Towards a low-code solution for monitoring machine learning model performance

Panagiotis Kourouklidis
University of York
British Telecom
panagiotis.kourouklidis@bt.com

Dimitris Kolovos
University of York
dimitris.kolovos@york.ac.uk

Nicholas Matragkas
University of York
nicholas.matragkas@york.ac.uk

Joost Noppen
British Telecom
johannes.noppen@bt.com

ABSTRACT

As the use of machine learning techniques by organisations has become more common, the need for software tools that provide the robustness required in a production environment has become apparent. In this paper, we review relevant literature and outline a research agenda for the development of a low-code solution for monitoring the performance of a deployed machine learning model on a continuous basis.

CCS CONCEPTS

• **Software and its engineering → Abstraction, modeling and modularity**.

KEYWORDS

software engineering, machine learning, concept drift, data drift, model monitoring

ACM Reference Format:
Panagiotis Kourouklidis, Dimitris Kolovos, Nicholas Matragkas, and Joost Noppen. 2020. Towards a low-code solution for monitoring machine learning model performance. In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20 Companion), October 18–23, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3417990.3420196

1 INTRODUCTION

In recent years, commoditisation of computing and digital storage resources as well as the development of new, so called machine learning, algorithms, has allowed businesses to perform tasks that used to require a prohibitive amount of human labour [5]. One of the basic tasks of machine learning is to provide methods for deriving descriptions of abstract concepts from their positive and negative examples [19]. These concept descriptions, that are also referred to as *models* in machine learning terminology, can subsequently be used to recognize previously unseen examples of the attained concept. With these machine learning models at hand, one can build a variety of systems, including, image classifiers, recommender systems, and price predictors.

In general, for the development of a system that incorporates machine learning capabilities, knowledge from several disciplines is required. The core of such systems is developed by data scientists, who perform the exploratory analysis on the available data and decide which algorithm is best suited for each use case. Beyond the core machine learning related tasks though, for the development of a production-ready system, additional tasks related to general software engineering need to be performed. These tasks usually include the provisioning and management of storage and computational resources as well as the development of data pipelines.

One way to reduce the software engineering effort required for building such systems, is by using tools that can automate parts of the process. According to market research firm Gartner, the use of such automation tools, usually called low-code application platforms, is a growing trend [18]. These platforms allow the user to develop applications with relatively little effort, compared to hand-written code, by incorporating a visual interface or a declarative domain specific language. This approach could also be applied for the development of machine learning-centric applications.

This paper's main aim is to outline a low-code architecture, that will enable data scientists to deploy algorithms, for monitoring a machine learning model's performance over time, without needing to provide explicit technical details.

The rest of the paper is organized as follows. Section 2 presents an overview of the field of software engineering for machine learning, by means of a literature review, for the purpose of putting our planned contribution in context. Section 3 introduces the key concepts relevant to model monitoring and a running example. Finally, in section 4 we outline our future research agenda.

2 STATE OF THE ART

In this section, we present the findings of a literature review conducted in the area of software engineering for machine learning. This is a growing area of scientific research that is concerned with the improvement of the software systems used to enable and streamline the delivery of machine learning capabilities to production-ready applications.

For the literature to be presented in the proper context, we need to introduce the workflow commonly used by teams that wish to

MODELS '20 Companion, October 18–23, 2020, Virtual Event, Canada

ACM ISBN 978-1-4503-8135-2/20/10…$15.00
https://doi.org/10.1145/3417990.3420196

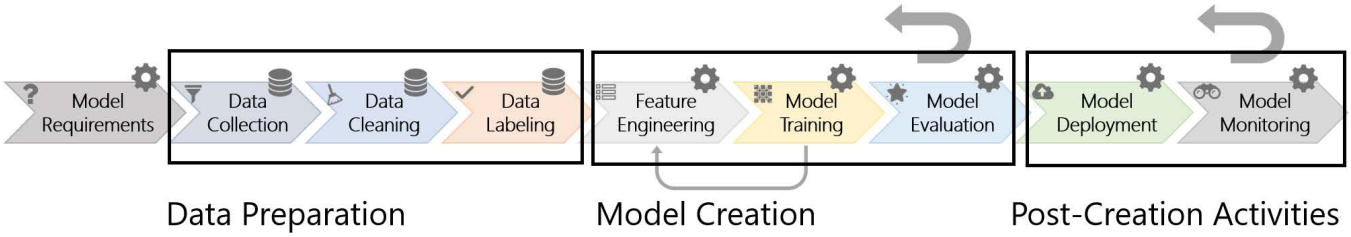

Figure 1: Machine Learning workflow (adapted from[1])

deliver an application enriched with machine learning capabilities. We will follow the nine-stage workflow as presented by Amershi et al. [1]. For the reader's convenience, the graphical representation of the workflow, that was used in the said paper can be seen in Figure 1, adapted to reflect the structure of our discussion. Instead of analysing each stage individually we will group them into 3 categories as follows: data preparation, model creation and post-creation activities. In Figure 1, the three black rectangles are added in order to illustrate which stages of the workflow are grouped together, for the purposes of our literature review.

2.1 Data Preparation

In contrast to classical computer science algorithms, machine learning solutions need vast amounts of data in order to be effective. Thus, a large portion of the machine learning workflow has to do with data-related activities. According to an internal survey conducted by Microsoft, the top ranked challenge in the field of machine learning, as perceived by employees working in it, is data availability, collection, cleaning, and management [1].

Due to the importance of data, organisations that integrate machine learning in their products, have invested substantial engineering resources in developing systems that can validate incoming data, as discussed in [3, 8, 12]. In these publications the authors present systems which enable users to specify the expected properties of the data that is to be received. Subsequently, the system automatically checks that the incoming data adheres to the specifications. The goal is to keep the quality of incoming data consistent and alert the engineers of any anomalies. By ensuring that all of the data is of high quality, it can then safely be used for training new models or fed into existing models for inference. In that regard, one could draw a parallel to software testing. In the same way, developers of traditional software products would like to test new code to make sure it doesn't introduce bugs in their code base, developers of machine learning systems should test new incoming data to make sure that they are consistent with what the system expects to receive.

2.2 Model Creation

The second group of activities in the machine learning workflow, has to do with the creation of the model artefact. These are perhaps the activities mostly perceived to be the core of machine learning but are usually just a small part of the overall system, in terms of code volume [14]. The model created at this stage is the result of an iterative process by which different aspects of the raw data, algorithms and input parameters for these algorithms are experimented with in order to find out which combination delivers the best performance. Keeping in mind that this experimentation might take place over several days and be performed collaboratively by several data scientists, it becomes evident that there is a need for a system that can keep track of the performance metrics of every combination of factors that was attempted.

Such systems have indeed been designed and are in use by teams that train machine learning models. The systems that we have encountered in the literature [11, 17] follow a similar approach. Every time a machine learning model is trained, the system stores a variety of metadata in a database for future reference. The metadata stored can include the dataset used for training and evaluating the model, the performance metrics achieved by the model when evaluated, the input parameters used for the training of the model and also custom fields that the developer wants to associate with that particular training run. These kind of metadata databases are usually private and access is given selectively to members of a specific team or company. One exception to this is the OpenML database which largely shares the same capabilities as described above but is open for anyone to contribute with new datasets, training runs etc. [16].

2.3 Post-Creation Activities

In a research setting, the machine learning workflow of the researcher commonly concludes with the training and evaluation of the model. If the produced model would achieve performance metrics superior to those of the state of the art, the results would be published for other researchers to be informed and build upon them. On the other hand, when applying machine learning techniques in a commercial setting, the goal is to incorporate the output of the model in a product that customers (internal or external) will use. For this reason, the model artefact will have to be somehow deployed so that it is reachable by the customer-facing part of the application. There are various tools that can facilitate the serving of machine learning models, such as TensorFlow-serving [7].

In addition to deploying the model, a strategy needs to be devised also for the monitoring of the model's performance over the course of time. There is no guarantee that the model's performance during its deployment will be consistent with its performance during evaluation. There are various reasons for that, such as, training based on non-representative samples, as well as the dynamic and ever-changing nature of the world. For this reason, it is essential that the performance of the model is monitored as to ensure the quality of its output on a continuous basis. As model monitoring

will be the main focus of this paper, the main concepts of it are presented in depth in the next section.

3 MODEL MONITORING

In this section, we are going to discuss two concepts that are important for model monitoring as they can affect a model's performance. These are, *data* and *concept drift*. We are going to provide the definition for these two core concepts as well as a practical example that clarifies the way in which they are relevant to model monitoring.

3.1 Data Drift

Data drift, also referred to as sampling shift [10], is observed when the statistical distribution of the incoming data, once the model is deployed, is different from that of the learning set. The reason behind this drift might be the sampling mechanism that collects the samples that comprise the learning set. If the sampling mechanism is not carefully designed, the learning set might be biased and not representative of the whole statistical population. Alternatively, the values of the input data might be dependent on a variable unknown to the data scientist. If that unknown variable changes, the distribution of the incoming data, from that point on, may change too. As a result, even if our learning set was representative of the whole statistical population at the time the samples were collected, that will no longer be the case and data drift will be observed. The presence of data drift does not necessarily invalidate the outputs of the model, but it is certainly a trigger for further investigation. Additionally, as it will be shown in section 3.3, the absence of data drift does not guarantee that the outputs are still accurate. On the other hand, because it can be detected directly from the inputs, it can be used as an indirect indicator of degraded performance, if additional information is not easily available.

3.2 Concept Drift

Concept drift, as first described by Schlimmer and Granger [13], is observed when the mapping that connects the input space to the output space of the examined domain, changes over time. This can cause the mapping extracted from the learning set to lose some of its accuracy. According to Widmer and Kubat [19] concept drift can be real or virtual, although it is not always possible to distinguish between the two. Concept drift is considered real when the change in the mapping is caused by a change by an unknown variable, also referred to as hidden context [19]. By contrast, in the case of virtual concept drift, nothing changes in the domain examined. In essence, the actual mapping extracted from the learning set was never accurate for the whole statistical population and that weakness was exposed when the model was deployed. Monitoring for concept drift, either real or virtual, is able to give us a definite answer on whether our deployed model is still accurate or not. The challenge is that, it is not possible to detect concept drift just by examining the outputs of a model. Additional information is required that can signal whether, or not, an output is accurate, for each respective input.

3.3 Concrete Example

A concrete and simple example is now presented in order to clarify the connection between concept drift, data drift and model monitoring. In our example, a telecommunications company is operating a call center where company representatives receive calls from the company's clients. For each interaction between a client and a representative, the company would like to know whether the client had a positive or a negative experience. For the sake of this example we assume that the only data available for each interaction is the duration that the customer was put on a waiting queue before talking to a representative, and the duration the client was actually talking to a representative before the call was terminated. Furthermore, we assume that on an average day the duration for which a client has to wait in the queue follows a normal distribution with a mean value of 10 minutes and a standard deviation of 3 minutes. We will assume the same for the duration that a client talks to a representative. Finally, we will assume that a client considers an interaction positive if the total duration of the waiting and service time is less than twenty minutes. In Figure 2 we can see a synthetically-created learning set, which follows the statistical distribution mentioned above. Each data point is represented by the waiting duration on the horizontal axis and the service duration on the vertical one. If the interaction was considered positive by the client the data point is represented by a '+' marker. On the other hand if the interaction was considered negative it is represented by a '|' marker. In this example we will assume that the company's data science team applied the right learning algorithm and was successful at attaining the concept. Subsequently the model was deployed and started classifying interactions as positive or negative. Finally, to ensure that the model functions as intended, the team is continuously monitoring the distribution of the incoming data and collecting feedback from the clients for each interaction in order to compare with the model's output. While monitoring the model's performance, four distinct scenarios can be observed.

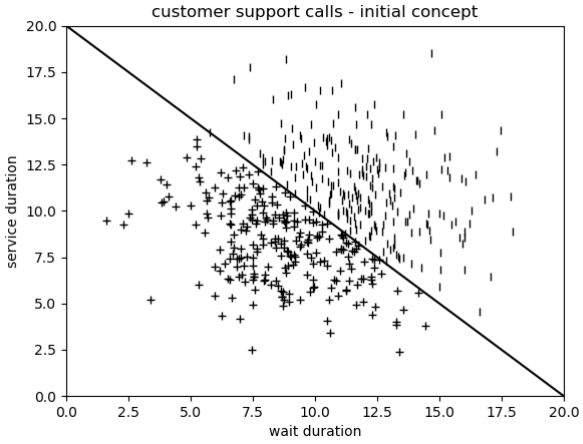

Figure 2: Initial concept

3.3.1 No Drift. The first scenario is that, incoming data follow the same distribution as the samples in the learning set and the output

of the model is accurate, according to the feedback received. This is the ideal case where the samples chosen to train the model is representative of the whole population and the underlying concept was successfully attained.

3.3.2 Concept Drift. The second scenario is that, incoming data follow the same distribution as the samples in the learning set, but the output of the model is no longer accurate. In this case we are observing real concept drift. As described above, concept drift can happen when a variable unknown to us, changes. In our example, this unknown variable, could be that the clients, over time, get used to waiting a little bit longer to be serviced. In this case, as can be seen by Figure 3, if our model is still classifying according to the old concept (dashed line), it would classify some interactions as negative, even though they are now positive, because they adhere to the new concept (solid line). In this case, the concept needs to be reattained by learning from new samples that adhere to it.

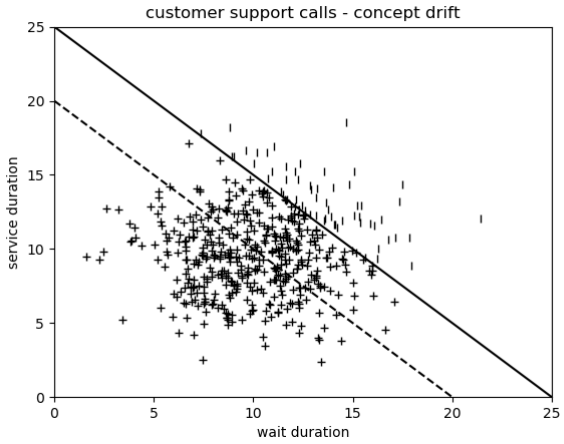

Figure 3: concept drift

3.3.3 Data Drift. The third scenario is that, incoming data do not follow the same distribution as the samples in the learning set, but the output of the model is still accurate. In our example this can happen, once again, due to a change to another unknown variable. If, for example, a large number of company representatives do not show up to work due to sickness on a particular day, we would expect the average waiting duration to increase. This does not mean that the customers' expectations for the waiting duration would change. In other words, they are still determining the interaction as positive or negative according to the same initial concept. In this scenario, that is depicted in Figure 4, the output of the model is still accurate despite the data drift. This means that the attained concept still holds and the model is still valid for the new incoming data.

3.3.4 Simultaneous Data and Concept Drift. The fourth and final scenario is that, incoming data does not follow the same distribution as the samples in the learning set and the output of the model is not accurate. In this case, we certainly have data drift and we also have either real or virtual concept drift. The case of virtual

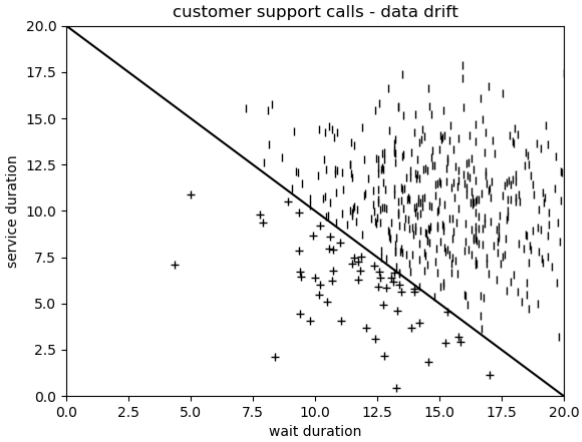

Figure 4: data drift

concept drift can be seen in Figure 5. In this Figure the dataset depicted is almost the same as the one in Figure 2 but with certain samples removed. This is done on purpose to simulate a biased sampling procedure. When this biased dataset is fed to our training algorithm, the concept attained (solid line) will deviate from what would have been attained, had we used representative samples (dashed line). As a result when the incoming data start following the actual distribution of the statistical population we will observe a number of misclassifications. Without knowledge of the actual concept and the fact that our sampling procedure was biased, these misclassifications could lead us to believe that we are observing data drift and real concept drift, as depicted in Figure 6. This is why it is said, that it is challenging to distinguish between real and virtual concept drift [19].

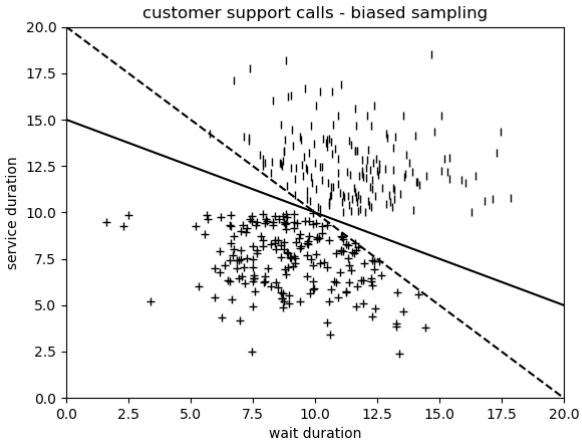

Figure 5: biased sampling

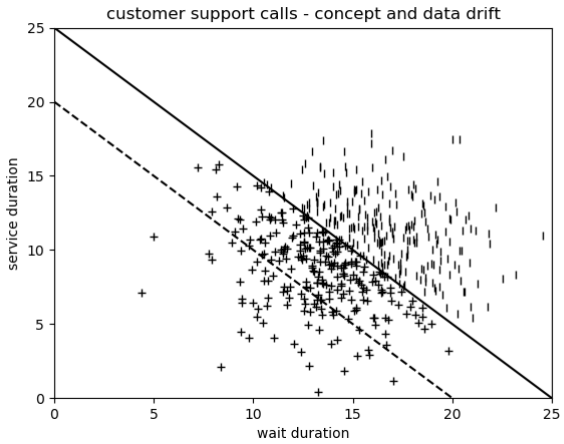

Figure 6: concept and data drift

4 RESEARCH DIRECTION

As was established in the previous section, the main factor that affects a model's performance after deployment, is the presence of data and concept drift. The theoretical aspects of ensuring that concept descriptions remain accurate, on an ongoing basis, despite the presence of data and concept drift, is a well researched topic and a number of relevant algorithms have been proposed [4, 9, 10, 13, 19]. On the other hand, work on software abstractions for model monitoring, is not as comprehensive.

In our view, choosing the right algorithms to combat the effects of drift on model performance, should remain the domain of data scientists. However, the increasing amount of complexity that modern computing architectures bring, could be abstracted away in a low-code solution. It is envisioned that our main contribution in this area, will be a domain-specific language, along with a reference implementation, that will allow data scientists to configure and reason about model monitoring without necessarily delving into technical details. An automation tool should then take this abstract description and execute all of the necessary transformations needed to arrive at a software artefact that is executable by the underlying computing infrastructure.

For the purposes of being more specific about the kinds of tasks that could potentially be automated using the envisioned low-code solution, we are going to present here, the tasks that machine learning practitioners would execute, to add monitoring capabilities to the call center example presented in section 3. We are going to split the tasks in three broad categories, analyse the technical details of each and also discuss how a low-code paradigm could benefit them. The three categories we are going to discuss in the following subsections are:

- Data capture.
- Algorithm execution.
- Responding to detected drift.

4.1 Data Capture

As mentioned in section 3, an important aspect of model monitoring is detecting data and concept drift. Regardless of the algorithm chosen, in order to calculate the data drift between the learning set and a set that includes recent incoming data, first of all we need to store the incoming data. Similarly, for the purposes of concept drift detection, we would like to store the output of the model so that we can compare it with the feedback received.

In the case of our call center example, in order to capture the incoming data and the corresponding model output and store for later use, a practitioner could work as follows. They could set up a reverse proxy server that can receive all the inference requests, forward them to the model serving infrastructure, receive the model output and send it back to the client that requested it. Then, since the reverse proxy server has access to both the incoming data and the model's output, it can persist it in a database, along with useful metadata such as date and time information. The benefit of this reverse proxy approach is that the model server itself doesn't need to be modified and thus this solution can be used with any model server.

Once we have the incoming data, which includes a customer's waiting and service duration, along with the model's prediction about whether the interaction was positive or negative, we also need to somehow receive the customer's feedback and store it alongside the rest of the data. One way to do it, in this specific example is to contact the customer via an automated text message or email, depending on what kind of contact information we have available. Some consideration is required here as to how the received feedback would be matched its corresponding data. That could be tackled by creating a unique identifier for each customer satisfaction prediction request, which can later be used by the feedback gathering mechanism to store the feedback with the rest of the data. The data capture system described here can be seen graphically in Figure 7.

Despite the fact that the call center example is just one possible scenario, we can observe several aspects that are applicable in general and can thus help us develop our low-code solution. The first such aspect is the mechanism by which the model's inputs and outputs can be collected. Regardless of the specifics of each use case, a low-code solution that offers a packaged, easy to use mechanism for data capture can bring significant value to its users. The second aspect is more difficult to generalize but equally as important as the first one. It has to do with the mechanism by which we collect feedback that indicates whether a model is performing well or not. The difficulty of generalization of this aspect is caused by the fact that the way we can collect feedback relies on the kind of model we are operating. For example, a product recommender system on an e-commerce website, can receive implicit feedback on its recommendations based on the user's behaviour (e.g whether they click on the recommendation or not). On the other hand, an image classifier has no way of receiving feedback without explicit human intervention. Nevertheless, we can always follow the tactic described in the call center example. In simple terms, we can let the application that consumes the model's output collect the feedback since it depends on the specifics of each use case. Our system can provide a unique identifier along with the model's output so that

if feedback is collected, it can be sent back to our system and be properly archived.

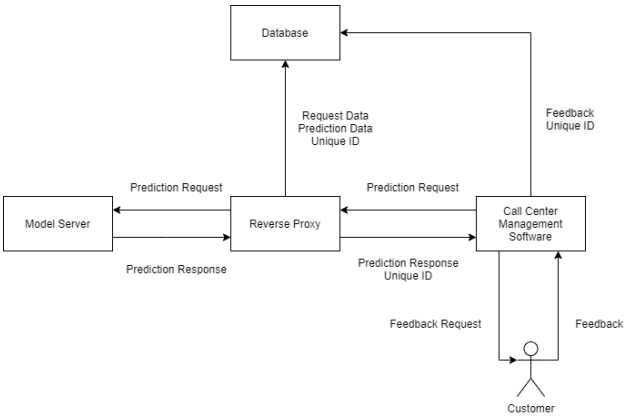

Figure 7: Data Capture

4.2 Algorithm Execution

After acquiring and storing the relevant data, the next task would be to execute the algorithms designed to detect data and concept drift. In our call center example, in order to detect data drift, a practitioner could choose to implement a two sample Kolmogorov - Smirnov test to check whether, or not, newer incoming data follow the same probability distribution as the learning set. In addition, the practitioner could choose to implement the STEPD concept drift detection method, as described in [6], that can indicate whether, or not, a change in the model's predictive accuracy over the newer samples, compared to the learning set, is statistically significant.

While the exact computations rely on the choice of algorithm, there is an aspect that is common amongst all algorithms. Whether one would like to calculate the statistical distance between two sets of data or use the collected feedback to decide if the model is still accurate, performance should always be an important consideration. This is where a low-code solution could bring value. Normally, when datasets become too large, algorithms need to be executed in a distributed manner. This usually means that a practitioner will have to reimplement their algorithm using a framework for distributed computing, such as Apache Spark[1]. One low-code approach to eliminate this reimplementation effort would be to develop a high-level domain-specific language (DSL), that can be used by data scientists to describe drift detecting algorithms, and then transpile the DSL code to framework specific code that can be executed by an organisation's computing infrastructure.

4.3 Responding to Detected Drift

The last part that needs to be designed might not be as technically challenging as the previous ones but it is just as important. Responding to the results of the previous step might be as simple as automatically sending an email to the relevant stakeholders or it might be more complex such as triggering a model retraining process. In our call center example, if data drift is detected (e.g the wait duration suddenly spiked), an email should be sent to the team responsible for the system so they can further investigate. On the other hand, if concept drift is detected, then the model can be automatically retrained with the most recent data. Once again, this process can be streamlined by the use of a low-code solution that can eliminate large parts of the engineering effort required.

4.4 Evaluation Plan

In order to evaluate the efficacy of our low-code architecture we plan to work as follows. Once the theoretical aspects of the architecture are finalized, we will proceed with the creation of a reference implementation. Having an implementation will allow us to work with data science practitioners inside British Telecom, that can help us empirically evaluate the value that our proposed solution brings on various dimensions, such as faster deployments and reduction of manual monitoring efforts.

For the evaluation of the proposed low-code architecture, the reference implementation will play a critical role. For this reason, even though the technical details of the implementation are still under consideration, we bring forth a set of requirements that the implementation needs to fulfil. These are as follows:

- Compatibility with various model serving infrastructures.
- Not inducing a large performance penalty when intercepting the inputs and outputs of a model in production.
- Ability to cope with large data volumes both in terms of storage and computing.
- Having enough flexibility so that data scientists can easily implement drift detection algorithms.
- Incorporate different ways that feedback on a model's performance can be collected.

5 RELATED WORK

Since the use of machine learning techniques in commercial products has become prevalent, various companies have sought out to develop tools that can streamline parts of the machine learning workflow. Unfortunately, because access to these tools can yield commercial advantages, some companies chose to keep their tools closed source. Thus, we don't know much about them, except for what the companies choose to share. Such tools include Facebook's FBLearner[2] and Uber's Michelangelo[3]. Fortunately, some such system are available to the public under open-source licenses and are described below.

The first software tool examined was MLflow [20]. MLflow's focus is on metadata tracking as well as improving experiment reproducibility. Essentially, it implements the concepts that were found in the literature, concerning tracking the parameters used and performance metrics of trained models for later comparisons. Its main advantage is that it is easy to use alongside the most popular machine learning frameworks, and additionally, it offers an intuitive user interface. On the other hand, MLflow cannot be considered an end-to-end tool as there are various aspects of the machine learning workflow that are not covered, such as data preparation tasks or model monitoring.

[1] https://spark.apache.org/

[2] https://engineering.fb.com/core-data/introducing-fblearner-flow-facebook-s-ai-backbone/

[3] https://eng.uber.com/michelangelo-machine-learning-platform/

The second tool examined was Tensorflow extended (TFX) [2]. It is developed by Google and promoted as an end-to-end tool for machine learning workflows. Indeed, thanks to the extensive Tensorflow ecosystem that it leverages, TFX can support a wide array of use cases. The tool wraps the capabilities of various libraries in the Tensorflow ecosystem in standard, ready to use components. The users can choose which components they need and use an orchestrator such as Apache Airflow[4] to schedule their execution as needed. The tool's most useful feature is that it eliminates a lot of the manual effort needed to set up pipelines of tasks where the output of one task is the input of the next. Its main disadvantage is that it only supports Tensorflow core as the framework used for the model training step of the workflow. In terms of model monitoring, it supports data drift detection but only for categorical features and only one distance metric is offered.

The third tool examined was alibi-detect by Seldon [15]. It is an open source library that offers numerous algorithms for data drift detection as well as outlier and adversarial detection. While the library itself doesn't provide additional functionality that can automate drift detection, it can be used as a component of a larger monitoring system.

Other systems that are worthy of mention, even though they are not open source and thus cannot be thoroughly examined, are Amazon's SageMaker[5] and Microsoft's Azure ML[6]. Both of these platforms offer functionality that can be used to detect data drift on incoming data. The major disadvantage of these platforms is that since they are proprietary, they cannot be easily extended with new functionality. In addition, organizations that invest a lot of engineering effort on building their products on top of these platforms are in risk of vendor lock-in.

6 CONCLUSIONS

In this paper, we have given a brief overview of the field of software engineering for machine learning. The recent rise in popularity of machine learning techniques, has resulted in quick growth in this relatively new field. After reviewing the relevant literature, we have given definition for two core concept in the area of model monitoring where we will pursue a contribution. In addition we have given practical examples as well as an overview of our research direction. Future work will focus on proposing relevant low-code solutions, as well as evaluating them, through empirical experimentation with reference implementations.

ACKNOWLEDGMENTS

This paper disseminates results from the Lowcomote project, that received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 813884.

REFERENCES

[1] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald C. Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: a case study. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019*, Helen Sharp and Mike Whalen (Eds.). IEEE / ACM, 291–300. https://doi.org/10.1109/ICSE-SEIP.2019.00042

[2] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*. ACM, 1387–1395. https://doi.org/10.1145/3097983.3098021

[3] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2019. Data validation for machine learning. In *Conference on Systems and Machine Learning (SysML). https://www. sysml. cc/doc/2019/167. pdf.*

[4] Ryan Elwell and Robi Polikar. 2011. Incremental Learning of Concept Drift in Nonstationary Environments. *IEEE Trans. Neural Networks* 22, 10 (2011), 1517–1531. https://doi.org/10.1109/TNN.2011.2160459

[5] Kim M. Hazelwood, Sarah Bird, David M. Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, and Xiaodong Wang. 2018. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24-28, 2018*. IEEE Computer Society, 620–629. https://doi.org/10.1109/HPCA.2018.00059

[6] Kyosuke Nishida and Koichiro Yamauchi. 2007. Detecting Concept Drift Using Statistical Testing. In *Discovery Science, 10th International Conference, DS 2007, Sendai, Japan, October 1-4, 2007, Proceedings (Lecture Notes in Computer Science, Vol. 4755)*, Vincent Corruble, Masayuki Takeda, and Einoshin Suzuki (Eds.). Springer, 264–269. https://doi.org/10.1007/978-3-540-75488-6_27

[7] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. 2017. TensorFlow-Serving: Flexible, High-Performance ML Serving. *CoRR* abs/1712.06139 (2017). arXiv:1712.06139 http://arxiv.org/abs/1712.06139

[8] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2018. Data Lifecycle Challenges in Production Machine Learning: A Survey. *SIGMOD Rec.* 47, 2 (2018), 17–28. https://doi.org/10.1145/3299887.3299891

[9] Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. 2015. Credit card fraud detection and concept-drift adaptation with delayed supervised information. In *2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, July 12-17, 2015*. IEEE, 1–8. https://doi.org/10.1109/IJCNN.2015.7280527

[10] Marcos Salganicoff. 1997. Tolerating Concept and Sampling Shift in Lazy Learning Using Prediction Error Context Switching. *Artif. Intell. Rev.* 11, 1-5 (1997), 133–155. https://doi.org/10.1023/A:1006515405170

[11] Sebastian Schelter, Joos-Hendrik Boese, Johannes Kirschnick, Thoralf Klein, and Stephan Seufert. 2017. Automatically tracking metadata and provenance of machine learning experiments. In *Machine Learning Systems Workshop at NIPS*. 27–29.

[12] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Bießmann, and Andreas Grafberger. 2018. Automating Large-Scale Data Quality Verification. *Proc. VLDB Endow.* 11, 12 (2018), 1781–1794. https://doi.org/10.14778/3229863.3229867

[13] Jeffrey C. Schlimmer and Richard H. Granger. 1986. Incremental Learning from Noisy Data. *Mach. Learn.* 1, 3 (1986), 317–354. https://doi.org/10.1023/A:1022810614389

[14] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett (Eds.). 2503–2511. http://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems

[15] Arnaud Van Looveren, Giovanni Vacanti, Janis Klaise, and Alexandru Coca. [n.d.]. *Alibi-Detect: Algorithms for outlier and adversarial instance detection, concept drift and metrics.* https://github.com/SeldonIO/alibi-detect

[16] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luís Torgo. 2013. OpenML: networked science in machine learning. *SIGKDD Explorations* 15, 2 (2013), 49–60. https://doi.org/10.1145/2641190.2641198

[17] Manasi Vartak, Harihar Subramanyam, Wei-En Lee, Srinidhi Viswanathan, Saadiyah Husnoo, Samuel Madden, and Matei Zaharia. 2016. ModelDB: a system for machine learning model management. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Carsten Binnig, Alan Fekete, and Arnab Nandi (Eds.). ACM, 14. https://doi.org/10.1145/2939502.2939516

[18] P Vincent, K Lijima, Mark Driver, Jason Wong, and Yefim Natis. 2019. Magic Quadrant for Enterprise Low-Code Application Platforms.

[4] https://airflow.apache.org/

[5] https://docs.aws.amazon.com/sagemaker/

[6] https://docs.microsoft.com/en-us/azure/machine-learning/

[19] Gerhard Widmer and Miroslav Kubat. 1993. Effective Learning in Dynamic Environments by Explicit Context Tracking. In *Machine Learning: ECML-93, European Conference on Machine Learning, Vienna, Austria, April 5-7, 1993, Proceedings (Lecture Notes in Computer Science, Vol. 667)*, Pavel Brazdil (Ed.). Springer, 227–243. https://doi.org/10.1007/3-540-56602-3_139

[20] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Fen Xie, and Corey Zumar. 2018. Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Eng. Bull.* 41, 4 (2018), 39–45. http://sites.computer.org/debull/A18dec/p39.pdf