Cost-sensitive learning and threshold-moving approach to improve industrial lots release process on imbalanced datasets

Armindo Lobo[1][0000−0002−1517−9328], Pedro Oliveira[1][0000−0001−7143−5413], Paulo Sampaio[1][0000−0002−0879−1084], and Paulo Novais[1][0000−0002−3549−0754]

ALGORITMI Centre, University of Minho, Braga, Portugal
{lobo.armindo,poliveira199208}@gmail.com,
paulosampaio@dps.uminho.pt, pjon@di.uminho.pt

Abstract. With Industry 4.0, companies must manage massive and generally imbalanced datasets. In an automotive company, the lots release decision process must cope with this problem by combining data from different sources to determine if a selected group of products can be released to the customers. This work focuses on this process and aims to classify the occurrence of customer complaints with a conception, tune and evaluation of five ML algorithms, namely XGBoost (XGB), LightGBM (LGBM), CatBoost (CatB), Random Forest(RF) and a Decision Tree (DT), based on an imbalanced dataset of automatic production tests. We used a non-sampling approach to deal with the problem of imbalanced datasets by analyzing two different methods, cost-sensitive learning and threshold-moving. Regarding the obtained results, both methods showed an effective impact on boosting algorithms, whereas RF only showed improvements with threshold-moving. Also, considering both approaches, the best overall results were achieved by the threshold-moving method, where RF obtained the best outcome with a F1-Score value of 76.2%.

Keywords: Cost-sensitive learning · Imbalanced data · Machine Learning · Threshold-moving · Lots release.

1 Introduction

Industry 4.0 companies rely on process digitization, automation, and real-time operations to improve customer service [1]. Among these processes, lots release can significantly impact service quality.

Manually configured rules usually govern this process. Machine Learning (ML) can optimize it, improving the quality of lots release decision rates and customer satisfaction by reducing complaints. For this, managing imbalanced data from different data sources is crucial [2]. This is the case of the studied company with a software application that manages this process by applying rules defined heuristically and dealing with heavily imbalanced datasets.

In this paper two approaches to manage imbalanced datasets from automatic production tests and customer complaints as case-study are explored:

cost-sensitive learning and threshold moving. Five ML algorithms were conceived, tuned, evaluated and compared to classify the occurrence of a customer complaint based on the results of automatic production tests: *XGBoost (XGB)*, *LightGBM (LGBM)*, *CatBoost (CatB)*, *Random Forest (RF)* and *Decision Tree (DT)* .

The remainder of this paper is organized as follows. Section 2 presents the related work. Section 3 describes the methods used to deal with imbalanced datasets, the ML algorithms considered, evaluation metrics, and how the data were collected and pre-processed. Section 4 describes the experiments carried out, and Section 5 discusses the obtained results. Finally, in Section 6 are given the main conclusions and future work directions.

2 State of the Art

Several studies were carried out to deal with the imbalanced datasets problem. Costa et al. [3] carried out a study that uses ML models to predict the failure of a specific component of the Air Pressure System. They evaluate four ML algorithms, K-Nearest Neighbors (Knn), Logistic Regression (LR), Support-vector machine (SVM), DT, and RF. To cope with the imbalanced dataset, they used a cost-sensitive learning approach by adjusting the weights of SVM and LR that were inversely proportional to the fraction of cases of the corresponding class. Regarding RF and Knn, they empirically adjust the classification threshold by increasing it to improve their performance. Considering the results of the four algorithms, RF obtained the best results with a misclassification value of 3.74% for False Positives and 3.7% for False Negatives.

Altinger et al. [4] focus on a resampling (oversampling and undersampling) approach to managing automotive software fault prediction on highly imbalanced datasets, analyzing different classification algorithms with F1-Score and G-Measure as performance metrics. Undersampling only achieves good results with SVM with radial basis function Kernel, whereas oversampling achieves good results for Naive Bayes (NB), Ada Boost M1 with NB, and the divide and conquer SVM. They also conclude that XGBoost shows improvements with massive oversampling and SVM performance decrease with high oversampling. Concerning the predictability by sampling, boosting algorithms (XGBoost, Ada Boost M1) are the most unpredictable, and NB was the most stable predictor achieving the best results with undersampling and oversampling approaches.

Pereira et al. [5] analyze the handle of highly imbalanced data from eight datasets of an automotive manufacturing company considering four ML algorithms (RF, two AutoML methods and AutoEncoder) and five balancing techniques based on a two-stage performance comparison. In the first stage, all algorithms were analyzed considering two products and three balancing strategies: None, Synthetic Minority Oversampling Technique (SMOTE) and Gaussian Copula (GC). RF achieved the best results considering the overall classification and computational effort. In the second phase, all datasets were considered with five balancing methods, namely: None, SMOTE, GC, Random Undersampling, and Tomek Links to deeper analyze RF. In this scenario, the authors achieved the best results by combining RF with Gaussian Copula with an average Area Under the Curve (AUC) of 67.31.

We consider cost-sensitive learning and a threshold-moving analysis approach instead of the sampling method, common in many studies to deal with imbalanced datasets. We also use a Precision-Recall Curve analysis to calculate the best value of the threshold instead of defining it empirically.

3 Materials and methods

This section details the materials and methods used in this study. Data preparation, data exploration, the ML models used and the evaluation metrics are described in the following lines.

3.1 Data exploration

The main dataset used in this study is based on the production tests database from an automotive company, considering the period between 2019 and 2021. Due to its nature and volume, millions of tests are generated daily. Hence, the data is stored in a *Hadoop* cluster. The initial dataset whose an excerpt is available [6] has 2076005 records and 27 features. Table 1 lists some of them:

Table 1: Some features of interest in the original dataset

Field ID	Description
product	Product ID number (type of product)
serial	Unique ID of a product
stationid	Unique station ID
gof_status	Result of test sequence(GOF- *"Good or Fail")*

3.2 Data preparation

Due to the huge amount of data, the following restrictions and conditions were applied to ensure a representative dataset that includes tests related to complaints and without complaints:

– Select only the top 10 products with more complaints in 2021.
– Select a subset of 2 million tests without complaints in 2019,2020, and 2021.

We applied feature engineering to create new features based on the first insights on data conducted with data exploration.

In this process, new features were created manually, namely *"hascomplaint"* classified as the target to identify if a test is related to a complaint or not, and others concerning tests and their limit values. *Featuretools*, which applies the concept of deep feature synthesis [7], was used to create automatically 18 aggregation and transformation features related to fails and *datetime* of tests, such as the percentage of the test fails and weekday.

In the data cleaning process, we dropped the rows with missing values. Spearman's rank correlation coefficient was used to evaluate correlation among features, excluding the target, and drop highly correlated ones to avoid multicollinearity.

After this process, the final dataset has 1611371 records and 33 features. The two classes are distributed in the following manner, 3.3% from the class of tests related to customer complaints and 96.7% from the other class.

3.3 Evaluation metrics

As the process is trying to classify which tests will lead to customer complaints, this classification problem was mainly evaluated using the confusion matrix [8].

The confusion matrix has four categories. *True Positives* is the number of examples of the positive class correctly predicted by the model. *True Negatives* is the number of examples of the negative class correctly predicted by the model. *False Positives* is the number of examples of positive class incorrectly predicted by the model. *False Negatives* is the number of examples of negative class incorrectly predicted by the model. Based on these values and as the focus is to deal with imbalanced datasets *F1-Score*, *Precision* and *Recall* metrics are analyzed. *Recall* is the fraction of positive labels correctly identified by the model.

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

Precision is the fraction of results that are relevant.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

F1-score is the harmonic-mean of *precision* and *recall*

$$FScore = 2 \times \frac{precision \times recall}{precision + recall}$$

.

3.4 Decision Trees

DT is a ML model structured with three main components, a root node, branches, and leaf nodes. Each internal node represents an attribute "test," each branch represents the test's result, and each leaf node represents a class label. The decisions are taken when all attributes are computed. DT models can be used both on classification and regression problems and, due to their nature, generate results that are easier to interpret and explain [9].

3.5 Random Forest

RF is an ensemble method created with several small decision trees called estimators. Each one of these estimators is conceived with a random subset of features and produces its predictions. These predictions, trained independently, are combined to produce a more accurate forecast. RF deals well with the overfitting and generalizes well to new data. The result of RF is the class selected by most trees on classification or the average of predictions on regression problems. RF uses the concept of bootstrap aggregating or bagging. Some samples from the original dataset are generated randomly by replacement, meaning that each row can be selected more than once [10].

3.6 Gradient Boosting: XGBoost, LightGBM and CatBoost

Boosting is an ensemble learning method consisting of a set of weak learners (models that perform slightly better than random guessing) that are applied sequentially and combined into a strong learner [11]. A boosting algorithm optimizes a loss function and assigns different weights to its estimator's outputs. In gradient boosting, which combines the gradient descent algorithm and boosting method, the predictors are not made independently, such as RF, but sequentially, where each tree corrects the errors made by the previous tree.

XGBoost is an advanced implementation of a gradient boosting algorithm that uses a new regularization technique to control the overfitting and was designed to improve scalability and speed using far fewer resources than existing systems. In XGB, decision trees are grown level-wise, meaning that trees are growing horizontally. A new level only expands when the previous is already expanded. Other distinctive features are its sparsity-aware algorithm and weighted quantile sketch for approximate tree learning to deal with sparse data and how it takes advantage of hardware resources. All of these capabilities make XGB able to deal with enormous datasets efficiently [11].

LightGBM is another advanced implementation of a gradient boosting algorithm designed to reduce the implementation time. One distinctive aspect of LGBM is that decision trees are grown leaf-wise, meaning that trees are growing vertically. Several advantages are associated with LGBM, such as a faster training speed, higher efficiency, better accuracy, support of parallel and GPU learning, handling of large-scale data, and memory optimization [11].

CatBoost is another advanced implementation of a gradient boosting algorithm. CatB is structured using symmetric or oblivious trees, meaning that at each level of the tree, it uses the same features to split learning instances. CatB brings two major advancements in gradient boosting algorithms, an innovative algorithm for processing categorical features, as well as the implementation of ordered boosting [11].

3.7 Cost-sensitive learning

Cost-sensitive learning is applied at an algorithmic level, based on a cost matrix, where conceptually, the cost of misclassification should always be greater than the cost of correct classification [12]. This technique aligns with the problem of dealing with imbalanced datasets with a skewed class distribution.

In this study, cost-sensitive learning is implemented by adjusting the weight of each class through the parameter *class_weight* available for each analyzed algorithm. The parameter *scale_pos_weight* calculates the weights of each class according to the following formula:

$$scale\_pos\_weight = \frac{\sum(testsok)}{\sum(testsfail)}$$

is used in *XGB*, *CatB* and *LGBM* models. For *RF* and *DT*, the *class_weight* parameter that automatically adjusts weights inversely proportional to class frequencies was instantiated with *"balanced"* mode.

3.8 Threshold-moving

In classification problems, algorithms try to predict the probability of class membership by comparing it against a defined threshold (0.5 by default). The default threshold may not be appropriate when dealing with imbalanced datasets and lead to poor results. One way to tackle this problem can be made through *threshold-moving* or *thresholding* which consists in adjusting the threshold from the training instances and tuning it to achieve the best results [13].

This threshold analysis can be made using *ROC Curve* or *Precision-Recall Curve*. *ROC Curve* is obtained by plotting the true positive rate versus the false positive rate at different threshold values. On the other hand, *Precision-Recall Curve* plots the trade-off between precision values and corresponding recall values for a predictive model at different probability thresholds.

To perform this threshold analysis, we considered the Precision-Recall curve, as different studies have concluded that it is more suitable than the ROC curve to deal with highly unbalanced data [14].

4 Experiments

Five ML models were conceived, tuned, and evaluated to carry out the study. The dataset was split, using 80% for training and 20% for testing. To tune and find the best hyperparameters for these models, *Hyperopt* was used [15].

The hyperparameters searched are the same for all gradient boosting algorithms. RF and DT also have similarly searched hyperparameters except for n_estimators which is not available for DT. Table 2 depicts the hyperparameter searching space.

This study was implemented using *Python*, version 3.7, including some libraries such as *Pandas*, *Numpy* and *MatPlotlib*. All the ML models were conceived using *Scikit-Learn*.

Table 2: Hyperparameter searching space

Hyperparameter	DT	RF	XGB	LGBM	CatB
n_estimators	—	{100,200,300}	{70,80,90,100}	{70,80,90,100}	{70,80,90,100}
max_depth	{10,20,30}	{10,20,30}	{3,4,5}	{3,4,5}	{3,4,5}
learning_rate	—	—	[.1,.3]	[.1,.3]	[.1,.3]
colsample_bytree	—	—	{.5,.6,.7,.8.,.9,1}	{.5,.6,.7,.8.,.9,1}	—
min_samples_leaf	{1,2,4}	{1,2,4}	—	—	—
min_samples_split	{2,4,6}	{2,4,6}	—	—	—

5 Results and Discussion

A cost-sensitive learning and threshold-moving approach were carried out on an imbalanced dataset to analyse the obtained results by the different models. Table 3 shows the results for each model, obtained without adjusting the class weight parameter. The best hyperparameters found and used in the whole process, with the different approaches, are included here.

Table 3: Obtained results for conceived models without class weight adjustments. Legend: a-n_estimators; b-max_depth; c-learning_rate; d-colsample_bytree; e-min_samples_leaf; f-min_samples_split

Model	a	b	c	d	e	f	Precision	Recall	F1-Score
DT	—	20	—	–	2	6	66.7%	**42.2%**	**51.7%**
RF	100	20	—	—	1	6	**100%**	21.7%	35.7%
XGB	70	5	.2527	.6	—	—	98.5%	26.3%	41.5%
LGBM	100	4	.2953	.9	—	—	83.7%	34.2%	48.5%
CatB	100	5	.2533	—	—	—	81.9%	27.3%	41.0%

As shown, without adjustments, DT achieves the best F1-Score with a value of 51.7%. Regarding hyperparameters, DT and RF shared the same value of max_depth and min_samples_split. XGB and CatB has very similar learning_rate values and the same value for max_depth. Table 4 shows the results by adjusting the class weight parameter for each model.

Table 4: Obtained results for conceived models with class weight adjustment

Model	Precision	Recall	F1-Score
DT	51.3%	31.1%	38.8%
RF	**100**%	15.6%	27.0%
XGB	84.9%	**47.2%**	**60.7%**
LGBM	67.7%	46.9%	55.4%
CatB	70.7%	41.1%	52.0%

The results demonstrate that the cost-sensitive learning approach effectively impacted boosting algorithms, where XGBoost achieved the best outcome globally with a F1-Score value of 60.7%. However, this approach was not effective on

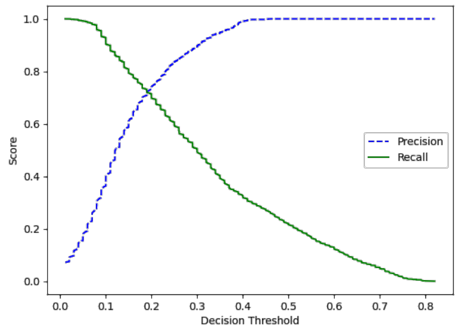
DT and RF, with a decrease in their results. When using the threshold-moving approach, it is necessary to analyze the precision-recall curve to identify the optimal threshold regarding F1-Score for each model [16]. By optimizing *F1-Score*, it is possible to calculate the threshold that results in the best trade-off between precision and recall. Table 5 lists the results of applying threshold-moving adjustment to the different ML models.

Table 5: Obtained results for conceived models with threshold-moving

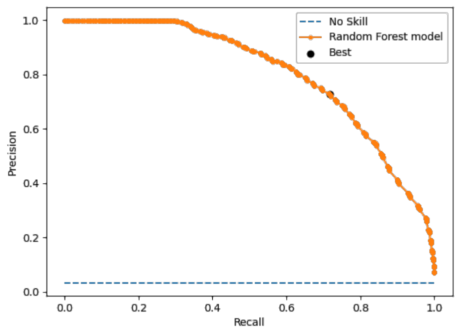
Model	Precision	Recall	F1-Score
DT	65.8%	41.5%	50.9%
RF	77.7%	**74.6%**	**76.2%**
XGB	**79.5%**	51.3%	62.4%
LGBM	61.4%	53.9%	57.4%
CatB	58.2%	47.0%	52.0%

As demonstrated on Table 5, when compared with the values without adjustments, all algorithms, except DT, improved their results with the threshold-moving method. The best overall result was achieved by RF with a *F1-Score* of 76.2%. Analyzing the best results for F1-Score and comparing its gains against the obtained results without any adjustments, RF obtained 40.5 percentage points gains with the threshold-moving approach.

Fig. 1 depicts the evolution of the *Precision-Recall* curve and *F1-Score* according to different thresholds for RF model. As shown in chart (a), as one of these values increases, the other decreases. The best trade-off between them is achieved when both curves intersect each other. F1-Score reflects this in the chart (b), which calculates the best balance between Precision and Recall.



(a) Precision-Recall Curve



(b) Optimal threshold for F1-Score

Fig. 1: Random Forest - threshold-moving analysis

The threshold-moving achieved the best overall outcome for *F1-Score*, where RF achieves the best result. Boosting algorithms showed consistent results and improvements with both methods, whereas DT didn't show improvements and RF only improved with threshold-moving.

6 Conclusion

In an industry 4.0 context, companies must deal with vast amounts of data gathered from multiple sensors and different data sources, leading to imbalanced datasets. This is the case of the studied company, whose lots release process relies on an SW based on rules defined heuristically. We propose a new approach that aims to establish a relationship between customer complaints and automatic tests by applying ML models that deal with imbalanced datasets, to help in the lots release decision process. We found that the overall efficiency of the process is improved thus reduce customer complaints.

This study focused on analyzing two different approaches to dealing with imbalanced datasets. Cost-sensitive learning and threshold-moving are analyzed and evaluated according to their impact on *F1-Score* results. To achieve this, several experiments were conceived, tuned, evaluated and compared, considering five ML algorithms.

The results showed that both methods are effective in improving the *F1-Score* result of boosting algorithms. Whereas RF only showed improvements with the threshold-moving approach, and DT did not improve with any of them. From the analysis of results, it is possible to conclude that globally, the threshold-moving approach achieved the best overall outcome, and among the analyzed algorithms, the best one was achieved by RF with a *F1-Score* value of 76.2%.

As future work, some sampling methods to deal with imbalanced datasets should also be considered and analyzed, namely SMOTE for oversampling, Tomek Links for undersampling and SMOTE-Tomek that is a combination of both. We also intend to consider adding new features from other datasets, such as product repairs. This should result in more correlations with customer complaints and improve the classification results. We also intend to explore new algorithms, namely artificial neural networks and improving hyper-parameter tuning, to compare the results with the ones in this study.

Acknowledgments. This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

References

1. Rojko, A. (2017). Industry 4.0 concept: Background and overview. International Journal of Interactive Mobile Technologies, 11(5). doi: 10.3991/ijim.v11i5.7072
2. Fathy, Y., Jaber, M., & Brintrup, A. (2020). Learning with imbalanced data in smart manufacturing: A comparative analysis. IEEE Access, 9, 2734-2757. doi: 10.1109/ACCESS.2020.3047838

3. Costa, C. F., & Nascimento, M. A. (2016, October). Ida 2016 industrial challenge: Using machine learning for predicting failures. In International Symposium on Intelligent Data Analysis (pp. 381-386). Springer, Cham. doi: 10.1007/978-3-319-46349-0_33
4. Altinger, H., Herbold, S., Schneemann, F., Grabowski, J., & Wotawa, F. (2017, February). Performance tuning for automotive software fault prediction. In 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 526-530). IEEE. doi: 10.1109/SANER.2017.7884667
5. Pereira, P. J., Pereira, A., Cortez, P., & Pilastri, A. (2021, September). A comparison of machine learning methods for extremely unbalanced industrial quality data. In EPIA Conference on Artificial Intelligence (pp. 561-572). Springer, Cham. doi: 10.1007/978-3-030-86230-5_44
6. Lobo, Armindo, Moreira, Guilherme (2022), "Tests and Complaints", Mendeley Data, V1, doi: 10.17632/5xnj2z5z48.1
7. Kanter, J. M., & Veeramachaneni, K. (2015, October). Deep feature synthesis: Towards automating data science endeavors. In 2015 IEEE international conference on data science and advanced analytics (DSAA) (pp. 1-10). IEEE. doi: 10.1109/DSAA.2015.7344858
8. Jeni, L., Cohn, J., & De La Torre, F. (2013) Facing Imbalanced Data– Recommendations for the Use of Performance Metrics In 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction (pp. 245-251). doi: 10.1109/ACII.2013.47
9. Sharma, H., & Kumar, S. (2016). A survey on decision tree algorithms of classification in data mining. International Journal of Science and Research (IJSR), 5(4), 2094-2097. doi: 10.21275/v5i4.NOV162954
10. Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32. doi: 10.1023/A:1010933404324
11. Al Daoud, E. (2019). Comparison between XGBoost, LightGBM and CatBoost using a home credit dataset. International Journal of Computer and Information Engineering, 13(1), 6-10. doi: 10.5281/zenodo.3607805
12. Elkan, C. (2001, August). The foundations of cost-sensitive learning. In International joint conference on artificial intelligence (Vol. 17, No. 1, pp. 973-978). Lawrence Erlbaum Associates Ltd.
13. Sheng, V. S., & Ling, C. X. (2006, July). Thresholding for making classifiers cost-sensitive. In AAAI (Vol. 6, pp. 476-81).
14. Davis, J., & Goadrich, M. (2006, June). The relationship between Precision-Recall and ROC curves. In Proceedings of the 23rd international conference on Machine learning (pp. 233-240). doi: 10.1145/1143844.1143874
15. Putatunda, S., & Rama, K. (2018, November). A comparative analysis of hyperopt as against other approaches for hyper-parameter optimization of XGBoost. In Proceedings of the 2018 International Conference on Signal Processing and Machine Learning (pp. 6-10). doi: 10.1145/3297067.3297080
16. Brownlee, J. (2019). Probability for machine learning: Discover how to harness uncertainty with Python. Machine Learning Mastery.