A Pipeline for Monitoring and Maintaining a Text Classification Tool in Production

Elene F. Ohata[1], **César Lincoln C. Mattos**[2], **Paulo Antonio L. Rêgo**[2]

[1]Department of Teleinformatics Engineering (DETI/UFC)
Campus do Pici, Bloco 725, Fortaleza, 60355-636, Ceará, Brazil

[2]Department of Computer Science (DC/UFC)
Campus do Pici, Bloco 910, Fortaleza, 60355-636, Ceará, Brazil

elene.ohata@lapisco.ifce.edu.br, {cesarlincoln,paulo}@dc.ufc.br

Abstract. *Text classification has been a core component of several applications. Modern machine learning operations strategies address challenges in deploying and maintaining models in production environments. In this work, we describe and experiment with a pipeline for monitoring and updating a text classification tool deployed in a major information technology company. The proposed fully automatic approach also enables visual inspection of its operations via dashboards. The solution is thoroughly evaluated in two experimental scenarios: a static one, focusing on the Natural Language Processing (NLP) and Machine Learning (ML) stages to build the text classifier; and a dynamic one, where the pipeline enables automatic model updates. The obtained results are promising and indicate the validity of the implemented methodology.*

1. Introduction

ML solutions have been adopted in the most diverse production environments, with companies deploying models to improve customer service and enhance internal processes. ML tools capable of handling textual data using NLP are especially prevalent due to the large amount of human-generated content in textual format. In that sense, besides asking how ML can transform a given procedure, recent ML literature has shed light on the question of how difficult is to deploy models in practice [Paleyes et al. 2022].

Seminal work by [Sculley et al. 2015] has looked at real-world ML solutions through the lens of software engineering, analyzing the technical debt inherent to such systems, which results in massive ongoing maintenance. For instance, the authors highlight how the system components dedicated to the actual ML algorithms are quite small compared to the other components necessary for the solution to work properly.

The collection of practices to automate all the processes of an ML system using methodologies from the DevOps literature has been called MLOps [Alla et al. 2021]. Besides sharing the interest of DevOps in the systematization of developing, releasing, and maintaining software, MLOps is also concerned with the automation of the data handling and modeling [Gift and Deza 2021]. After deployment, the requests received by the system and the resulting predictions should be monitored. Newly acquired data should then be used to update the model with the aim of improving its performance [Symeonidis et al. 2022].

The benefits of MLOps are especially relevant when frequent retraining is necessary [Mäkinen et al. 2021]. The result is a cyclic workflow, such as the one defined by the CRISP-ML(Q), a cross-industry standard process model for developing ML applications with quality assurance methodology [Studer et al. 2021]. The main steps of CRISP-ML (Q) are business and data understanding, data preparation, modeling, evaluation, deployment, monitoring, and maintenance.

In this work, we focus on the modeling, evaluation, monitoring, and maintenance components of the CRISP-ML(Q) workflow. We consider the task of performing automatic classification of technical texts within an internal support system from a large company. The goal is to predict the main category of a given textual description of a technical problem. The learning task presents challenges such as the predominance of short texts, the heavy usage of technical jargon and acronyms, and the presence of ambiguous labels. We detail a learning strategy based on the use of NLP techniques and supervised models to perform multiclass classification.

Fortunately, the intrinsic nature of the analyzed system operation results in the frequent collection of new labeled instances, which enables periodic model updates. Thus, we propose and implement a fully automated pipeline to perform model retraining, including an inner loop of model selection, leveraging new data acquired while monitoring the deployed solution. Such a scenario where new labeled examples become available over time is not uncommon in systems that aim to predict information that may be later filled by a human, e.g., a product rating or book categorization. Thus, we envision the proposed pipeline being leveraged in other scenarios and applications.

In summary, the contributions of this work are as follows:

1. The proposal of a comprehensive NLP and ML workflow to perform model training and selection for a text classification tool;
2. The proposal of a fully automated pipeline to monitor the deployed solution in production and use new data to retrain the model;
3. Evaluation of the obtained ML tools in two scenarios, a static one, with fixed model and data, and a dynamic one, where the model is updated periodically.

In addition, we illustrate how the deployed model can be monitored using customizable dashboards built with the Grafana software. Finally, we use such dashboards to visualize the improvement behavior of the ML system over successive updates, which exemplifies the importance of this step for an enduring ML solution.

2. Related Work

Numerous studies within the existing literature have presented techniques and workflows for executing model monitoring and updates in adherence to modern MLOps conventions and tools. Furthermore, there are studies proposing approaches aimed at enhancing the comprehension of human language and enabling more proficient responses to human needs by taking advantage of the capabilities of NLP and ML methodologies.

[Nigenda et al. 2022] introduced the Amazon SageMaker Model Monitor, a service designed for continuous monitoring of the performance of ML models deployed on Amazon SageMaker. This system autonomously identifies various types of model drift in real-time. It also generates alerts, enabling model owners to take corrective actions,

thereby ensuring the ongoing quality of their models. Their focus was primarily on tabular ML applications, but users may have a wide array of use cases, including the monitoring of image classification models or tracking metrics associated with NLP models.

[Kaminwar et al. 2021] presented a collection of recommendations aimed at validating models, code, and data at various stages of the ML life cycle. They proposed some guidelines substantiated by an examination of common errors and performance challenges observed across various industrial contexts. They stated that the root causes of these errors extend beyond the model itself, with code and data assuming significant roles. In addition, the development of an all-encompassing checklist was deemed impractical due to the intricate nature of ML tasks.

The combination of NLP and ML has been employed to assist large companies in various tasks. [Borg et al. 2021] proposed the use of ML techniques to classify emails into 33 different classes, to enhance customer support within a telecommunications company. [Arias-Barahona et al. 2023] suggested a methodology that involves utilizing the combination of Term Frequency-Inverse Document Frequency (TF-IDF) along with the Synthetic Minority Over-sampling Technique, complemented by the Support Vector Machine, as a method for categorizing client requests into specific categories.

The learning task in our work is located within the scope of the latter contributions. However, after designing the NLP strategy and the ML training procedure, we also pursue a formal pipeline to perform monitoring and model updates, following the recent MLOps practices and tools. We aim to automate most of those steps, while also enabling inspection of the deployed solution results via customizable dashboards.

3. Problem Description

We aim to aid the operation of an asynchronous support system within a major IT company by categorizing new issues to promote a faster and more effective response. In summary, a technical support agent submits relevant information about equipment or system components that require maintenance to the business team through a text-based tool. In this tool, agents can submit general information using select boxes and describe the issue in detail textually. The business team relies on this information to analyze and address the issue. Among the select boxes, the main issue category is crucial for efficiently resolving issues. Correctly categorizing issues helps not only solve problems, but also monitor recurring problems. However, inaccurately categorized issues can slow down resolution processes and corrupt system logs, which hinders their value. Since selecting the main category is mandatory, the business team is advised to correct any inaccuracies in this selection to maintain system integrity. Moreover, the requirement to choose a main issue category during operations means that the system can record reviewed issues, including those corrected and those evaluated without category changes, as newly gathered data.

The above scenario provides an incentive for employing an AI assistant. The AI assistant should act when the agent submits the issue, where it will check if the written text corresponds to the main issue category; if it does not, an alert message should appear, indicating a suggested main category. Figure 1 illustrates the sequence of steps expounded in the previous paragraphs, beginning with the identification of system issue by technical agents and concluding with the presentation of suggested solutions by the business team.

Updating the ML model by integrating newly acquired data is essential to ensure

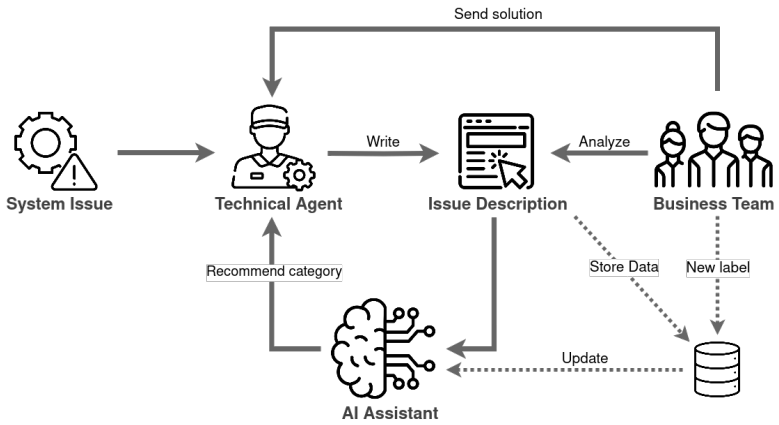

Figure 1. Steps of the resolution of a system issue.

its continued effectiveness and relevance. This practice will serve as a crucial aspect of model maintenance, allowing the system to adapt to new issues, thus enabling more accurate predictions. Regularly feeding newly acquired data facilitates ongoing learning and refinement, which ensures that the model remains valuable in assisting the agents to better categorize issues.

The available data includes unconventional text elements characterized by abbreviations and industry-specific jargon; some terms are unique to the company's internal context. We have also observed numerous spelling errors within the dataset. There is an additional challenge where agents occasionally assign incorrect main categories, and the error persists without correction in certain instances. As follows, we describe a methodology to process such data via NLP techniques and use it to train supervised ML models. Importantly, we also describe how to monitor and update the resulting solution.

4. Proposed Methodology

At a higher level, our pipeline consists of API, monitoring, and visualization modules. The first implements the endpoints of the AI model, while the others enable tracking and observing the main metrics related to the AI tool. Figure 2 illustrates this workflow and its technologies. It includes a traffic simulation component, using the Locust library for controlled experimentation, which is suppressed after deployment. We employed the FastAPI framework to build the API. We chose a combination of Prometheus and Grafana for the monitoring stage, enabling us to gather metrics and visualize them effectively. In this work, we use Locust to simulate traffic in the form of model requests for controlled experimentation. In the following sections, we detail the data representation, the processing steps, and each macro component of the proposed workflow.

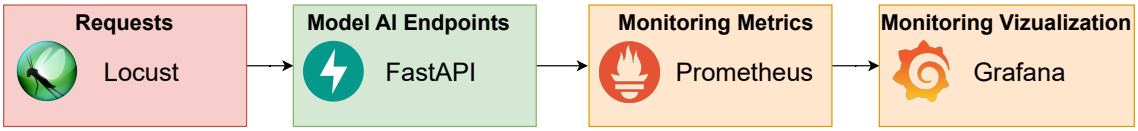

Figure 2. High-level view of the considered pipeline and technologies.

4.1. Data Preparation

The collected dataset consists of stored data from an Oracle Database, totaling 6145 samples. These samples are accompanied by labels that fall into 11 categories[1], from which

[1]Due to confidentiality reasons, the categories are not fully detailed here.

users must select the most relevant one to categorize their reported issue. In Figure 3, the distribution of samples across each category is depicted, revealing a substantial imbalance in the dataset. Notably, class "3" contains a significantly higher number of samples, which can be attributed to two primary factors. Firstly, class "3" represents the most common problem in the support system. Secondly, many agents tend to select this class when they are uncertain about the appropriate category to choose. As we will discuss later, this is a relevant cause of confusion in the ML model predictions.

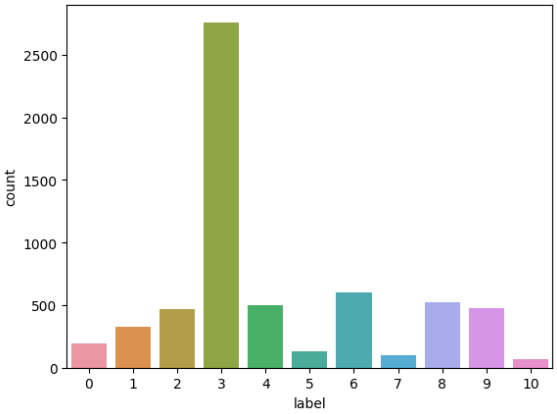

Figure 3. Amount of samples for each label.

The dataset presents challenges that make its classification difficult. Its texts include technical acronyms, incomplete phrases, misspellings, and specialized terminology. Consequently, before the classification phase, the raw data requires careful preprocessing. We adopt the following preprocessing steps: conversion to lowercase text, correction of common misspellings, identification of language (with current support limited to English), tokenization, lemmatization, and exclusion of punctuation, special symbols, unnecessary white spaces, and stopwords.

The next step consists of converting the textual data into a numeric vectorized format, which can be performed with several techniques. We investigate three extensively used methods: Term Frequency-Inverse Document Frequency (TF-IDF), Word2Vec, and Doc2Vec. Due to very specific wording and acronyms, both Word2Vec and Doc2Vec embeddings were trained from scratch instead of using pre-trained solutions [Kim 2014].

4.2. Model training and evaluation

We consider multiple supervised ML models in the classification step, which were chosen based on their popularity in NLP applications [Haq et al. 2022, Essa et al. 2023]. We also opted to evaluate diverse learning strategies. The methods are k-Nearest Neighbors (KNN), Light Gradient Boosting (LightGBM), Adaptive Boosting of decision trees (AdaBoost), Multinomial Naive Bayes (MNB), Multilayer Perceptron (MLP), Random Forest (RF), and Support Vector Machine (SVM). The latter is considered with either a linear kernel or a nonlinear RBF (radial basis function) kernel.

The classification stage is performed using a nested cross-validation technique, as indicated in Figure 4. This method divides the dataset into p outer partitions, forming p unique training and validation pairs. Concurrently, the p training sets are subdivided into q inner partitions to create the inner folds. The inner loop is responsible for optimizing the model hyperparameters, while the outer loop is responsible for

selecting the best ML model [Cawley and Talbot 2010]. This careful approach guarantees that the validation sets of the outer folds are not employed in hyperparameter tuning, preventing any data leakage and overly optimistic bias in the models' performance [Wainer and Cawley 2021].

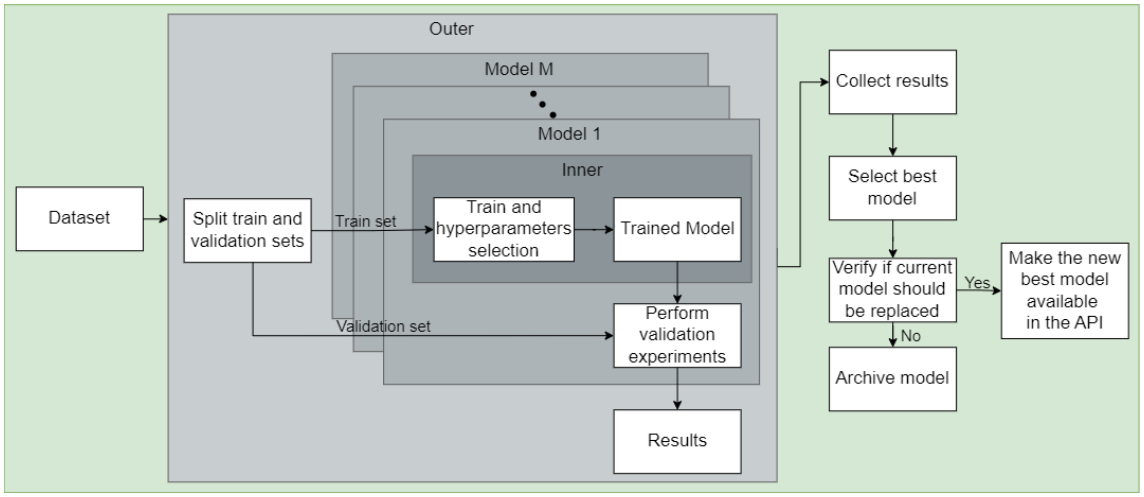

Figure 4. Nested cross-validation scheme to perform model selection and hyperparameter optimization with no data leakage in any step.

In this work, we define $p = q = 5$. In addition, we used random search to optimize the hyperparameters. Table 1 shows the hyperparameters considered for each classifier.

Table 1. Hyperparameters grid for each classifier during the optimization phase.

Classifier	Parameter	Setup
AdaBoost	learning rate	[0.0001, 0.001, 0.01, 0.1, 1.0]
	number of estimators	10 to 500 in steps of 10
KNN	number of neighbors	[3, 5, 7, 9, 11, 15, 17]
LightGBM	learning rate	[0.0001, 0.001, 0.01, 0.1, 1.0]
	number of estimators	10 to 500 in steps of 10
MLP	neurons in the hidden layer	2 to 1000 in steps of 50
MNB	-	-
Random Forest	number of estimators	25 to 2000 in steps of 50
SVM (Linear)	C	$[2^{-5}, 2^{-3}, \ldots, 2^{15}]$
SVM (RBF)	C	$[2^{-5}, 2^{-3}, \ldots, 2^{15}]$
	γ	$[2^{-15}, 2^{-13}, \ldots, 2^{3}]$

We also employed hyperparameter selection for the TF-IDF technique. The threshold used to eliminate very frequent tokens is adjusted within the interval from 0.3 to 1. Tokens with frequency above this threshold are not considered in the embedding phase. We also restrict the selection to unigrams, bigrams, or trigrams.

When comparing distinct model strategies and configurations, we consider accuracy and F1-score metrics. The accuracy represents the proportion of the accurately predicted classes. Since we are dealing with multiple classes, we employ the weighted F1-score. Thus, the F1-score is initially calculated individually for each label. Subsequently, a weighted mean is executed based on the number of instances in each category.

4.3. Monitoring and Maintenance

This section outlines the steps to implement and manage the ML model for monitoring and maintenance. For the monitoring stage, we use Prometheus for metrics collection and

Grafana for metrics visualization. We use an endpoint to perform the model maintenance. The workflow used to perform monitoring and maintenance is depicted in Figure 5.

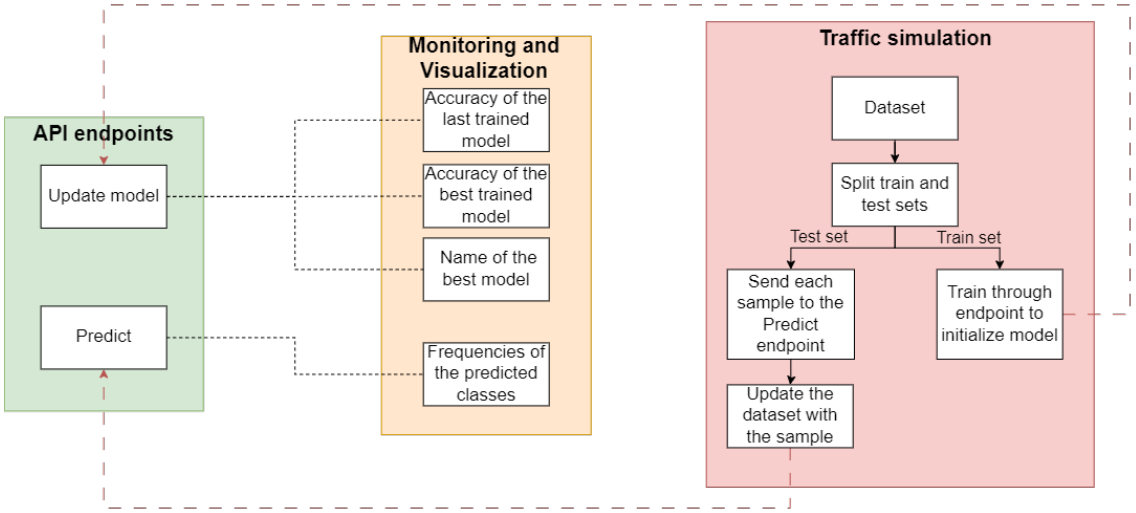

Figure 5. Workflow employed in the monitoring and maintenance stage.

To monitor the performance of the ML model, the main metrics collected from the designated endpoints are i) accuracy of the last trained model; ii) accuracy and iii) name of the best-trained model; and iv) frequencies of the predicted classes. Visualizing these collected metrics is also essential; then, a dedicated dashboard was created in a user-friendly way that allows users to gain a quick understanding of model performance at a glance. A suggested dashboard is shown in Section 5.2. The metrics and their visualization track the historical validation accuracy scores of the model and performance trends over time. Moreover, Grafana can be configured to emit alerts to notify when model performance deviates from predefined thresholds.

To ensure that the ML model remains up-to-date, it is crucial to adapt it to new data. To facilitate model maintenance, an "Update model" endpoint was established. This endpoint ensures that the ML model can be continuously improved and adapted to changing data. The enhancement of the model can be accomplished through the execution of a request to the endpoint. Additionally, an alternative approach involves the utilization of a third-party library, such as *fastapi-utils*, to facilitate the initiation of periodic tasks.

In the context of traffic simulation for controlled experimentation, we partition the dataset into test and train sets. The train set is employed to initialize the model using the existing endpoint, while the test set simulates the requests. After each prediction, the sample updates the dataset used for training. This approach yields a dynamic dataset, which enables the observation of the collected metrics. It is important to emphasize that the traffic simulation does not affect the trained models presented in Section 5.1.

5. Experimental Results and Discussion

In this section, we present the outcomes achieved by employing various methods for extracting textual features and diverse ML approaches to recommend an appropriate main issue category to enhance a technical support system. This section is divided into two subsections. The first comprises the static classification metrics, while the second presents the experiments related to model monitoring and maintenance.

5.1. Static classification experiment

Table 2 presents the outcomes derived from 5 iterations, as previously described in Section 4.2. A total of 24 experiments were conducted, combining three distinct text feature extraction approaches with eight classification methods. The algorithms were coded in Python 3.8, with key libraries including scikit-learn, gensim, nltk, and spacy.

Table 2. Quantitative results (average ± std. deviation) for every text feature extraction and classifier combination. The best result is highlighted in blue.

Feature Extraction	Classifier	Accuracy (%)	F1-score (%)
Word2Vec	AdaBoost	47.23 ± 1.23	35.54 ± 1.91
	KNN	48.14 ± 1.64	43.28 ± 1.47
	LightGBM	55.02 ± 1.10	50.07 ± 1.12
	MLP	49.8 ± 3.15	38.93 ± 6.24
	MNB	43.69 ± 1.74	29.04 ± 2.63
	Random Forest	54.0 ± 1.04	47.05 ± 1.09
	SVM (Linear)	52.04 ± 1.28	45.42 ± 0.91
	SVM (RBF)	50.51 ± 1.33	48.82 ± 1.04
Doc2Vec	AdaBoost	55.04 ± 2.02	49.4 ± 2.62
	KNN	57.02 ± 1.08	50.71 ± 1.38
	LightGBM	64.0 ± 1.24	60.94 ± 1.24
	MLP	60.94 ± 1.41	56.12 ± 1.57
	MNB	44.9 ± 1.53	27.84 ± 1.62
	Random Forest	60.75 ± 1.46	55.02 ± 1.96
	SVM (Linear)	61.64 ± 0.62	58.92 ± 0.96
	SVM (RBF)	62.36 ± 0.61	59.45 ± 0.73
TF-IDF	AdaBoost	57.09 ± 12.80	52.24 ± 9.36
	KNN	64.39 ± 1.23	58.77 ± 1.75
	LightGBM	75.0 ± 1.71	73.31 ± 1.91
	MLP	72.61 ± 1.19	70.58 ± 1.45
	MNB	53.12 ± 1.55	40.78 ± 1.86
	Random Forest	75.66 ± 1.62	72.6 ± 2.28
	SVM (Linear)	**77.30 ± 1.80**	**75.94 ± 2.05**
	SVM (RBF)	77.22 ± 1.69	75.88 ± 1.96

It is noticeable that Word2Vec produced the least favorable results, in contrast to the other techniques for feature extraction, regardless of the classification approach employed. The best performance was attained by LightGBM, achieving an accuracy of 55.02% and an F1-score of 50.07%.

Even though the combination of Doc2Vec and LightGBM enhanced the top-performing Word2Vec outcome by 8.8% in accuracy and 10.9% in F1-score, we still deem this level of both metrics unsatisfactory. This increase can be attributed to the enhanced capacity of the generated Doc2Vec embedding in effectively encapsulating the contextual nuances of a given sentence. The unimpressive overall Word2Vec and Doc2Vec performances were not surprising, given the restricted quantity of annotated data and the content specificity, which can hinder sentence embedding.

On the other hand, the TF-IDF encoding exhibited overall superior performance compared to the Word2Vec and Doc2Vec, achieving an accuracy of 77.3% and an F1-score of 75.94% when coupled with the SVM (Linear) classifier. A better performance for TF-IDF over Word2Vec or Doc2Vec has been reported before for text classification tasks [Cahyani and Patasik 2021]. One of the reasons for such a superior performance could be due to TF-IDF's ability to capture the significance of domain-specific terms. This effectiveness is likely attributed to assigning higher weights to rare and informative terms. In contrast, Word2Vec and Doc2Vec may display reduced efficacy in capturing such nuances, especially due to the dataset not being large enough.

Figure 6 displays the confusion matrix for the best pairing, TF-IDF combined with SVM (Linear) classifier. It can be noted that most of the errors are concentrated in classes "5", "7", and "10", which are the classes with fewer samples. In addition, it is possible to observe that most confusions arise between class "3" or class "6" and other classes. The confusion surrounding class "3" is expected due to this class's large quantity of samples. Moreover, it is the class agents choose when unsure which option to select. Class "6", similar to class "3", is chosen when generic errors occur. Since both classes "3" and "6" are equally generic, agents choose one or another indiscriminately. The model also reflects this confusion, as presented in Figure 6. It is important to highlight that this result may encourage the business team to reformulate the available issue labels in the future.

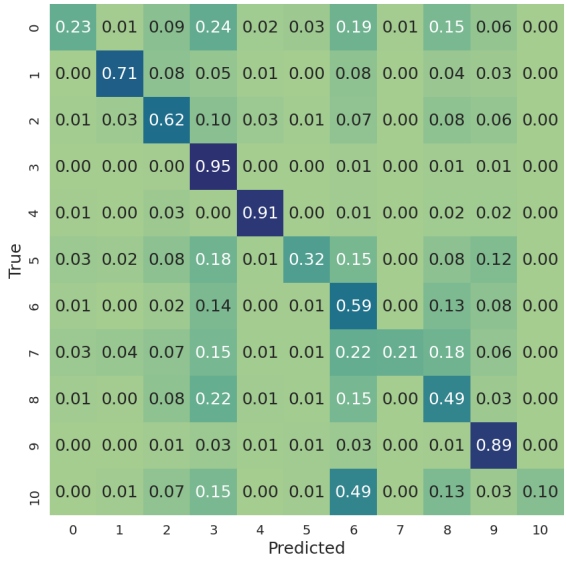

Figure 6. Average confusion matrix of the TF-IDF and linear SVM combination.

We employed the Kruskal-Wallis nonparametric test with Dunn's post-hoc test on the F1-score outcomes from the five outer loop iterations to verify the null hypothesis that the classifier and feature extractor pairings exhibit no significant difference between them. The Kruskal-Wallis test yielded a $p = 1.7 \times 10^{-5}$, along with a Kruskal-Wallis statistic value of 33.97, which is enough to reject the null hypothesis. Then, we apply Dunn's post-hoc test to determine which of the sample pairs are significantly different. Figure 7 shows the adjusted p-value from Dunn's test considering only TF-IDF; $*$ represents a p-value between 10^{-2} and 5×10^{-2}, $**$ is a p-value between 10^{-3} and 10^{-2}, and $***$ is a p-value between 10^{-4} and 10^{-3}. We can see that SVM (both variants), LightGBM, RF, and MLP are the best results, which are statistically similar and better than the others.

5.2. Monitoring and maintenance experiment

Prometheus works as a metrics aggregator, collecting all data received from the application requests and sent by the model responses. Subsequently, Grafana is employed to render such data visually. By employing Grafana, diverse panel types are seamlessly integrated into a dashboard interface, where Prometheus metrics find visualization through fitting panels incorporated via an API. This integration process facilitates the monitoring of the ML model. We use traffic simulation with Locust to feed the dashboard, which simulates real users interacting with the system.

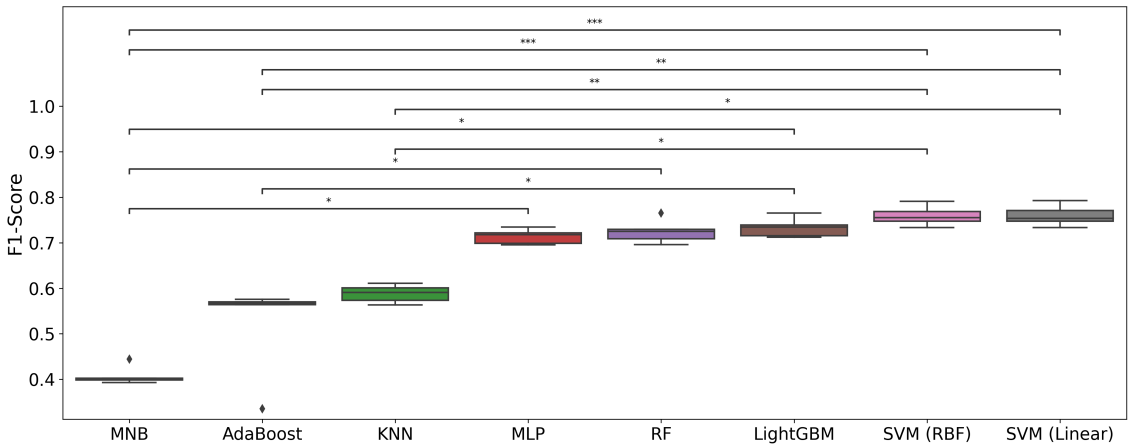

Figure 7. Kruskal-Wallis with Dunn's post-hoc test for multiple comparisons between classifiers when TF-IDF is used as a feature extractor. The box-plot graph illustrates the median F1-score in each outer-fold.

Figure 8 illustrates an exemplified dashboard arrangement. We can observe three main rows: i) accuracy and name of the model currently in production, ii) history of train accuracy, and iii) distribution of the predicted values. The prediction distribution is shown using a histogram, which resembles the counts presented by Figure 3, since the data for the request simulation was stratified. Using a bar plot in "History of Train Accuracy" makes it possible to note the metric improvement over time due to the newly labeled samples included in the train set. Finally, the third row can be used to monitor the best model deployed in production. As expected, we can see in the second row of Figure 8 that the overall accuracy increases over time, following the model update with new data from past requests.

Figure 8. Example of a Grafana dashboard arrangement layout comprising three primary rows.

We emphasize that the model update is automatic and does not require manual intervention. It is also worth mentioning that the entire double loop depicted in Figure

4 is executed after some predetermined time window, related to the availability of new labeled data. Thus, different learning algorithms with varying configurations of hyperparameter can be automatically selected from time to time based on their cross-validation performances, which includes the comparison with the model currently in production. It is also worth mentioning that in the case of massive data regimes, one should undersample the training data, possibly with a bias towards newer instances, or pursue incremental learning strategies to update the model without fully retraining it [van de Ven et al. 2022]. The latter is a promising direction for future investigations.

6. Conclusion

In this study, we have created and assessed a natural language processing system designed to assist in selecting an issue category in a text-based technical support platform. Following recent machine learning literature, besides handling model training and evaluation, we also focus on applying MLOps practices to automate a pipeline for the solution. The suggested workflow encompasses data processing, model training, and both monitoring and maintenance modules. The monitoring module enables the assessment of model performance, while the updating module leverages new data to periodically refine the model. The result is a cyclic workflow of automated steps to maintain the system in production.

Through experimentation and evaluation, we have provided evidence of the effectiveness of our approach. The static results showed that the optimal performance was achieved using a TF-IDF combined with SVM with the Linear kernel, yielding an accuracy of 77.3% and an average weighted F1-score of 75.94%.

Moreover, in a dynamic setting with simulated requests, the proposed automatic model retraining strategy increased the model accuracy over time, given additional data. This study demonstrated the validity and practical applicability of our methodology, offering valuable insights for the deployment of text classification tools in large organizations and dynamic real-world settings, especially the ones that require frequent updates. This research contributes to the ongoing advancement of automatic text classification and MLOps strategies, paving the way for more accurate and adaptive solutions in the future.

Future work includes improving the solution by incorporating metadata related to the textual data, such as information from the user who reported the issue. Furthermore, we intend to analyze how correcting the main issue category affects the support system workflow and the agents' interaction with it.

References

Alla, S., Adari, S. K., Alla, S., and Adari, S. K. (2021). What is mlops? *Beginning MLOps with MLFlow: Deploy Models in AWS SageMaker, Google Cloud, and Microsoft Azure*, pages 79–124.

Arias-Barahona, M. X., Arteaga-Arteaga, H. B., Orozco-Arias, S., Flórez-Ruíz, J. C., Valencia-Díaz, M. A., and Tabares-Soto, R. (2023). Requests classification in the customer service area for software companies using machine learning and natural language processing. *PeerJ Computer Science*, 9:e1016.

Borg, A., Boldt, M., Rosander, O., and Ahlstrand, J. (2021). E-mail classification with machine learning and word embeddings for improved customer support. *Neural Computing and Applications*, 33(6):1881–1902.

Cahyani, D. E. and Patasik, I. (2021). Performance comparison of tf-idf and word2vec models for emotion text classification. *Bulletin of Electrical Engineering and Informatics*, 10(5):2780–2788.

Cawley, G. C. and Talbot, N. L. (2010). On over-fitting in model selection and subsequent selection bias in performance evaluation. *The Journal of Machine Learning Research*, 11:2079–2107.

Essa, E., Omar, K., and Alqahtani, A. (2023). Fake news detection based on a hybrid bert and lightgbm models. *Complex & Intelligent Systems*, pages 1–12.

Gift, N. and Deza, A. (2021). *Practical MLOps*. O'Reilly Media, Inc.

Haq, M. A., Khan, M. A. R., and Alshehri, M. (2022). Insider threat detection based on nlp word embedding and machine learning. *Intell. Autom. Soft Comput*, 33:619–635.

Kaminwar, S. R., Goschenhofer, J., Thomas, J., Thon, I., and Bischl, B. (2021). Structured verification of machine learning models in industrial settings. *Big Data*.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Mäkinen, S., Skogström, H., Laaksonen, E., and Mikkonen, T. (2021). Who needs mlops: What data scientists seek to accomplish and how can mlops help? In *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*, pages 109–112. IEEE.

Nigenda, D., Karnin, Z., Zafar, M. B., Ramesha, R., Tan, A., Donini, M., and Kenthapadi, K. (2022). Amazon sagemaker model monitor: A system for real-time insights into deployed machine learning models. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3671–3681.

Paleyes, A., Urma, R.-G., and Lawrence, N. D. (2022). Challenges in deploying machine learning: a survey of case studies. *ACM Computing Surveys*, 55(6):1–29.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., and Dennison, D. (2015). Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, 28.

Studer, S., Bui, T. B., Drescher, C., Hanuschkin, A., Winkler, L., Peters, S., and Müller, K.-R. (2021). Towards crisp-ml (q): a machine learning process model with quality assurance methodology. *Machine learning and knowledge extraction*, 3(2):392–413.

Symeonidis, G., Nerantzis, E., Kazakis, A., and Papakostas, G. A. (2022). Mlops-definitions, tools and challenges. In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0453–0460. IEEE.

van de Ven, G. M., Tuytelaars, T., and Tolias, A. S. (2022). Three types of incremental learning. *Nature Machine Intelligence*, 4(12):1185–1197.

Wainer, J. and Cawley, G. (2021). Nested cross-validation when selecting classifiers is overzealous for most practical applications. *Expert Systems with Applications*, 182:115222.