Received <day> <Month>, <year>; | Revised <day> <Month>, <year>; | Accepted <day> <Month>, <year>

DOI: xxx/xxxx

ARTICLE TYPE

An empirical study of learning-to-rank for spare parts consumption in the repair process

Edson Duarte | Daniel de Haro Moraes | Lucas Leonardo Padula

Venturus Innovation Center, São Paulo, Brazil

Correspondence
Edson Duarte, Venturus Innovation Center, Campinas, São Paulo, Brazil. Email: edson.duarte@venturus.org.br

Funding Information
This research was supported by the Ingenico do Brasil LTDA

Abstract

The repair process of devices is an important part of the business of many original equipment manufacturers. The consumption of spare parts, during the repair process, is driven by the defects found during inspection of the devices, and these parts are a big part of the costs in the repair process. In previous work we proposed a data-driven method for Supply Chain Control Tower solutions to provide support for the automatic check of spare parts consumption in the repair process. In this paper, we continue our investigation of a multi-label classification problem and explore alternatives in the learning-to-rank approach, where we simulate the passage of time using more data while training and comparing hundreds of Machine Learning models to provide an automatic check in the consumption of spare parts. We investigate the effects of different train set sizes, retraining intervals, models and hyper-parameter search using Bayesian Optimization. The results show that we were able to improve the trained models and achieve a higher mean NDCG@20 score of 86% when ranking the expected parts. Focusing on the most recent data, we achieve a NDCG@20 score of 90%, while obtaining a ratio of marked parts of just 4% of the consumed parts for use in alert generation.

KEYWORDS:
supply chain, spare parts, repair process, alert generation, information retrieval, machine learning

1 | INTRODUCTION

The repair process of digital devices used by operators in the field is an important part and, sometimes, the main component of the operation of many original equipment manufacturers (OEMs). A device is sent for repair for many reasons, some are non-functional like scratches, but most are functional problems that prevent the device from working properly. A common practice is that the repair process is executed by a partner repair shop of the OEM that sells these devices and their after-sales maintenance.

In such cases, it is difficult to validate if the defects reported by the operator and validated by the repair shop are sufficient to explain the spare parts that are consumed in the repair process. Spare parts are an expensive resource, provided by the OEM, and considered a crucial resource to keep track of. The unrestricted consumption of spare parts may generate out-of-stock and downtime situations that increase the cost and the duration of the repair services, and both may cause the perception of a poor quality of service.

Usually, Supply Chain Control Tower (SCCT) solutions are employed to ensure an end-to-end view of the entire process but popular tools still lack the support required to validate spare parts consumption. To help check this consumption, the industry is applying Data Science and Machine Learning (ML) to identify possible issues and help improve the repair process.

In this work, we continue our investigation of a learning-to-rank approach to automatically check the consumption of spare parts in the repair process of digital devices. Using data from an SCCT about each repair service, their identified defects and consumed parts, we investigate the effects of different train set sizes, retraining intervals, models and hyper-parameter search using Bayesian Optimization, while training hundreds of ML models.

We have designed two rounds of experiments, in the first round we select the train set size and retraining interval that improve most the results, and in the second round we perform Output Fusion and hyper-parameter search. With the trained models, we generate predictions that tell us which parts should be used in each repair service. We collect these predictions to generate a ranked list of suggestions of parts to be replaced.

We were able to improve our learning-to-rank approach and that the trained models now obtain a mean NDCG@20 score of 86% overall when suggesting lists with 20 parts. Next, we focus on the last 8 weeks of data to obtain a mean NDCG@20 of 90% and we use the models predictions to compare the suggested parts against the list of actual replaced parts. The replaced parts that are not in the suggestions are marked and the revised approach reduced the ratio of marked parts to 4%. And, in summary, our main contribution is a revised and improved data-driven method for automatically identifying deviations in the consumption of spare parts in the repair process, from the perspective of the OEMs. These deviations can be seen as additional information to be integrated and used in SCCTs to enrich alert generation.

The remaining of this paper is organized as follows. Section 2 presents the related work. Section 3 presents details about the data used. Section 4 details the methodology used in this study. Section 5 shows discussion about the results. Section 6 details our conclusions.

2 | RELATED WORK

2.1 | Spare Parts Supply Chain

The supply chain of a company is composed of all the necessary systems and facilities required to provide a product or service to the end-users. The spare parts supply chain is the subset of the entire supply chain that is necessary to provide spare parts for the repair service sold by the OEMs.

A Supply Chain Control Tower (SCCT), or a Service Control Tower (SCT), is an umbrella term for emerging solutions that bring an end-to-end view of the supply chain by acting as a central hub that integrates tools and processes to drive business outcomes. An SCCT solution is a complex set of systems and processes, and an important component of any SCCT is alert generation.

Following the framework proposed by Topan et. al.[1], an alert, in this context, is any form of notification that is generated with the intent of triggering any one of their identified interventions into the supply chain processes. Usually, alert generation is based on ad-hoc rules or traditional forecasting techniques, this causes a high number of alerts being generated while a low percentage of them are identified as high priority. And currently, SCCT solutions generate too many alerts and rely in the need for finding arbitrary thresholds.

Specifically, Supply Chain Event Management (SCEM) focuses in real-time monitoring using statistical process control to generate alerts. Still, there is not a common solution for alert generation and, the literature does not address how trends and deviations are identified, how alerts are usually generated, or how to determine interventions.

Also, the identified interventions are divided between reactive and proactive interventions but they are all related to reducing downtime risk or, when a downtime is not avoided, to reduce recovery times. An analysis found out that some interventions are more cost effective than others[2]. Another analysis, combining event-based and periodic interventions, found out that joint interventions can considerably reduce total downtime[3]. Still, there are no proactive interventions related to refining the repair process or improving repair shop efficacy in the consumption of spare parts.

The repair service of capital goods is usually placed as part of after-sales services[4]. When evaluating repair services, top-down frameworks view them as another component in a high-level performance tree[5]. While bottom-up frameworks view the repair services as a subjective component, usually evaluated by customers[6].

In our previous work, we proposed a data-driven method to check spare parts consumption in the repair process that generates additional information that can be effectively integrated into the alert generation processes of an SCCT[7]. This was an important result because improving spare parts consumption helps to ensure the availability of spare parts, which in turn greatly reduces downtime. And, as reported in a recent literature review[8], very few works take the perspective of OEMs.

2.2 | Information Retrieval

Information Retrieval (IR) is defined as dealing with methods that retrieve subsets from data based on the needs of users and it has become popular in the 21st century[9].

The most common way of organizing the results in IR systems is by ranking them. In this context, learning-to-rank are methods that try to learn the ordering of the results[10]. In pointwise methods, when given a document and query pair together with a score or a label, the ranking problem is similar to a regression or a classification problem, respectively.

For classification problems, it is common to use the Logistic Regression or Maximum-Entropy method. Although, when used for learning-to-rank problems, other methods like Support Vector Machine (SVM) seem to perform better[11]. The Logistic Regression is a well known method for classification, which models its outcomes as probabilities using the logistic function. It is known for being a well calibrated classifier that outputs confidence values for the available labels[1]. While SVMs are known for being very powerful models, they are effective in high dimensional spaces and versatile due to being able to use different kernel functions, they are also very slow, being a quadratic programming problem, because their compute and memory requirements increase very rapidly with the size of the train set.

Multi-label classification is a learning-to-rank extension where each document is assigned to a set of labels and some methods, like K-Nearest Neighbors and Decision Trees, already have support for multi-label classification[2]. Still, when there is no support for multi-label classification, the most common problem transformation method is to learn a binary classifier for each one of the different labels in the set[12].

Ensemble methods are a family of techniques for combining predictions with the goal of improving generalization and robustness. Random Forest is a method were many Decision Trees are trained in subsets of the training data and their predictions are averaged to generate the final predictions. Adaptive Boosting (AdaBoost) is a classic boosting method in which many copies of another base method are trained in sequence and subsequent copies are weighted to focus on the errors of previous copies. XGBoost[13] is another ensemble method that incorporated many theoretical advances under a single framework, it trains many Decision Trees but introduces improvements over the classic boosting strategy, with regularization and better purging, randomization, shrinking and splitting of trees[14].

When working with conceptually different methods, we can combine them by a process called Output Fusion, one of the main approaches for doing this is the weighting or voting method, where we simply apply a weighted average to the outputs of the base methods, the weights can be selected in several different ways but the base methods can also have equal weights[14].

hyper-parameter search is the process of optimizing a ML model over a configuration space. A ML method exposes a set of knobs that can be changed to drive the learning phase of the algorithm. Each of these knobs is a hyper-parameter, a value that configures the learning phase, it is defined prior to the execution of the learning phase. There are several ways to choose hyper-parameters, and Grid and Random searches are the simplest and most used ones. The Grid Search exhaustively searches a list of sets of hyper-parameters, but the list needs to be manually defined and grows exponentially. With Random Search, we define ranges of values to be searched for each hyper-parameter, but because it is random, the algorithm may never find a good set of hyper-parameters[15].

For larger models, like the XGBClassifier with dozens of hyper-parameters, better approaches are necessary for hyper-parameter search. Bayesian Optimization is a generic sequential strategy for expensive functions that can be used for hyper-parameter search because each model is an expensive algorithm that needs to be executed and evaluated. Bayesian Optimization builds a prior of the ML model using Gaussian Processes, trains the model with a set of hyper-parameters, updates the posterior and uses it to build an acquisition function that generates a new set of hyper-parameters to train the model again. And this process is repeated until a budget is reached[16].

During hyper-parameter search, the process of cross-validation is employed for each set of hyper-parameters being evaluated. In cross-validation, the train set is divided into several folds and a training round is performed for each fold as the test set, and in this case, the test set is usually called the validation set. After all folds have been used once as the validation set, all scores are collected to generate a final score for the hyper-parameter set. There are several ways to divide the train set, the most common being the K-Fold, which simply divides the train set into k folds of similar sizes performing, or not, a prior shuffle of the data. Still, most schemes to generate the folds assume that the data is independent and identically distributed (iid). When this is not the case, which is a common situation with production systems, the most common way to divide the train set is by sorting the data temporally and dividing into blocks that respect this time axis, this is usually called time blocking, time series split or walk-forward[17].

3 | DATA

As described in Section 1, the repair process is usually performed by a partner repair shop and, to have a coherent SCCT solution, these partners need to report the status of each repair service in a timely manner. A single repair service may have a duration from some minutes to several days depending on many factors, but at each step in the repair process an update is generated, usually several times during a single day.

In Figure 1 , we show components of an SCCT solution, mainly how the OEM and repair shops interact to generate and consume data from the SCCT and how our solution can be integrated to provide automatic consumption checks information back into the SCCT to be used in alert generation by the OEM.

A service order registers all the relevant information for a single repair service and may contain hundreds of fields, e.g. the identifier of a client, the start date of the service, the type of product that is being repaired, and many others. The relevant fields for our study are those specifying the defects validated by a trained technician and the parts replaced by another, or possibly the same, trained technician. A single device may present, after inspection, several defects that will require the replacement of several parts. Usually, a single defect implies a single part being replaced but this is not a strict rule and a defect may rarely have no parts or two or more parts being replaced. Also, a device may present the same defect multiple times and multiple parts of the same type may be replaced.

Another important aspect and, perhaps, a more difficult issue is that the relationship between defects and parts is not a one-to-one relationship, meaning that a single defect can be addressed by different parts, that may not be of the same type, and that a single part can address different defects. This is common, because a digital device is a really complex piece of machinery that is built from parts from several providers, and a single part can have many similar parts that come from different providers, these providers may have used different materials in the production of similar parts and all these parts have vastly different life expectancy.

We have collected historical data from an emerging SCCT solution used by a global leader in their sector, the data covers a period close to 20 months, more than 1.8 million repair services, and 23 types of digital devices with 225 types of unique defects and 722 types of unique spare parts.

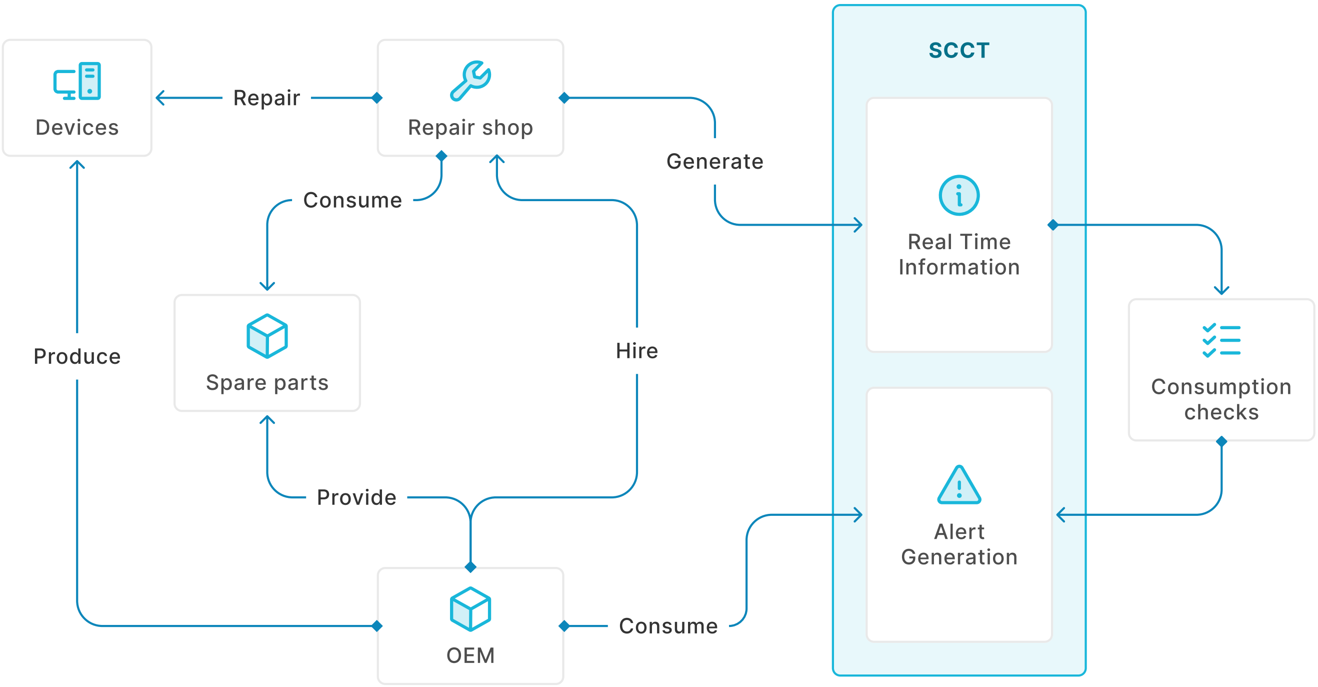

FIGURE 1 A diagram highlighting SCCT components. To the right, our solution to check spare parts consumption uses real time information and provides data for alert generation.

4 | METHODOLOGY

In this section we show the methodology used to evaluate the improvements to the data-driven approach that we have previously introduced to automatically check the consumption of spare parts for each service order. First we describe the overall data setup and its improvements. Next, we describe the main Machine Learning models employed and the auxiliary methods used for Output Fusion and hyper-parameter search. After that, we show the metrics used for evaluation of the different models and how the consumed parts are marked. Last, we describe the experiments that were performed.

4.1 | Setup

An important aspect of any solution that will be used in a production setting is that the solution will be in continuous usage. Additionally, when an ML algorithm is employed, this means that a given model must be trained at a given point in time and its use will be spread during a determined time frame that can be from days to months, or even years.

Because of this, it is usual to frame the setup in terms of a time series with the variables of interest evolving as the time advances. This framing helps reducing problems of data and prediction drift that can happen when future data differs from past data, a common scenario for most ongoing systems. This difference between past and future is usually identified using statistical tests.

Since our previous work, we have totally revised how to handle the data described in Section 3 to support our current investigations. We managed to decouple the number of unique defects and unique parts between train and test sets. This allowed us to have more data and to change the size of the train set after the snapshots have been generated.

We have now divided the data into 80 sequential snapshots, with each snapshot representing the passage of only 1 week from a previous snapshot. A given snapshot is composed of a train set and a test set. The train set represents the past and is used for training the ML models, while the test set represents the future and is used for evaluating the models. Further, the test set of a previous snapshot is merged into the train set of the next immediate snapshot. And all the test sets have no overlapping data.

In Figure 2 , we show the total number of samples of each set for all the 80 snapshots, one snapshot per week. As can be seen, in each snapshot the test set has a variable number of samples and since a previous test set is incorporated into the next train set, this would cause the train set to keep growing but, due mainly to memory restrictions, we keep purging old data as we move forward.

Two important dimensions that we have investigated are the impacts of the size of the train set and the interval between retraining the models. We have selected the 20k train set size and the 4 weeks retraining interval to be comparable to the results presented in our last work. Additionally, we have increased the train set size to 60k, 100k and 140k samples and we have

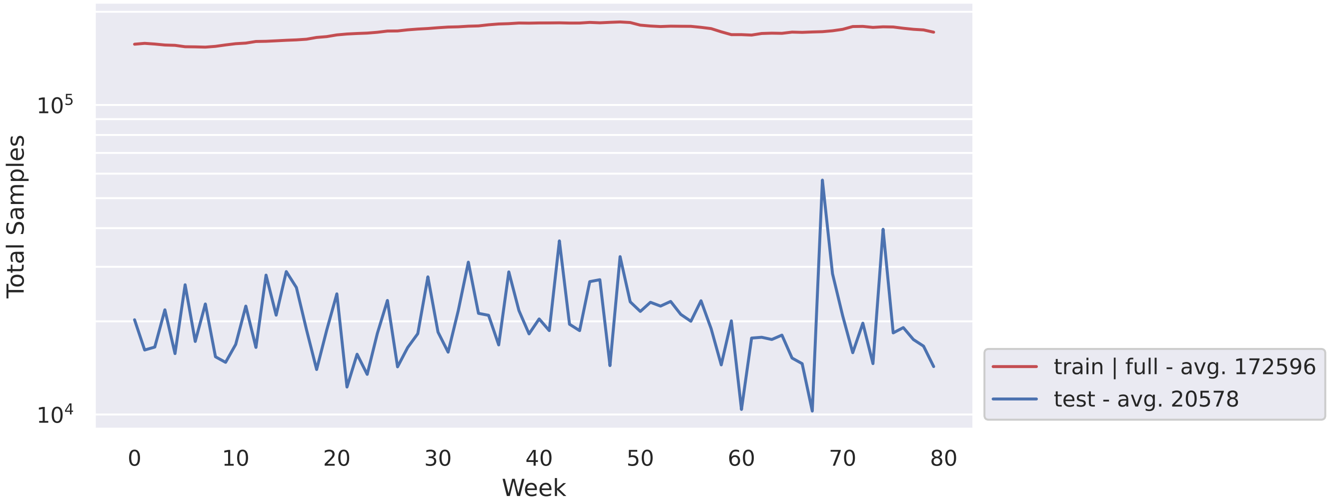

FIGURE 2 A time series plot showing the total number of samples for the train and test sets for each of the 80 weeks.

decreased the retraining interval to 1 and 2 weeks. This gives us a total of 12 combinations of training schemes that we will delve into in the following sections.

In Figure 3 , we show the number of unique defect types of each set for all the snapshots. And in Figure 4 , we show the number of unique part types of each set for all the snapshots. In both cases, we also show lines for the different train set sizes to show how much of the diversity of types are covered by each selected size. As the size of the train sets changes, the number of types changes accordingly and with more data the number of unique types increases. Also, we see that as time advances, the number of types in each snapshot changes. This is due to many factors, like seasonal defects because of more rain in a given season of the year, old devices being deprecated or new ones being released, or the low supply of one or more spare parts.

We used virtual machines running in Microsoft Azure, specifically several memory optimized second generation Ds-Series 14 with 16 vCPUs and 112GB of RAM memory, these were the biggest general purpose machines with a reasonable pricing

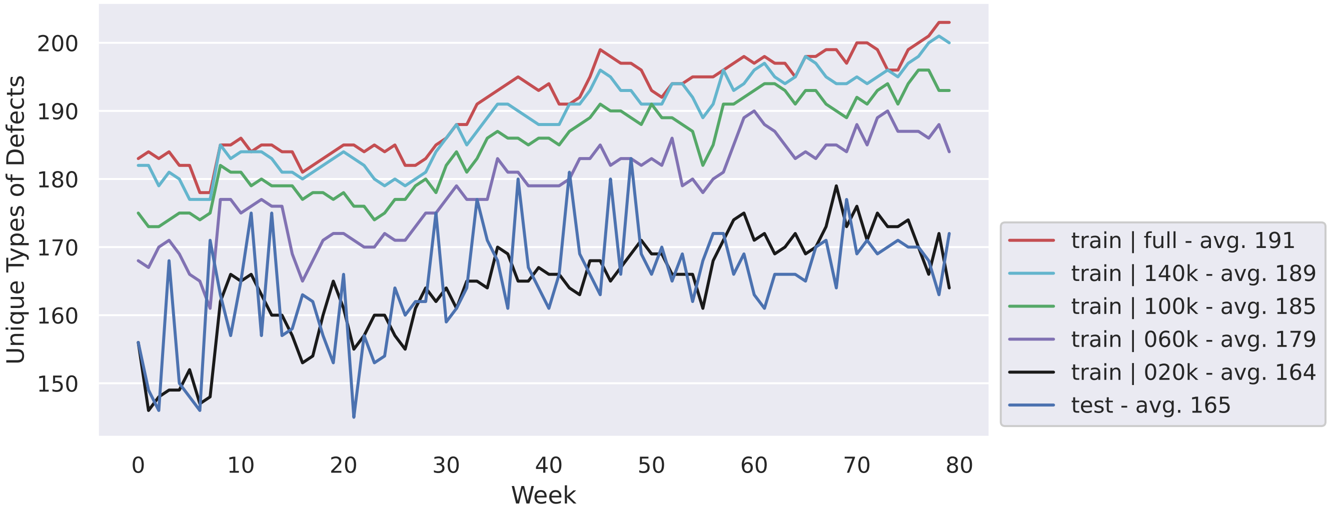

FIGURE 3 A time series plot showing the number of unique defect types for both the train and test sets for all the snapshots. The total number of unique types is 225.

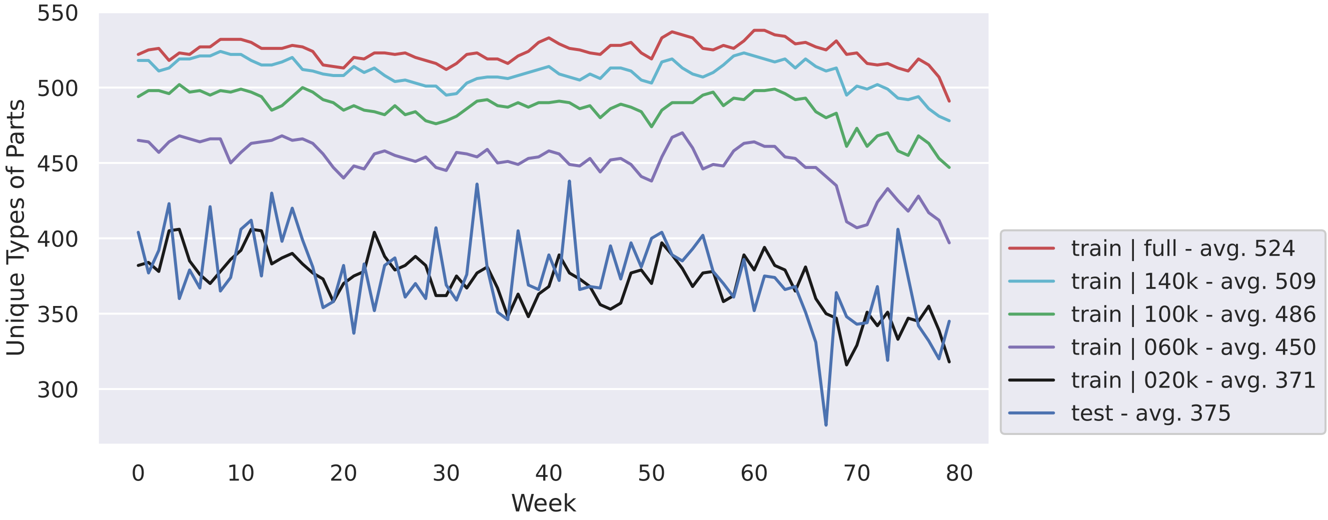

FIGURE 4 A time series plot showing the number of part types for both the train and test sets for all the snapshots. The total number of unique types is 722.

available at the time. All the data was preprocessed and stored before selecting the train set size and the retraining interval to be used.

4.2 | Models

Following the discussion from Section 2.2, for a multi-label classification problem, we have two options: training a model with native support for multi-label or training a model for each label. Specifically for our problem, in the first option we would train hundreds of models, and in the second option we would train thousands of models. In this work, we have employed the use of several algorithms readily available in the scikit-learn Python package[18]. For training the models with simple pipelines and parallel execution the joblib Python package was used[3].

From the methods described previously, we maintained the use of the Logistic Regression algorithm (LogisticRegression[4]), as the baseline from our previous work. We added the XGBoost algorithm (XGBClassifier[5]) for comparison against a more powerful method. We also tried using Random Forest (RandomForestClassifer[6]), AdaBoost (AdaBoostClassifier[7]) and SVM (SVC[8]) as alternative methods without success. The RandomForestClassifier consumed all the RAM memory of the virtual machines and crashed. And both AdaBoostClassifier and SVC took too much time, our estimates are that they take from 10 to 20 times longer than the LogisticRegression model, measured using the smaller 20k train set.

To easily combine the training and evaluation of hundreds of models, we used a wrapper, or meta-classifier (MultiOutputClassifier[9]). Also, as we have decoupled the number of unique parts between train and test sets, we can have a situation in which a spare part is never or always used. Both during training or evaluation time, causing the existence of only one class during training, because of this, we have designed an additional simple wrapper that checks the number of classes during training and that generates predictions using only one class in the case that a single class was seen during the training.

We have also employed a meta-classifier for Output Fusion (VotingClassifier[10]) to investigate the performance of combining the LogisticRegression and XGBClassifier models. And finally, we have employed a wrapper model to easily perform hyper-parameter search using the Bayesian Optimization framework (BayesSearchCV[11]) from the scikit-optimize Python package[19] and for cross-validation we have used the time series split scheme (TimeSeriesSplit[12]).

4.3 | Metrics

An important part of a project employing ML algorithms is the evaluation of the models. To evaluate if a model may perform well when in use, we need a way to measure its performance. For this purpose, we employ the use of different evaluation metrics to assess model performance from different perspectives:

1. Accuracy score[13], in multi-label classification, computes subset accuracy, the set of predicted labels must match exactly the expected labels. It is a common metric for classification problems and is easily adaptable for multi-label classification, still it is a very difficult metric in our case due to the need of matching all the labels exactly. All the models need to match all the labels exactly in all the spare parts. That is a really tall order;

2. Coverage error[14] is a measure of how far is needed to go in the ranked scores to cover the expected labels. In our case, it gives us an idea of how long our lists should be to cover all the expected parts. And we use it as a proxy for selecting the parameter k for the next metric;

3. Normalized Discounted Cumulative Gain (NDCG) score[15] considers the top k ranked scores to count the expected labels that were predicted correctly. It is a common metric in IR and multi-label problems[20]. This metric sorts the expected labels by using the predictions, applies a logarithmic discount and, then, sums all the results. And it uses an ideal Discounted Cumulative Gain to normalize the values and keep the metric between 0 and 1.

Using a trained model and a data set, we generate predictions for the samples in the data set. In a classification problem, the data set contains the expected class for each sample, the ground-truth. Using the expected classes and the predicted classes, we are able to calculate the evaluation metrics. And calculating the metrics for both the train and test sets, we are able to investigate the learning and generalization capabilities of the trained model and infer if there is a problem of underfitting or overfitting.

4.4 | Ratio of Marked Parts

When evaluating models, generic metrics, like the Accuracy or the NDCG scores, are a good start but they measure the performance from the perspective of a common problem that may not be related to the actual problem at hand. Because of that, it is usual to design a custom metric, a measure of score or error, that is aligned with the problem and actually measures some kind of business objective directly from the models predictions. But achieving a good business metric is a complex process that requires the interaction of business managers, subject matter experts, and data scientists.

We have designed the Ratio of Marked Parts (RMP) as an initial and simple custom metric, it counts any consumed parts that do not appear in the top k ranked parts and divides them by the total number of all consumed parts. After having trained the ML models using both the identified defects and the consumed spare parts. We use their predictions to sort and rank the spare parts like a list of suggestions, where the parts with higher scores will appear at the top.

For each repair service, we generate a list of suggestions with k parts, using the same value used for the k parameter in NDCG. We compare the suggestions for a service against the actual consumed parts for that same service, if a consumed part does not show up in the list of suggestions, we count it. Repeating the process for all the services, we count all the consumed parts that do not appear at the top k. In the end, we divide the total count by the total number of consumed parts to obtain the RMP.

The RMP being a simple metric has problems, the most evident one is that it treats all consumed parts equally but we know that spare parts have different characteristics, and to address this would require additional data.

4.5 | Experiments

We have designed two rounds of experiments. In the first round, we explore the effects of the size of the train set and the retraining interval for both the LogisticRegression and XGBClassifier models, for a total of 24 runs. The values used for the train set sizes were 20k, 60k, 100k, and 140k. And the values used for the retraining interval were 1 week, 2 weeks and 4 weeks. With the results, we selected just the combination of train set size and retraining interval that best improves the performance of both models at the same time. In the second round, we explore the effects of Output Fusion and hyper-parameter search for both models using the selected values from the first round, for five additional runs. For Output Fusion, we employed the simple average, or voting, scheme to combine the results of the base models. And we have performed hyper-parameter search for each base model only separately.

In Table 1 , we list all the options used in the hyper-parameters search, each training round was allowed to evaluate 20 sets of hyper-parameters, a 4-fold time series cross-validation and the NDCG@20 metric were used for optimization. The LogisticRegression model has only a few hyper-parameters, we have chosen not to change the penalty and solver settings because they could generate unsupported combinations. The XGBClassifier has dozens of hyper-parameters but, since it is theoretically much more powerful, we have chosen to deal only with the hyper-parameters more close related to the problem of under and overfitting[16].

TABLE 1 The hyper-parameters included for search in the Bayesian Optimization framework.

Model	hyper-parameter	Type	Prior	Values
LogisticRegression	C	Real	log-uniform	$[10^{-6}, 10^{6}]$
	intercept_scaling	Real	log-uniform	$[10^{-2}, 10^{2}]$
	class_weight	Categorical	None	[balanced, None]
XGBClassifier	colsample_bytree	Categorical	None	[0.1, 0.3, 0.5, 0.75, 1.0]
	gamma	Real	uniform	[0, 1]
	learning_rate	Real	uniform	[0, 1]
	max_delta_step	Integer	uniform	[0, 10]
	max_depth	Integer	uniform	[1, 10]
	min_child_weight	Integer	uniform	[0, 3]
	scale_pos_weight	Integer	uniform	[1, 100]
	subsample	Categorical	None	[0.1, 0.3, 0.5, 0.75, 1.0]

5 | RESULTS

The results obtained in the first round of experiments are summarized in Table 2 . The table shows the results for both the LogisticRegression and XGBClassifier models and each of the train set sizes and retraining intervals. For each combination it shows values for both the train and test sets for the Accuracy score, the Coverage error and the NDCG@20 score, and also the total training time. For the test scores, all scores use all the 80 snapshots. For the training scores, the 1 week retraining interval scores use all the 80 snapshots, the 2 week retraining interval scores use just 40 snapshots, it skips 1 week after each training round, and the 4 weeks retraining interval scores use only 20 snapshots because it skips 3 weeks after each training round. The values for the metrics are the average and standard deviation over all the used snapshots, and the total training time is the sum over all the used snapshots, in hours. We note that there are missing rows for the 1 week retraining interval and both 100k and 140k train set sizes, we were unable to finish these experiments due to memory errors.

From the results, we can see that, overall in this first round of experiments, the 60k train set size and the 1 week retraining interval have presented the best improvements for both models, we will focus the analysis in these values since they are the ones that are used in the second round of experiments.

As explained in Section 4.3, accuracy is a very difficult metric for our context and the obtained scores are low with the best score in the test sets being 21% for the LogisticRegression and 33% for the XGBClassifier. Still the scores are high given the difficulty of the task, meaning that for the average 21k samples in each of the test sets (Figure 2) with an average of 450 consumed spare parts (Figure 4), the LogisticRegression models get correctly all the consumed spare parts for 1 in every 5 services and the XGBClassifier models get 1 in every 3 services, this amounts to more than 1.9 and 3.1 millions correct predictions per test set, respectively.

For the coverage errors, we got very close values for both models, with values for the train and test sets of 12 and 34 for the LogisticRegression and 8 and 31 for the XGBClassifier. From our analyses, 95% of the services have less than 10 parts replaced, less than 1% of the services have more than 20 parts replaced, and with such values obtained for the Coverage error, we decided to keep the value of 20 for the k parameter in the NDCG score from our last work.

Next, for the NDCG@20 scores, we obtained rates of 83% for the LogisticRegression and 86% for the XGBClassifier, both are really high values given the difficulty of the task. This NDCG@20 score is not a true probability, it is a ratio given an ideal score. Still, it means that given the scale set by the ideal score, the models are ranking the expected parts very high in the predictions.

TABLE 2 Results obtained in the first round of experiments, we show the average and standard deviation for all the combinations of models, retraining intervals, train set sizes and metrics for both the train and test sets. The total training time taken is shown in hours. The best values for each model are in bold face.

Model	Interval	Size	Accuracy		Coverage		NDCG@20		Time
			Train	Test	Train	Test	Train	Test	
LogisticRegression	1 week	020k	0.29 ± 0.05	**0.21 ± 0.04**	11.35 ± 2.28	44.42 ± 18.16	0.90 ± 0.03	0.82 ± 0.07	13.1
		060k	0.27 ± 0.03	0.20 ± 0.04	12.45 ± 1.59	33.76 ± 14.00	0.89 ± 0.03	**0.83 ± 0.07**	90.6
	2 weeks	020k	0.29 ± 0.04	0.19 ± 0.04	11.14 ± 1.66	53.21 ± 24.69	0.90 ± 0.02	0.80 ± 0.07	6.5
		060k	0.27 ± 0.03	0.19 ± 0.04	12.38 ± 1.55	38.82 ± 17.55	0.89 ± 0.02	0.82 ± 0.07	43.9
		100k	0.27 ± 0.03	0.19 ± 0.04	12.86 ± 1.24	34.40 ± 17.52	0.88 ± 0.02	0.82 ± 0.07	89.9
		140k	0.26 ± 0.04	0.19 ± 0.04	13.15 ± 1.07	**32.28 ± 15.92**	0.88 ± 0.02	0.82 ± 0.07	148.2
	4 weeks	020k	0.30 ± 0.03	0.18 ± 0.04	10.94 ± 0.96	63.06 ± 28.39	0.90 ± 0.02	0.79 ± 0.08	4.6
		060k	0.28 ± 0.03	0.19 ± 0.04	12.42 ± 1.61	43.33 ± 19.40	0.89 ± 0.02	0.81 ± 0.07	21.8
		100k	0.27 ± 0.04	0.19 ± 0.04	12.87 ± 1.24	38.57 ± 18.78	0.88 ± 0.02	0.81 ± 0.08	44.6
		140k	0.27 ± 0.04	0.18 ± 0.04	13.17 ± 1.08	36.15 ± 17.32	0.88 ± 0.02	0.81 ± 0.07	72.5
XGBClassifier	1 week	020k	0.54 ± 0.05	**0.33 ± 0.06**	7.64 ± 2.17	44.51 ± 19.27	0.94 ± 0.03	0.84 ± 0.06	23.5
		060k	0.47 ± 0.03	**0.33 ± 0.06**	8.43 ± 1.38	30.87 ± 14.39	0.93 ± 0.02	**0.86 ± 0.07**	91.3
	2 weeks	020k	0.54 ± 0.04	0.30 ± 0.06	7.45 ± 1.48	53.73 ± 25.13	0.94 ± 0.02	0.83 ± 0.07	11.7
		060k	0.47 ± 0.03	0.31 ± 0.06	8.35 ± 1.32	35.83 ± 17.26	0.93 ± 0.02	0.85 ± 0.07	45.2
		100k	0.45 ± 0.03	0.31 ± 0.06	8.74 ± 0.99	30.59 ± 17.13	0.92 ± 0.02	**0.86 ± 0.07**	85.4
		140k	0.44 ± 0.03	0.31 ± 0.06	8.98 ± 0.77	**28.00 ± 15.88**	0.92 ± 0.02	**0.86 ± 0.06**	128.6
	4 weeks	020k	0.55 ± 0.03	0.28 ± 0.06	7.31 ± 0.78	66.10 ± 29.13	0.94 ± 0.02	0.81 ± 0.08	6.0
		060k	0.48 ± 0.03	0.29 ± 0.06	8.39 ± 1.41	41.85 ± 20.10	0.93 ± 0.02	0.84 ± 0.07	23.1
		100k	0.45 ± 0.03	0.29 ± 0.06	8.73 ± 0.97	35.54 ± 19.07	0.92 ± 0.02	0.84 ± 0.07	42.7
		140k	0.44 ± 0.03	0.29 ± 0.06	9.00 ± 0.77	32.35 ± 17.36	0.92 ± 0.02	0.85 ± 0.07	64.2

TABLE 3 Results obtained in the second round of experiments, we show the average and standard deviation for all the combinations of models, retraining intervals, train set sizes and metrics for both the train and test sets. The total training time taken is shown in hours. The best values are in bold face.

Model	Interval	Size	Accuracy		Coverage		NDCG@20		Time
			Train	Test	Train	Test	Train	Test	
LogisticRegression	1 week	060k	0.25 ± 0.02	0.20 ± 0.03	11.77 ± 0.19	29.31 ± 14.00	0.92 ± 0.00	0.88 ± 0.01	14.4
XGBClassifier	1 week	060k	0.45 ± 0.02	**0.32 ± 0.04**	8.16 ± 0.15	27.31 ± 14.48	0.95 ± 0.00	**0.90 ± 0.01**	32.3
VotingClassifier	1 week	060k	0.37 ± 0.02	0.27 ± 0.04	8.76 ± 0.17	**26.83 ± 14.51**	0.95 ± 0.00	**0.90 ± 0.01**	46.1
BayesLogisticRegression	1 week	060k	0.27 ± 0.02	0.21 ± 0.03	10.77 ± 0.15	29.08 ± 14.32	0.92 ± 0.00	0.88 ± 0.01	146.6
BayesXGBClassifier	1 week	060k	0.24 ± 0.05	0.16 ± 0.04	7.67 ± 0.45	28.04 ± 15.41	0.96 ± 0.01	**0.90 ± 0.02**	137.3

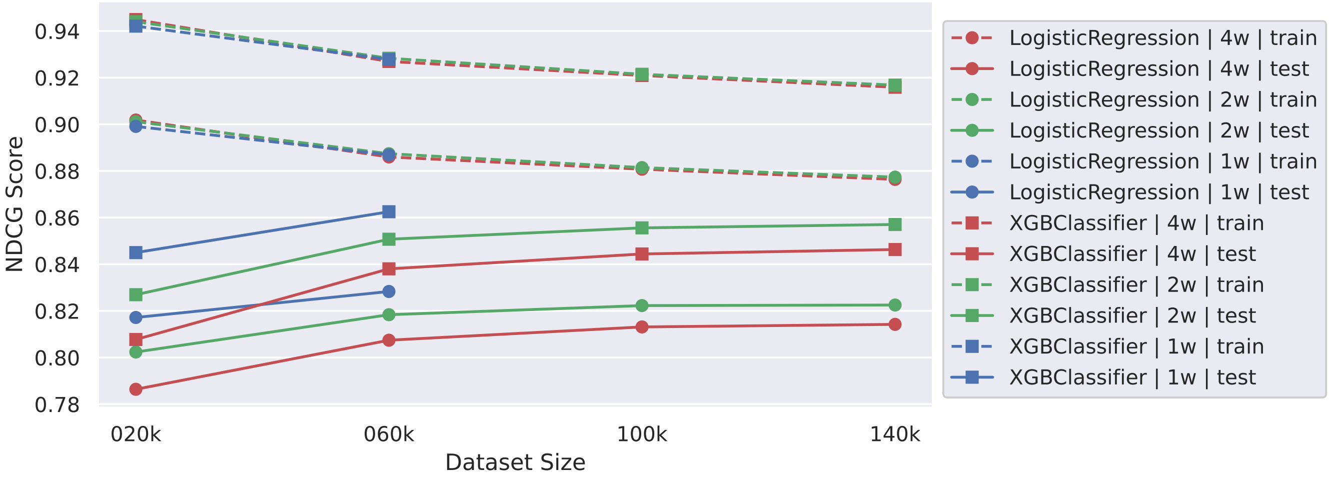

FIGURE 5 A line plot showing the average NDCG@20 scores, with the train set sizes in the x axis, each model using a different symbol, and each retraining interval using a different color.

In Figure 5 , we show the average values for the NDCG@20 scores, the same values from Table 2 , for each combination of train set sizes and retraining intervals for both models. As expected, the training scores are much better than the test scores. The retraining interval has little impact in the scores but the train set size greatly reduces the scores, this indicates that overfitting occurs with less data. For the test scores, both the train set size and retraining interval have great effects, increasing the scores. The effect of the train set size indicates that with more data the models are able to generalize better. And the effect of the retraining interval indicates that the data shows some data drift, with the future being considerably different further away it gets from the training data.

Overall, taking the LogisticRegression with 20k train size and 4 weeks retraining interval as the baseline results comparable to our last work, the NDCG@20 score is 79% in the test set. This score is lower than reported previously, given the changes to the data preprocessing this is not a problem, we are considering now a lot more test data with more defects and parts and we can now have different defects and parts between train and test sets. We have, in fact, made the problem more difficult. And, compared to this new baseline, all changes of train set size and retraining interval improve results. Also, the XGBClassifier, being a more powerful method, has better scores than the LogisticRegression but it is easier to overfit. The coverage error shows an interesting contrast, being quite stable between the models, it is greatly affected by the train set size on the test scores.

Additionally, for the total training times, it is interesting to note that the total time is close to 1058 hours, or 44 days. The baseline shows the lowest value, and all changes of model, train set size and retraining interval have increased the training time but we can see that as the size of the train set increases, the XGBClassifier becomes faster than the LogisticRegression for the same train set size.

Having concluded this first round of experiments, we selected the 60k train set size and 1 week retraining interval for use in the next round. Since the 1 week retraining interval has all the 80 snapshots available for training and the process of hyper-parameter search is very time consuming, we chose to use only the last 8 snapshots, almost 2 months of data.

The results obtained in the second round of experiments are summarized in Table 3 . The table shows results for both models using only the 60k train set size and 1 week retraining interval. It shows values for both the train and test sets for the Accuracy score, the Coverage error and the NDCG@20 score, and also the total training time. For all scores only the last 8 snapshots were used. The values for the metrics are the average and standard deviation over all the snapshots, and the total training time is the sum over all the snapshots, in hours. Additionally, it shows results for combining the models using Output Fusion and for hyper-parameter search using Bayesian Optimization.

From the results, we can see that, all models have very close NDCG@20 scores. From the other metrics we have interesting insights. The XGBClassifier has the highest Accuracy score but after the hyper-parameter search it has the worst score. The VotingClassifier, combining the other methods, has a slightly lower Coverage error. And the LogisticRegression model is still the fastest method, twice as fast as the XGBClassifier, but for the hyper-parameter search, the XGBClassifier is faster.

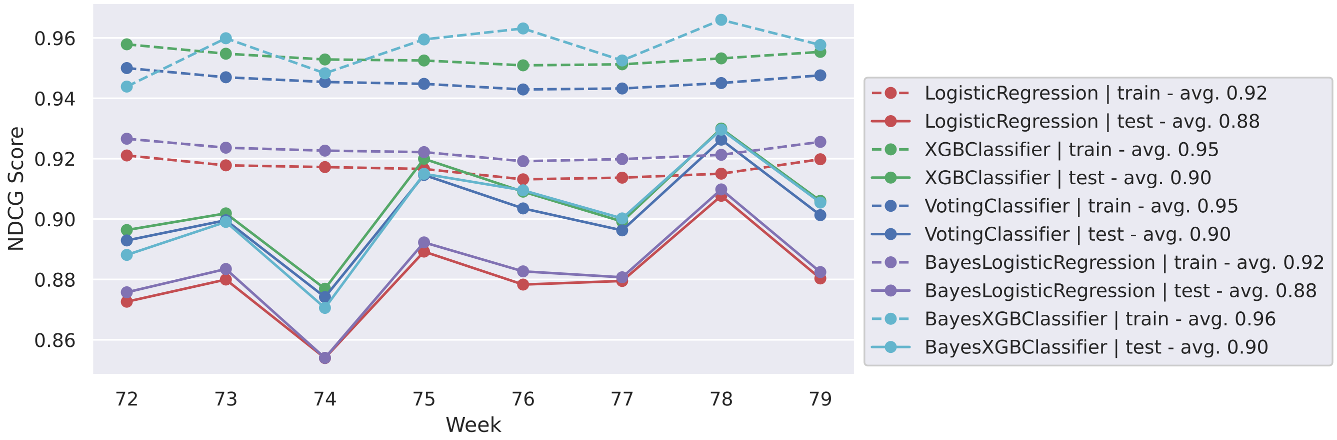

FIGURE 6 A time series plot showing the NDCG@20 scores for both the train and test sets and all the different models for the last 8 snapshots.

In Figure 6 , we show the actual values for the NDCG@20 scores for the train and test sets for all the models and the last 8 snapshots. We can see that for the test sets, all models generate similar scores for all snapshots. The VotingClassifier stays more similar to the XGBClassifier, which may indicate that the XBGClassifier is more confident of its predictions. For the train sets, most models present very stable scores, but after the hyper-parameter search, the XGBClassifier scores for the train set look much more similar to the scores for the test set.

Overall, the LogisticRegression model is the fastest and it still has a very good NDCG@20 score. The XGBClassifier model is the second fastest with a better NDCG@20 score. While the remaining models are much slower with no visible advantage, although theoretically, they should have obtained better scores. The solution for this situation would be an extensive investigation of the behaviors of the models to identify possible routes for improvements. Also, employing a business metric directly into model comparison and hyper-parameter search could provide better results.

In Figure 7 , we show the RMP values, calculated as discussed in 4.4, for the train and test sets for all the models and the last 8 snapshots. The train values are very low and stable while the test values are more unstable and higher, still we have lowered the test values from the 10% reported in our last work to just 4%.

As discussed previously, the RMP is not a good business metric, with the obtained values, we are unable to tell if marking less parts is a good result because maybe important parts are being missed. The solution for this situation would be an investigation

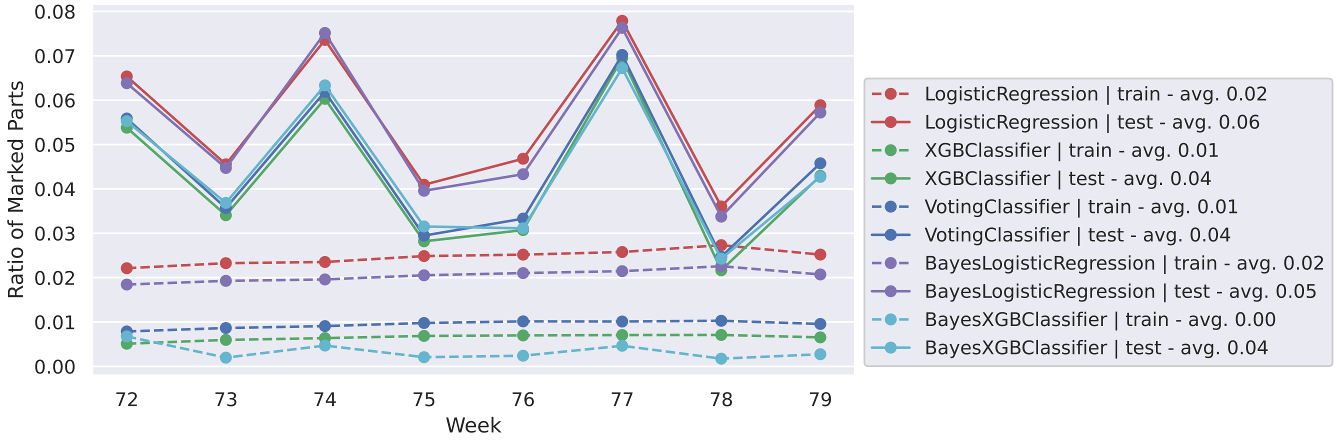

FIGURE 7 A time series plot showing the RMP for both the train and test sets and all the different models for the last 8 snapshots.

into what makes a given part more important than another part. Maybe incorporating additional data, such as life expectancy, production cost or replacement time for each spare part, could lead to an improved metric.

In Figure 8 , we show the total number of defects of each set for all snapshots. In Figure 9 , we show the total number of replaced parts of each set for all the snapshots. These plots give us an idea of the huge amount of work employed in the repair process because each defect and replaced part is, actually, work done by a trained professional. For the test sets, that have no overlapping data, we have an average of 119k identified defects and an average of 115k replaced parts per week, considering that each one of these activities takes just 1 minute of work from a professional, this amounts to a total work of almost 98 people working 40 hours per week.

Marking 4% of the 115k replaced parts amounts to close to 5k parts per week. Using these marked parts directly for alert generation would create a lot of alerts. In our last work, we have shown some simple ways to group, filter and prioritize the marked parts for use with alert generation. We note again, that alert generation is a feature of the SCCT solution, meaning that the solution has many options of how and when to generate alerts. With the marked parts integrated into an SCCT solution, additional data can be used, fields like parts availability, device type or client, and many other factors could easily be incorporated into more complex schemes to generate fine-grained alerts.

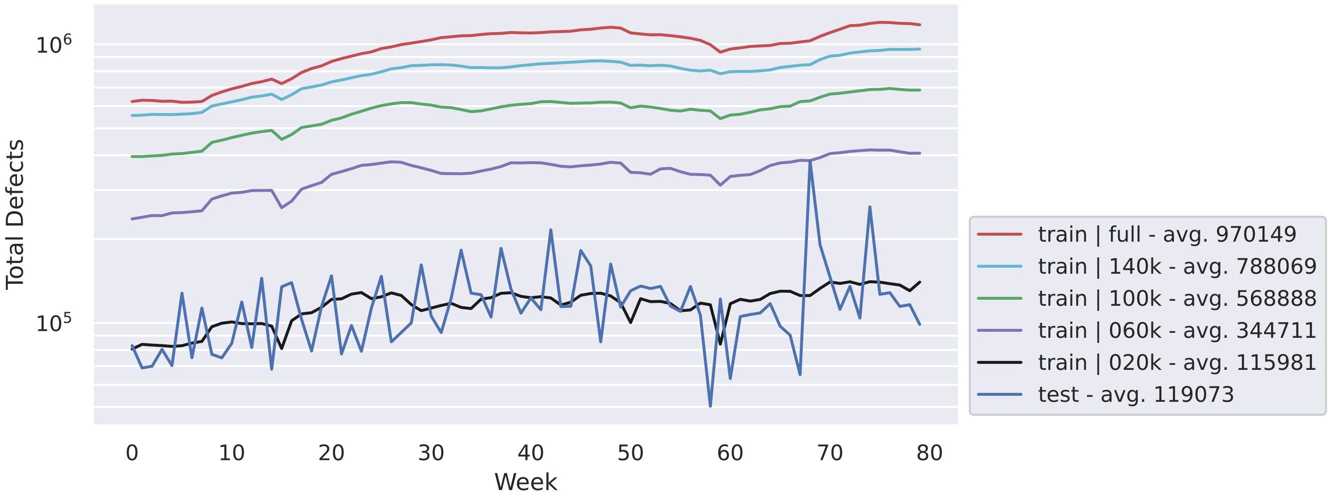

FIGURE 8 A time series plot showing the total number of defects for both the train and test sets for all the snapshots.

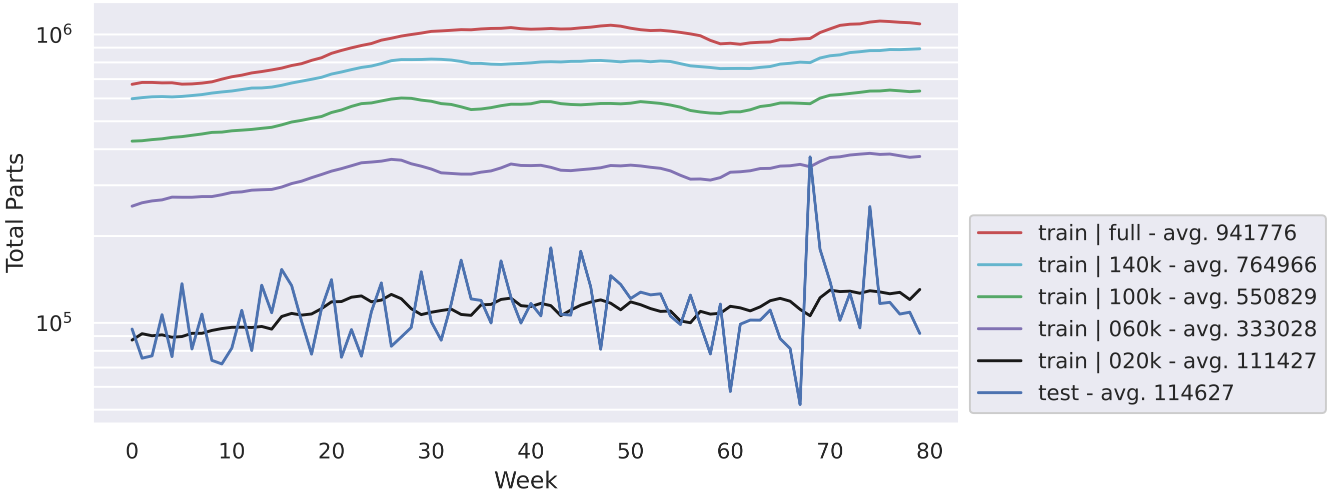

FIGURE 9 A time series plot showing the total number of parts for both the train and test sets for all the snapshots.

6 | CONCLUSION

The repair process is a complex part of many businesses. Investigating how spare parts consumption can be automatically checked allows for the development of better ways to generate alerts that drive supply chain interventions.

In this paper, we revised our learning-to-rank method to automatically check the consumption of spare parts in the repair process. We used data from an SCCT solution about the identified defects that digital devices present when sent to repair and the consumed spare parts used to address these defects. We have revised the data preprocessing and, in doing so, we have made the problem more difficult, incorporating more data and more defect and part types.

As a multi-label classification problem, we trained hundreds of Machine Learning models to rank spare parts and automatically check the consumption while simulating the passage of time. We have also explored the effects of train set size, retraining interval, different models and hyper-parameter search. From the reported results, the revised approach achieves a mean NDCG@20 score of 86% overall. When looking at the last 8 weeks of data, the revised approach achieves a mean NDCG@20 score of 90% and is able to mark a low volume of just 4% of the consumed spare parts.

The revised approach obtained with this work is promising and for future work, we intend to evaluate how the solution has impacted the real industrial setting, and to explore additional data for designing a more meaningful business metric for this task.

ACKNOWLEDGEMENTS

This work is funded by Ingenico do Brasil LTDA using incentive resources by the law 13.969/2019, and in a technical cooperation agreement with Venturus under CATI resolution 135/2020.

Notes

1. scikit-learn.org/stable/modules/calibration.html#calibration-curves
2. scikit-learn.org/stable/modules/multiclass.html
3. joblib.readthedocs.io
4. scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
5. xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier
6. scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
7. scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
8. scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
9. scikit-learn.org/stable/modules/generated/sklearn.multioutput.MultiOutputClassifier.html
10. scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
11. scikit-optimize.github.io/stable/modules/generated/skopt.BayesSearchCV.html
12. scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html
13. scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
14. scikit-learn.org/stable/modules/generated/sklearn.metrics.coverage_error.html
15. scikit-learn.org/stable/modules/generated/sklearn.metrics.ndcg_score.html
16. xgboost.readthedocs.io/en/stable/tutorials/param_tuning.html

References

1. Topan E, Eruguz AS, Ma W, Van Der Heijden M, Dekker R. A review of operational spare parts service logistics in service control towers. *European journal of operational research* 2020; 282(2): 401–414.
2. Gerrits B, Topan E, Heijden v. dM. Operational planning in service control towers–heuristics and case study. *European Journal of Operational Research* 2022.

3. Topan E, Heijden v. dMC. Operational level planning of a multi-item two-echelon spare parts inventory system with reactive and proactive interventions. *European journal of operational research* 2020; 284(1): 164–175.

4. Durugbo CM. After-sales services and aftermarket support: a systematic review, theory and future research directions. *International Journal of Production Research* 2020; 58(6): 1857–1892.

5. Gaiardelli P, Saccani N, Songini L. Performance measurement systems in after-sales service: an integrated framework. *International Journal of Business Performance Management* 2007; 9(2): 145–171.

6. Çevik Onar S, Oztaysi B, Kahraman C. Dynamic intuitionistic fuzzy multi-attribute aftersales performance evaluation. *Complex & Intelligent Systems* 2017; 3(3): 197–204.

7. Duarte E, Haro Moraes dD, Padula LL. A Learning-to-Rank Approach for Spare Parts Consumption in the Repair Process. In: Springer. ; 2022: 611–622.

8. Bhalla S, Alfnes E, Hvolby HH, Sgarbossa F. Advances in Spare Parts Classification and Forecasting for Inventory Control: A Literature Review. *IFAC-PapersOnLine* 2021; 54(1): 982–987.

9. Liu J, Kong X, Zhou X, et al. Data mining and information retrieval in the 21st century: A bibliographic review. *Computer science review* 2019; 34: 100193.

10. Liu TY, others . Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval* 2009; 3(3): 225–331.

11. Rahangdale A, Raut S. Machine learning methods for ranking. *International Journal of Software Engineering and Knowledge Engineering* 2019; 29(06): 729–761.

12. Tsoumakas G, Katakis I. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)* 2007; 3(3): 1–13.

13. Chen T, Guestrin C. Xgboost: A scalable tree boosting system. In: ; 2016: 785–794.

14. Sagi O, Rokach L. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2018; 8(4): e1249.

15. Bergstra J, Bardenet R, Bengio Y, Kégl B. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems* 2011; 24.

16. Močkus J. On Bayesian methods for seeking the extremum. In: Springer. ; 1975: 400–404.

17. Roberts DR, Bahn V, Ciuti S, et al. Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. *Ecography* 2017; 40(8): 913–929.

18. Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 2011; 12: 2825–2830.

19. Head T, Kumar M, Nahrstaedt H, Louppe G, Shcherbatyi I. scikit-optimize/scikit-optimize. online; 2021

20. Järvelin K, Kekäläinen J. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 2002; 20(4): 422–446.

How to cite this article: E Duarte, D de Haro Moraes, and LL Padula (2023), An empirical study of learning-to-rank for spare parts consumption in the repair process, *Expert Systems*, *2023*.