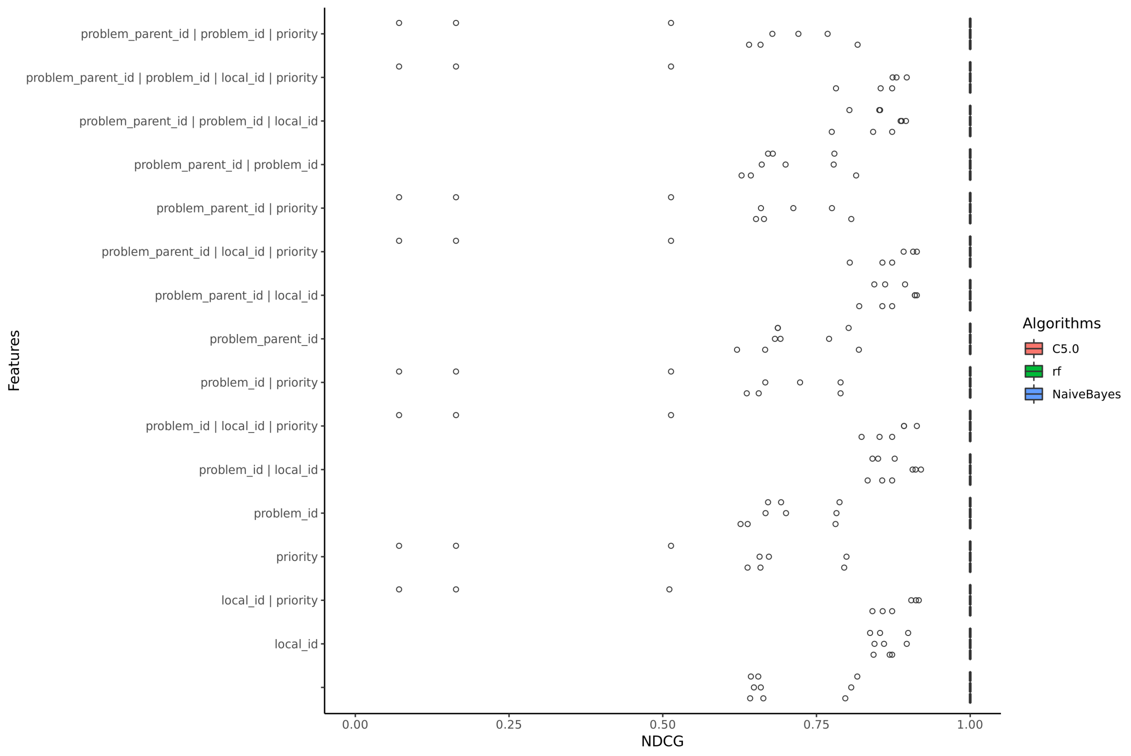


Figure B.3: RPE Sample 1 - Complete NDCG for Target client_id

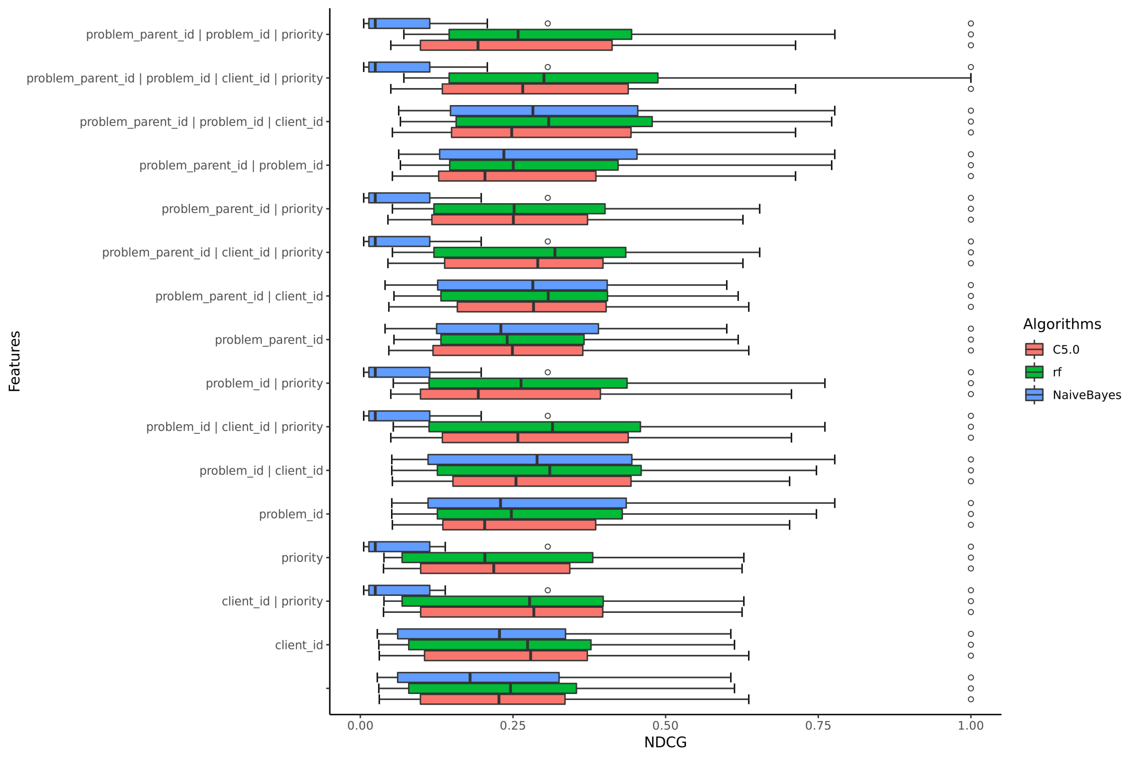


Figure B.4: RPE Sample 1 - Complete NDCG for Target local_id

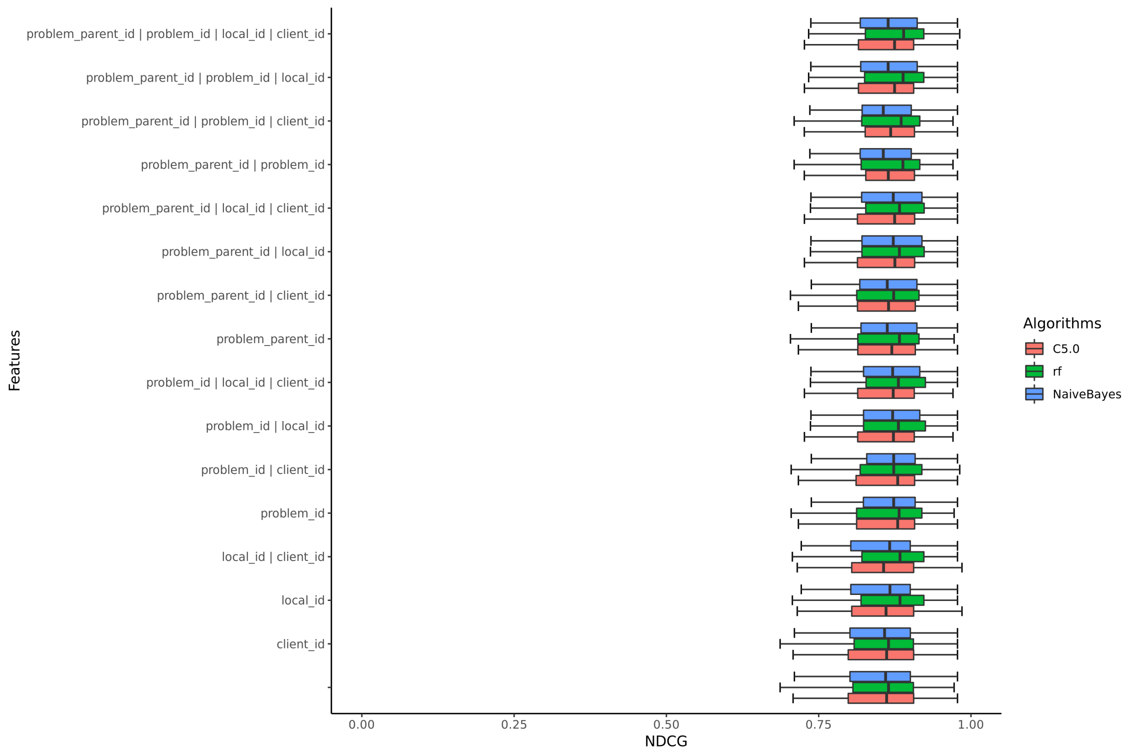


Figure B.5: RPE Sample 1 - Complete NDCG for Target priority

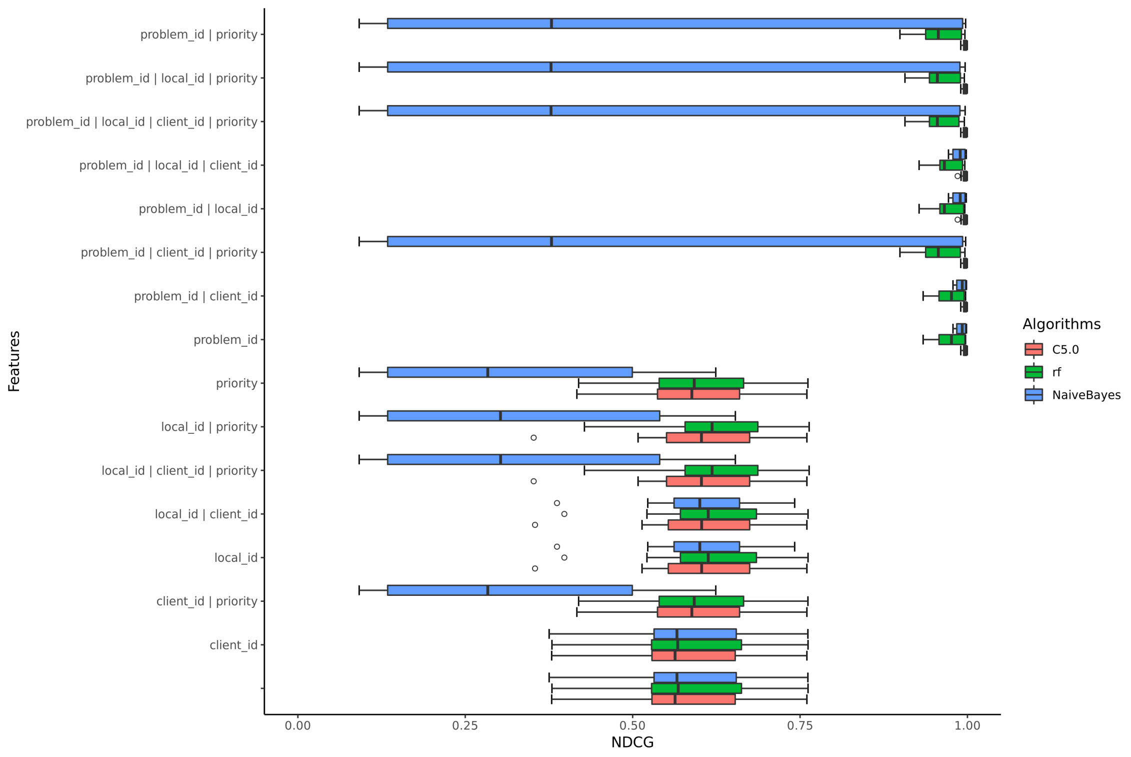


Figure B.6: RPE Sample 2 - Complete NDCG for Target problem_parent_id

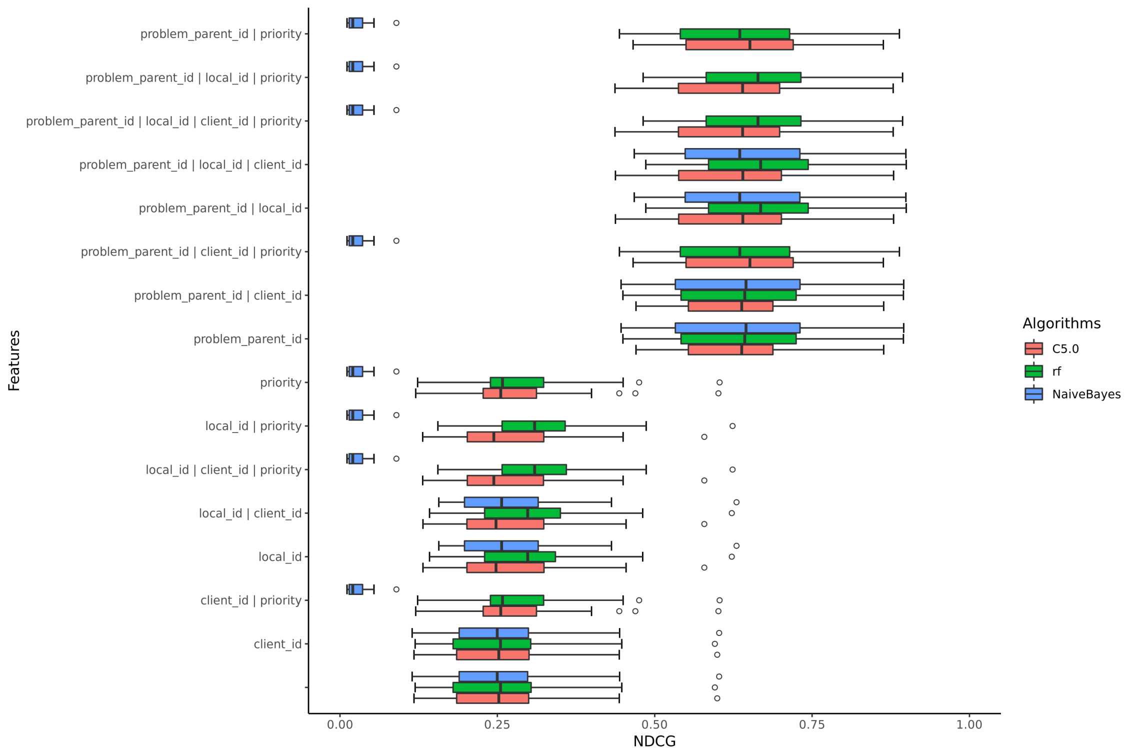


Figure B.7: RPE Sample 2 - Complete NDCG for Target problem_id

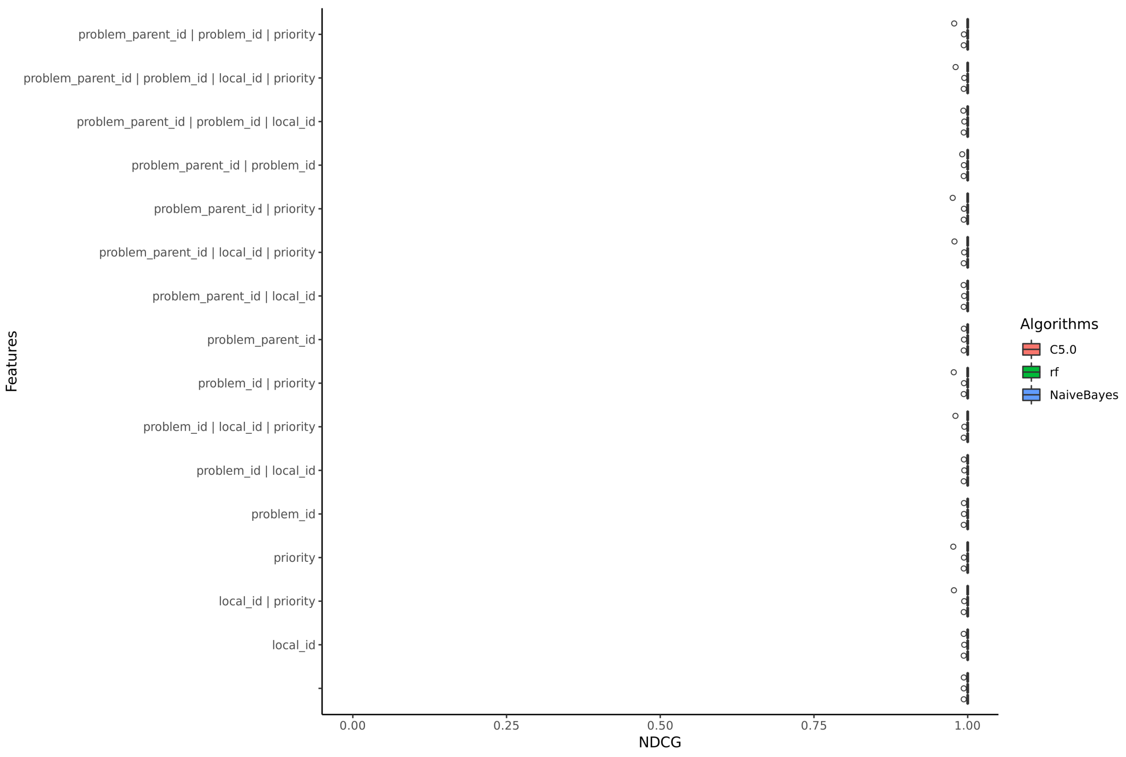


Figure B.8: RPE Sample 2 - Complete NDCG for Target client_id

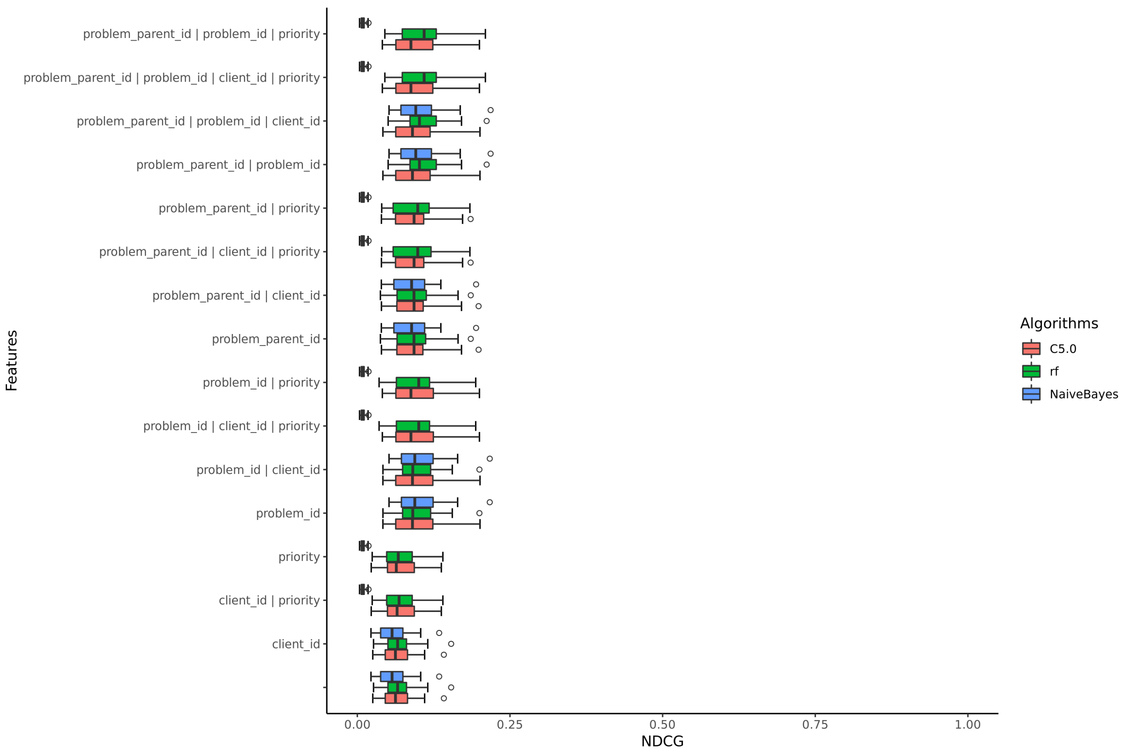


Figure B.9: RPE Sample 2 - Complete NDCG for Target local_id

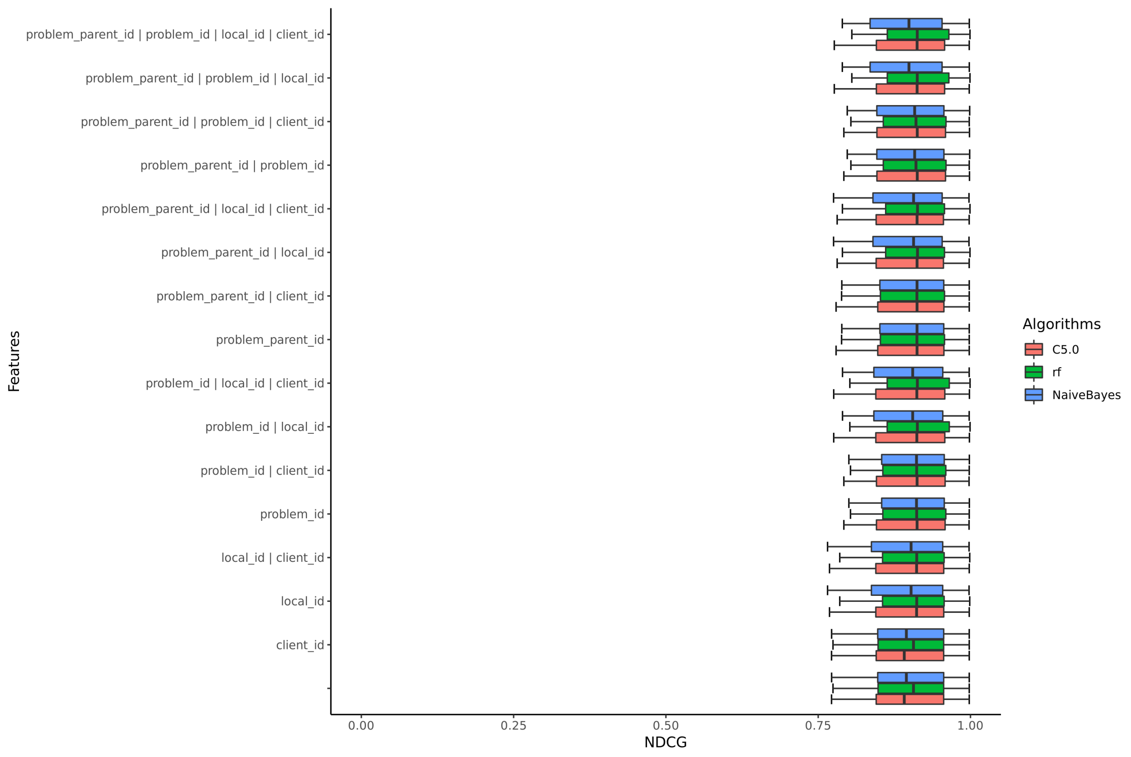


Figure B.10: RPE Sample 2 - Complete NDCG for Target priority

Appendix C

Experiment 3 Detailed NDCG Results

Same as Appendix A and Appendix B but for Experiment 3.

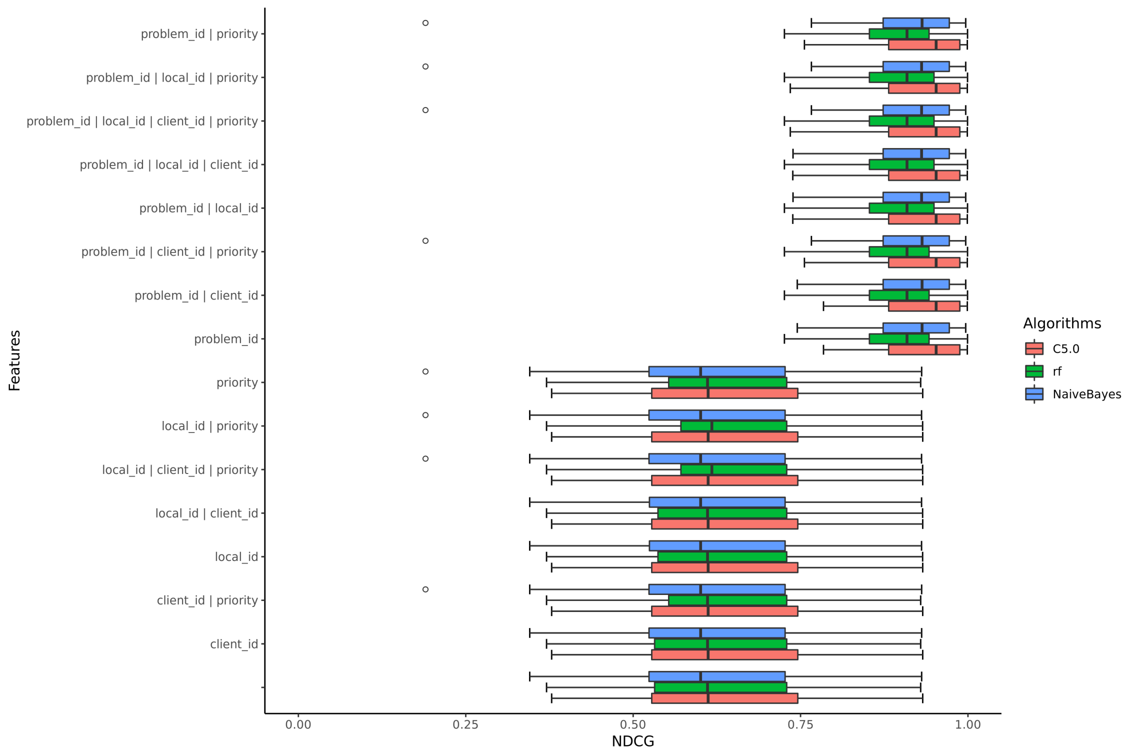


Figure C.1: Smart RPE Sample 1 - Complete NDCG for Target problem_parent_id

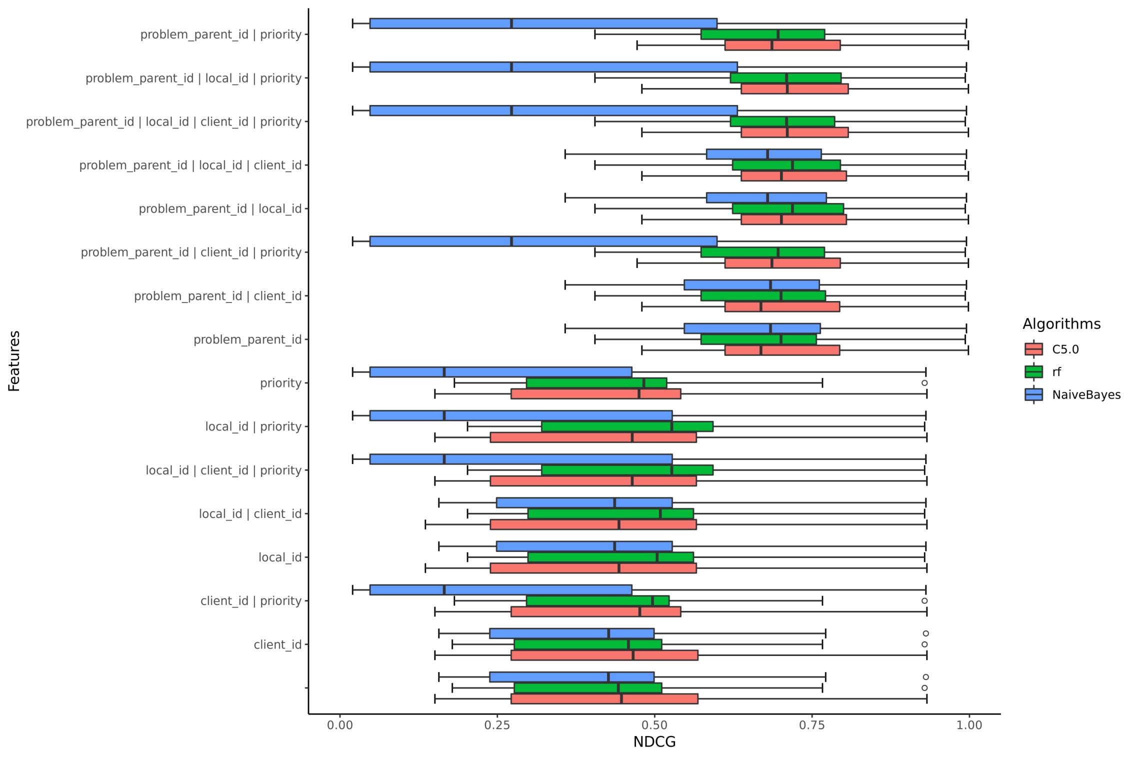


Figure C.2: Smart RPE Sample 1 - Complete NDCG for Target problem_id

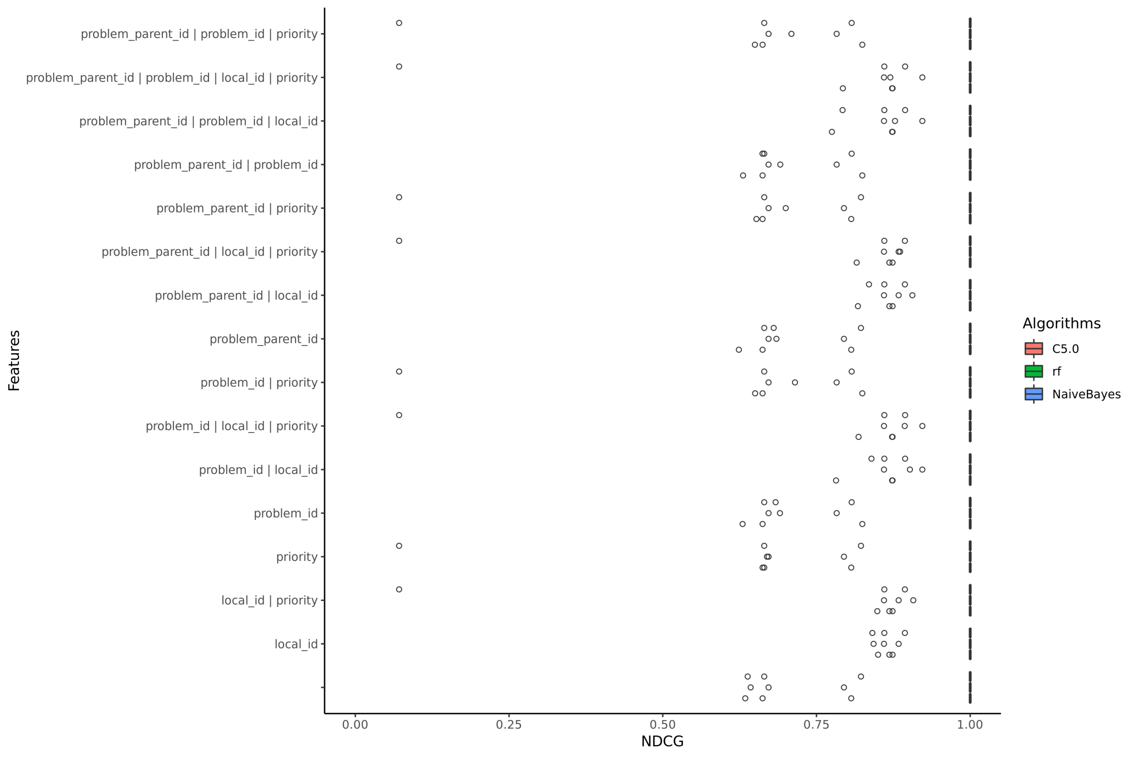


Figure C.3: Smart RPE Sample 1 - Complete NDCG for Target client_id

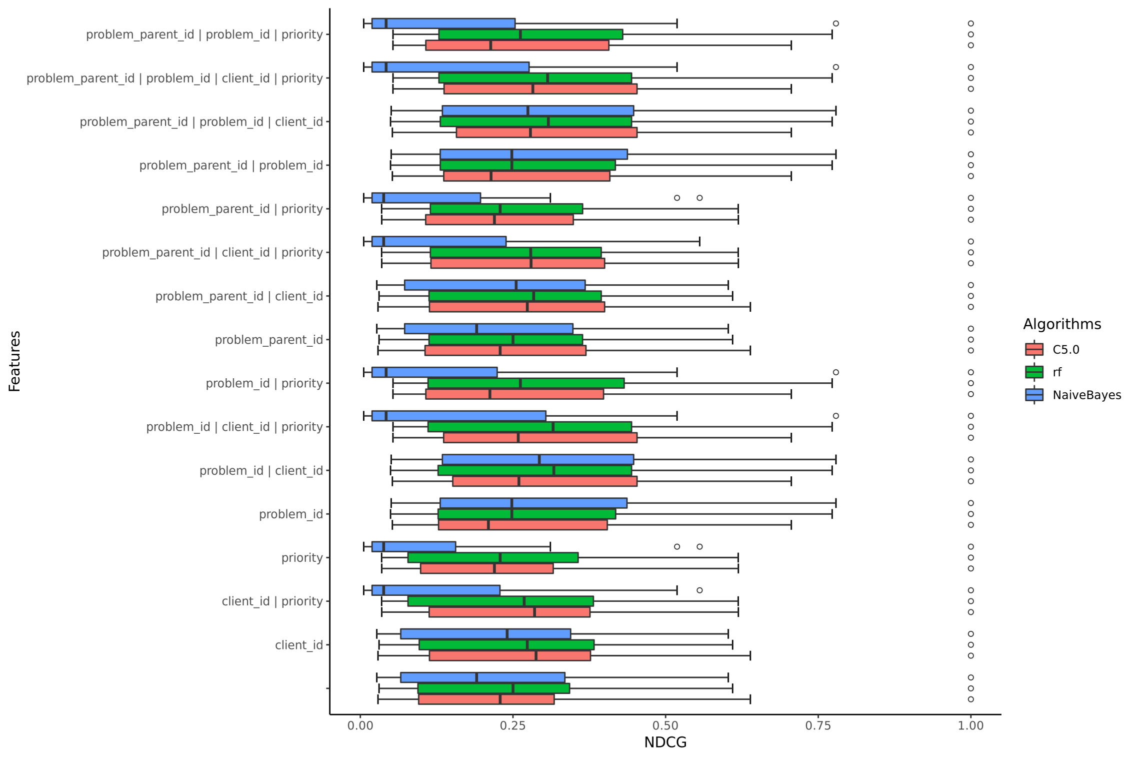


Figure C.4: Smart RPE Sample 1 - Complete NDCG for Target local_id

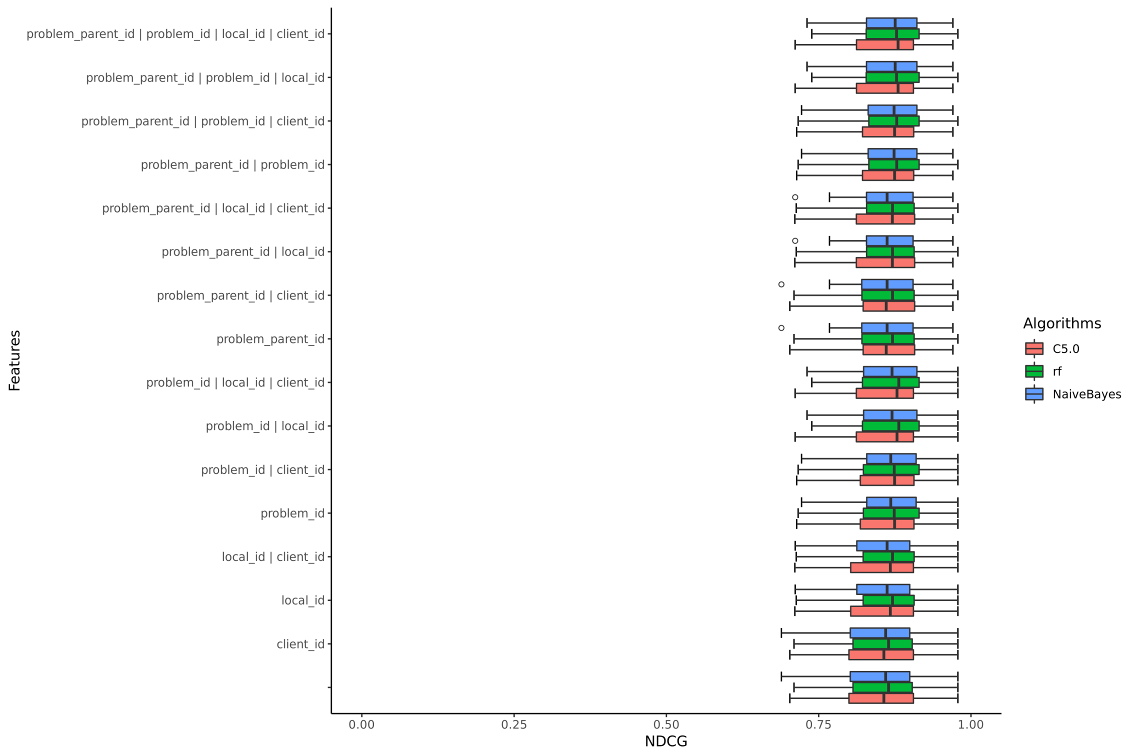


Figure C.5: Smart RPE Sample 1 - Complete NDCG for Target priority

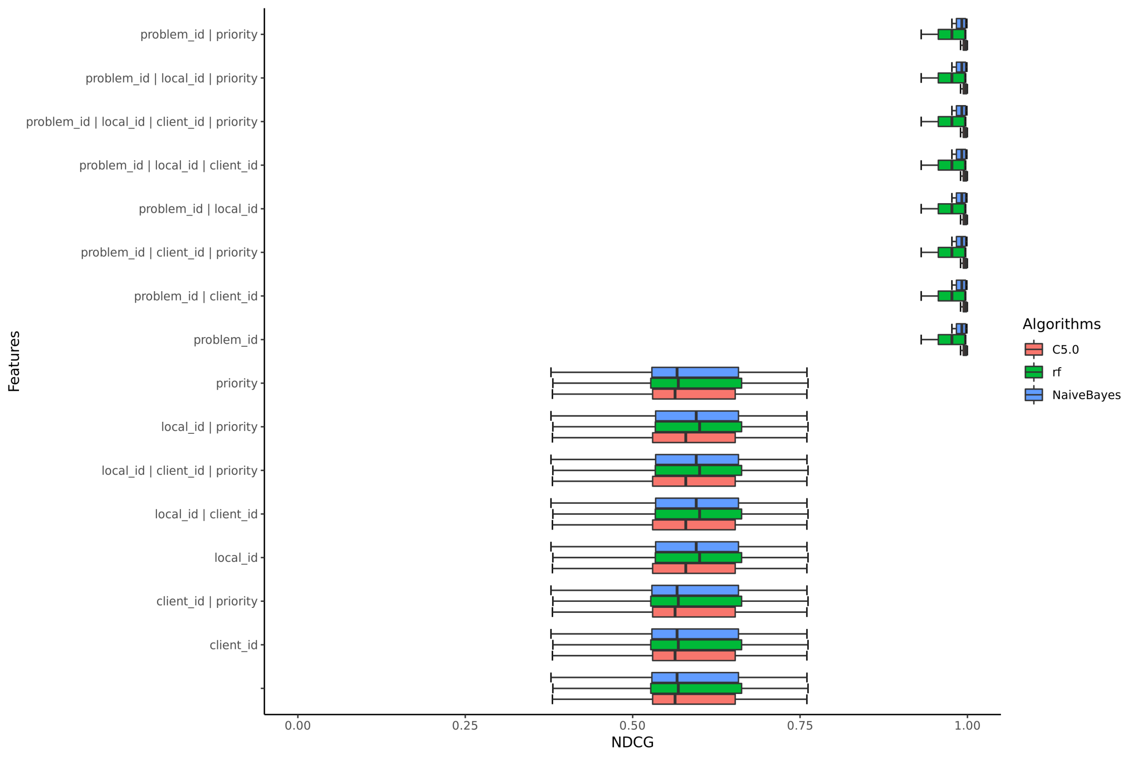


Figure C.6: Smart RPE Sample 2 - Complete NDCG for Target problem_parent_id

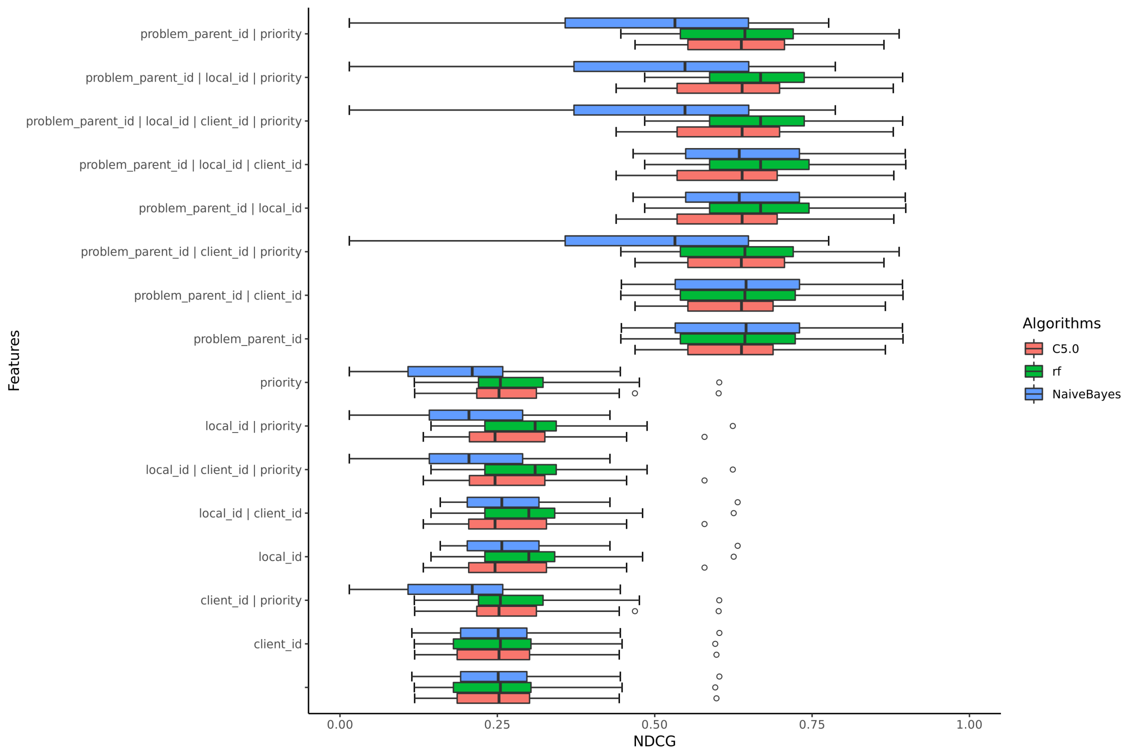


Figure C.7: Smart RPE Sample 2 - Complete NDCG for Target problem_id

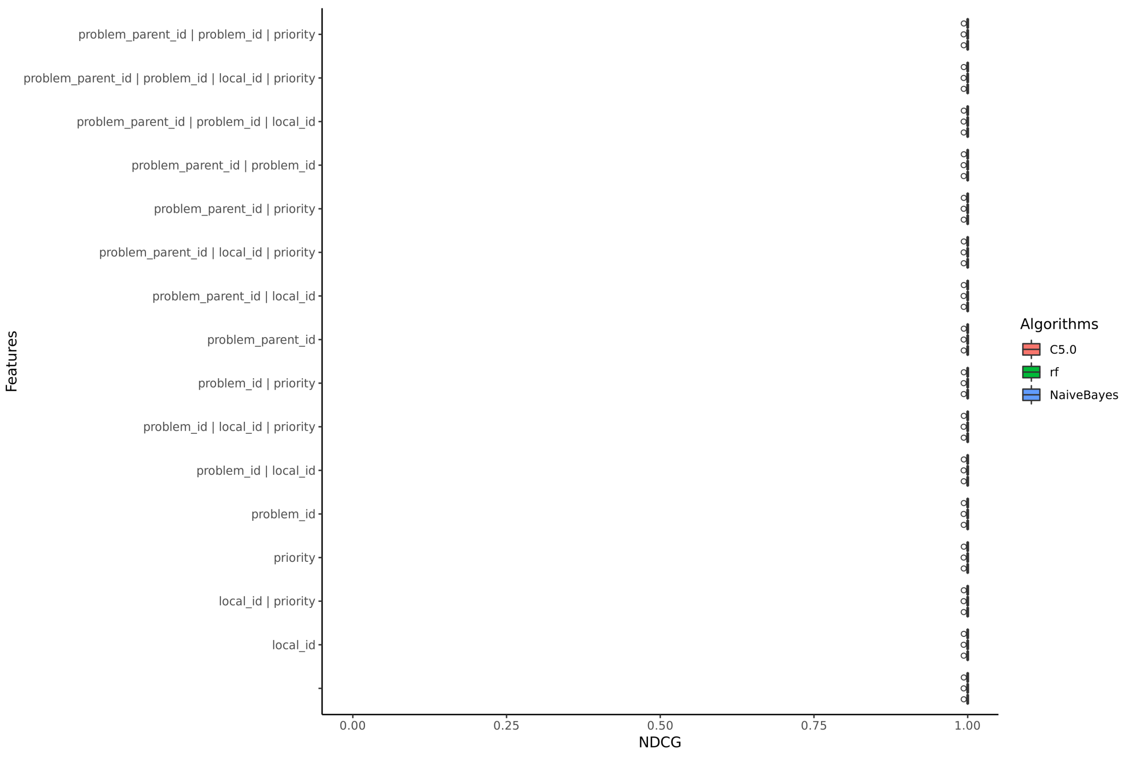


Figure C.8: Smart RPE Sample 2 - Complete NDCG for Target client_id

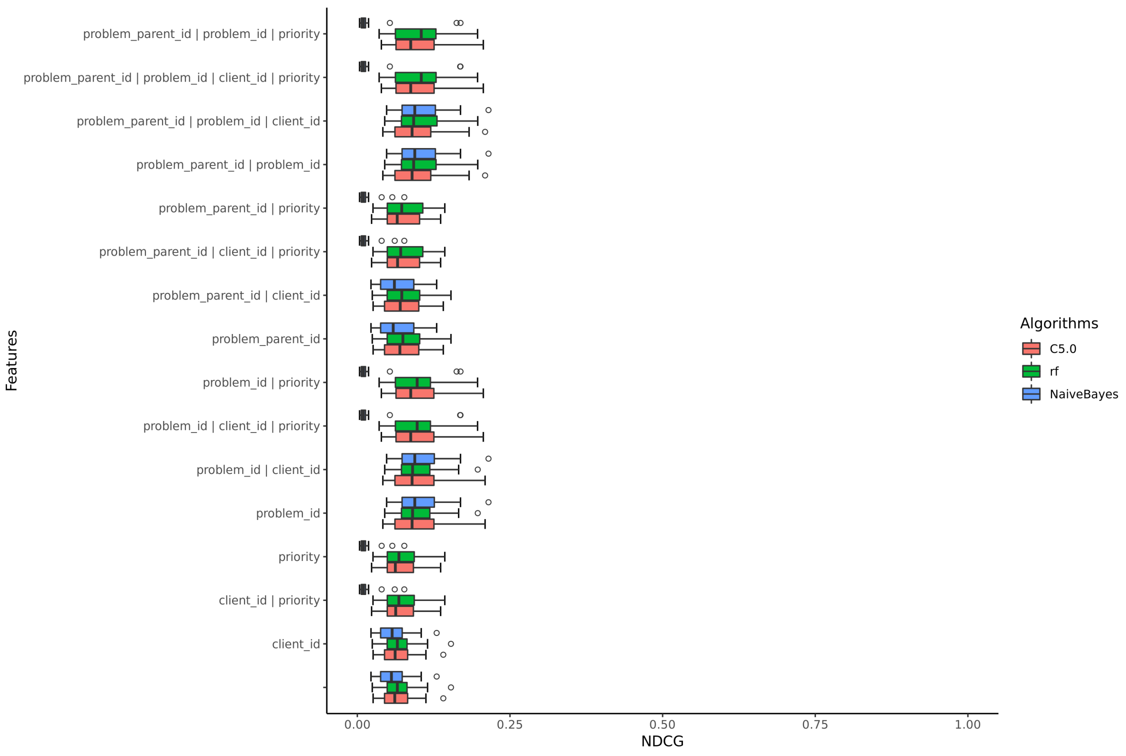


Figure C.9: Smart RPE Sample 2 - Complete NDCG for Target local_id

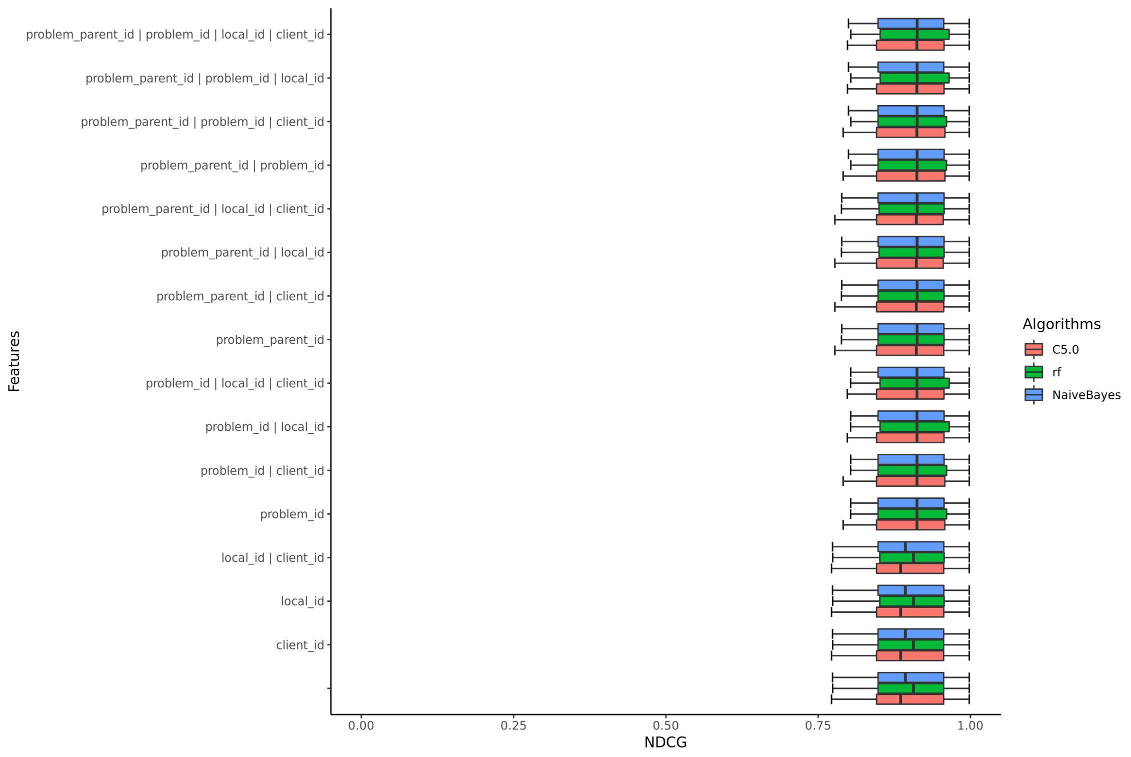


Figure C.10: Smart RPE Sample 2 - Complete NDCG for Target priority

References

Charu C Aggarwal. *Data mining: the textbook*. Springer, 2015.

Alan Agresti. *Analysis of ordinal categorical data*, volume 656. John Wiley & Sons, 2010.

Artur Aiguzhinov, Carlos Soares, and Ana Paula Serra. A similarity-based adaptation of naive bayes for label ranking: Application to the metalearning problem of algorithm recommendation. In *International Conference on Discovery Science*, pages 16–26. Springer, 2010.

Alnur Ali and Chris Meek. Predictive models of form filling. 2009.

Javier A Bargas-Avila, O Brenzikofer, SP Roth, AN Tuch, S Orsini, and K Opwis. Simple but crucial user interfaces in the world wide web: introducing 20 guidelines for usable web form design. In *User interfaces*. InTech, 2010.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

Pavel B Brazdil, Carlos Soares, and Joaquim Pinto Da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, 2003.

Klaus Brinker and Eyke Hüllermeier. Case-based multilabel ranking. In *IJCAI*, pages 702–707, 2007.

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96. ACM, 2005.

Weiwei Cheng and Eyke Hüllermeier. Probability estimation for multi-class classification based on label ranking. In *Proceedings of the 2012th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part II*, ECMLPKDD'12, pages 83–98, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-33485-6. doi: 10.1007/978-3-642-33486-3_6. URL https://doi.org/10.1007/978-3-642-33486-3_6.

Weiwei Cheng, Jens Hühn, and Eyke Hüllermeier. Decision tree and instance-based learning for label ranking. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 161–168. ACM, 2009.

Cláudio Rebelo de Sá, Carlos Soares, Arno Knobbe, and Paulo Cortez. Label ranking forests. *Expert Systems*, 34(1):e12166, 2017.

Ofer Dekel, Yoram Singer, and Christopher D Manning. Log-linear models for label ranking. In *Advances in neural information processing systems*, pages 497–504, 2004.

Peter A. Flach. Putting things in order: On the fundamental role of ranking in classification and probability estimation. In *Proceedings of the 18th European Conference on Machine Learning*, ECML '07, pages 2–3, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-74957-8. doi: 10.1007/978-3-540-74958-5_2. URL http://dx.doi.org/10.1007/978-3-540-74958-5_2.

Johannes Fürnkranz and Eyke Hüllermeier. Preference learning. In *Encyclopedia of Machine Learning*, pages 789–795. Springer, 2011.

Johannes Fürnkranz, Eyke Hüllermeier, Eneldo Loza Mencía, and Klaus Brinker. Multilabel classification via calibrated label ranking. *Machine learning*, 73(2):133–153, 2008.

Eyke Hüllermeier, Johannes Fürnkranz, Weiwei Cheng, and Klaus Brinker. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172(16-17):1897–1916, 2008.

Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

Kalervo Järvelin and Jaana Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 41–48. ACM, 2000.

Toshihiro Kamishima, Hideto Kazawa, and Shotaro Akaho. A survey and empirical comparison of object ranking methods. In *Preference learning*, pages 181–201. Springer, 2010.

Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007.

Han Liang and Yuhong Yan. Improve decision trees for probability-based ranking by lazy learners. In *Tools with Artificial Intelligence, 2006. ICTAI'06. 18th IEEE International Conference on*, pages 427–435. IEEE, 2006.

James D Malley, Jochen Kruppa, Abhijit Dasgupta, Karen G Malley, and Andreas Ziegler. Probability machines. *Methods of Information in Medicine*, 51(01):74–81, 2012.

Frank McSherry and Marc Najork. Computing information retrieval performance measures efficiently in the presence of tied scores. In *Proceedings of the IR Research, 30th European Conference on Advances in Information Retrieval*, ECIR'08, pages 414–421, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-78645-7, 978-3-540-78645-0. URL http://dl.acm.org/citation.cfm?id=1793274.1793325.

Ryszard S Michalski, Jaime G Carbonell, and Tom M Mitchell. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

Shuzi Niu, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. Top-k learning to rank: labeling, ranking and evaluation. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 751–760. ACM, 2012.

Matthew A Olson and Abraham J Wyner. Making sense of random forest probabilities: a kernel perspective. *arXiv preprint arXiv:1812.05792*, 2018.

Foster Provost and Pedro Domingos. Tree induction for probability-based ranking. *Machine learning*, 52(3):199–215, 2003.

Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine learning*, 85(3):333, 2011.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

Francesco Ricci, Lior Rokach, and Bracha Shapira. Recommender systems: introduction and challenges. In *Recommender systems handbook*, pages 1–34. Springer, 2015.

Robert E Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine learning*, 39(2-3):135–168, 2000.

Mirjam Seckler, Alexandre N Tuch, Klaus Opwis, and Javier A Bargas-Avila. User-friendly locations of error messages in web forms: Put them on the right side of the erroneous input field. *Interacting with Computers*, 24(3):107–118, 2012.

Shai Shalev-Shwartz and Yoram Singer. Efficient learning of label ranking by soft projections onto polyhedra. *Journal of Machine Learning Research*, 7(Jul):1567–1599, 2006.

Bayan Abu Shawar and Eric Atwell. Chatbots: are they really useful? In *Ldv Forum*, volume 22, pages 29–49, 2007.

Charles Spearman. The proof and measurement of association between two things. *The American journal of psychology*, 15(1):72–101, 1904.

Elia Van Wolputte, Evgeniya Korneva, and Hendrik Blockeel. Mercs: multi-directional ensembles of regression and classification trees. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Shankar Vembu and Thomas Gärtner. Label ranking algorithms: A survey. In *Preference learning*, pages 45–64. Springer, 2010.

Luke Wroblewski. *Web Form Design: Filling in the Blanks*. Rosenfeld Media, Brooklyn, New York, first edition, 2008. ISBN 1933820241, 9781933820248.

Ting-Fan Wu, Chih-Jen Lin, and Ruby C Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5(Aug):975–1005, 2004.

Bianca Zadrozny and Charles Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 609–616, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1. URL http://dl.acm.org/citation.cfm?id=645530.655658.

Yangming Zhou, Yangguang Liu, Jiangang Yang, Xiaoqi He, and Liangliang Liu. A taxonomy of label ranking algorithms. *JCP*, 9(3):557–565, 2014.