Using monitoring to improve model performance

Atanas Marinov (4946251), Jeongwoo Park (4773543), Jahson O'Dwyer Wha Binda (4772288), Eduard Klein Onstenk (4930894)

ABSTRACT

Release Engineering is a rapidly growing subject in the field of Software Engineering. The agility it provides is known to be a major factor in the development of the IT industry. At the same time, the field of Machine learning is experiencing a significant rise in popularity. It is being utilized in a wide range of applications in the modern software systems. Still, their combination is a relatively new topic of interest for the scientists, and there is little research about it. Thus, the goal of this project is to show how Machine Learning Applications can be supported by using the principles of Release Engineering and also our unique extension on Monitoring. Machine Learning models are not perfect, and they can cause problems occasionally. To easily and quickly detect the weaknesses of the Machine Learning model, an advanced monitoring system beyond the basic features is developed, which includes metrics of the models and presents the distribution of the tags. This paper looks into the functionality and details of the basic Release Engineering skills and the additional Monitoring implementation. To accomplish this, StackOverFlow Prediction Model developed by @partoftheorigin was used in our experiment.

1 INTRODUCTION

Machine learning software projects differ from other software projects in several key factors, such as the need for updating and retraining the model and the large quantities of data that are needed. This begs the question: *What release engineering practices and concepts are needed for implementing a machine learning project*? A given baseline project [1] is extended as part of the course "Release Engineering for Machine Learning Applications" at Delft University of Technology. The project is divided up into two parts, the basic extensions and the extension proposal. There is a list of basic extensions that have to be added to the baseline project which consists of Pipeline Management, Automated versioning, Containerization, Orchestration and Monitoring. Then the team came up with the idea to extend the current structure by further enhancing the monitoring system. To improve the lives of expected users, namely, developers and data scientists, the team developed a Grafana dashboard that visualizes variety of metrics. Figure 1 presents an overview of all components implemented.

Let us split the course into Release Engineering and Machine learning. "[..] release engineering is the difference between manufacturing software in small teams or startups and manufacturing software in an industrial way that is repeatable, gives predictable results, and scales well. These industrial-style practices not only contribute to the growth of a company but also are key factors in enabling growth." [4]. Second, machine learning as defined by NVIDEA: "Machine learning at its most basic is the practice of using algorithms to parse data, learn from it, and then make a determination or prediction about something in the world." [2]. Machine learning has been around for decades, but the recent progress in amount of data, cheap storage and advances in computing speeds have massively increased its usefulness and impact. An increasing amount of software utilises machine learning every day across many sectors.

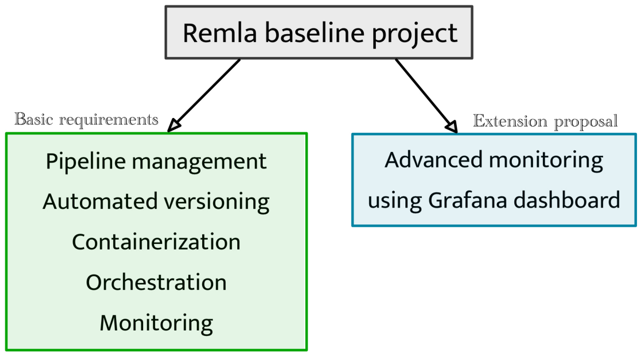


Figure 1: An overview of the components implemented in this paper. First the basic requirements on the left, and the extension proposal on the right.

Three aspects justify the importance of this paper's topic: importance of release engineering for software quality, predictability and results. The recent growth in machine learning applications, and the gap in research on machine learning and software engineering combined. According to Openja [12], there is not much empirical evidence on testing strategies by ML/ RE engineers as they struggle to implement proposed practices. This is not ideal given its utilisation in safety-critical fields such as health care and autonomous driving. Furthermore, there is much research on release engineering and machine learning individually, or even on applying machine learning on release engineering problems [6, 7, 9]. However little research could be found on release engineering for machine learning applications, which would differ from other applications because of the need for retraining or updating the model for example.

Given that many relevant practices already exist, the objective of this paper is to look at our implementation of release engineering process for the chosen machine learning application. User Interface between the model and the user and the visualisation of the monitoring system were improved to easily generate and retrieve the data needed for the machine learning engineer or data scientists.

The rest of this document is structured as follows. Next, in Chapter 2 some Related Work and topics are reviewed and discussed to provide the reader with some context on how much research opportunities are available and related to this paper. Then in Chapter 3

[1]https://github.com/luiscruz/remla-baseline-project

[2]https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/

System Design, Project structure, Pipeline management and Version Control, Container Orchestration and Data validation are explained as implemented in this project. Chapter 4 'Added Improvements' contains the extension improvements. Chapter 5 provides some depth to the implications of the changes proposed and implemented. Chapter 6 is a conclusion followed by suggestions for future work in Chapter 7.

2 RELATED WORK

To get to know about the precedent research on both release engineering and machine learning, the literature regarding the two topics have been extensively studied. Although there are many research on each one, studies of their overlap are not common. Considering this, the body of related work is subdivided into three main categories (non-exhaustive): the study of release engineering, software engineering for machine learning applications and machine learning testing.

The study of release engineering for machine learning applications is a sub-field of the study of release engineering as a whole. "[..] the release engineering process is the process that brings high quality code changes from a developer's workspace to the end user, encompassing code change integration, continuous integration, build system specifications, infrastructure-as-code, deployment and release." [3]. The authors, Adams and McIntosh, argue the importance of release pipelines resulted in much progress lately, however also identify many opportunities for further research that could solve problems software engineers face, as well as providing a convenient checklist for any release engineer. Both [2, 3] agree that numerous practices have been adopted and improved overall release engineering, but that plenty of problems still require researched solutions.

Motivated by recent progress in machine learning applications, software engineering specifically for machine learning has gained a lot of interest. A case study on the software development of AI features designed for customers by various Microsoft teams was performed to find out how the process is adjusted and what common practices are across teams [5]. Their findings include a list of best practices which "address issues fundamental to large-scale development and deployment of ML-based applications". Khomh et al. take a look at the possible future of software engineering for machine learning applications in [10]. They argue many shortcomings are the result of the changing patterns in development caused by among other ML. They also discuss another reason for failure, which is that the SE community and ML community widely researched their problems but a rift between the communities is caused by failing to study both combined.

Finally, one can consider testing for machine learning. Testing is vital for any release engineering pipeline, yet this process is different for machine learning applications. A systematic literature review [1] was conducted on the testing of scientific software aiming to identify current problems and solutions from 62 studies. A comprehensive survey of testing for the field of machine learning specifically covers 144 papers [13]. To address the ML specific issues that arise when testing, Google provides a metric [8] to quantify how much testing and monitoring is enough, awarding points for each task and automating for each task plus a description for the quality of testing for your system given the amount of points. [11] provides an approach to software testing for machine learning applications which do not yet provide correct labels to have a test oracle in place.

3 SYSTEM DESIGN

This section describes and discusses both the basic and advanced requirements (see Figure 1), as well as providing some details about the implementation.

3.1 StackOverFlow Prediction Model

The model used in this paper predicts tags for posts from StackOverFlow with multilabel classification approach, originally developed by @partoftheorigin. The training set consists of the 100001 post titles and the corresponding tags selected by the original author. Two models were used, Bag-of-Words and TF-IDF models. The detailed functionalities and explanations can be found in the provided Jupyter Notebook. For the implementation of the machine learning models, Numpy, Pandas, Scikit-learn and NLTK were used.

3.2 Project Structure

The original code provided was in Jupyter notebook (.ipynb) format. However, this hinders the project in several ways. The programming style can be very subjective, but the blog post by Alexander Mueller [3] suggests five drawbacks of using Jupyter notebook. First of all, Jupyter notebook is not ideal for code versioning. It does not provide IDE or specific linting, and also very hard to test. It is based on nonlinear workflow, and it does not support asynchronous tasks in high quality. Therefore, to implement other mandatory basic requirements, transforming Jupyter notebook code to the Python project module was necessary. Keeping future extensions and scalability of the project in mind, five folders were created, *analysis*, *classification*, *evaluation*, *preprocessing* and *transformation*, for a better structure and overview.

(1) **Preprocessing** is where dataset is retrieved and processed into a certain format required by the transformation.
(2) **Transformation** is where the text is vectorised and transformed in a way suitable for the model training.
(3) **Classification** is where train code is. There is a possibility that different models require different structure of training process, or cross-validation can require different training style.
(4) **Evaluation** is a place for code that assists evaluating the model such as Confusion Matrix, Precision, and etc. This allows evaluation process to be separate from the training process.
(5) **Analysis** currently contains the analysis of the relationship between tags.

'training_classifier_mybag.py' and *'training_classifier_tfidf.py'* are file where training the models is executed and the model is saved. *'serve_models.py'* opens the server for the trained model so that these models can be used the predict the input text.

[3]https://towardsdatascience.com/5-reasons-why-jupyter-notebooks-suck-4dc201e27086

3.3 Pipeline Management

Two checks, **Linter** and **PyTest**, have been added to Github Actions to manage the code quality. These two are activated in all the branches for every commit so that every code that is pushed to the Github always maintain certain level of code quality.

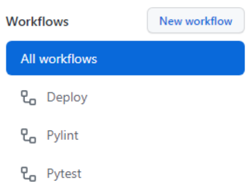


Figure 2: Github Action Workflows

3.3.1 *Automated Linting.*

Pylint is a static code analyser for Python 2 or 3, which analyses the code without actually running it. It checks for errors in the code, enforces a coding standard which can be adapted by the user, looks for code smells, and make suggestions about how the code could be refactored [4].

With the help of Pylint, the code can be maintained coherently in terms of quality, and also can reduce possibilities of introducing any error. To add automated linting to our project, *'pylint.yml'* is added to the .github/workflows. Pylint is triggered for every commit.

3.3.2 *Automated Testing.*

PyTest [5] is a testing framework which allows developers to write and run tests on their python code. With PyTest a failing pipeline can be triggered when not all tests have passed, or the code is not able to build on a recent commit. This lets the development team know that there is something wrong with their code. PyTest also allows for the display and enforcement of code coverage rules, so that if code is not tested up to a certain threshold the pipeline will fail which prompts the development team that more tests are necessary. Though this was not implemented in this project as there was not a meaningful amount of tests to reach a significantly high code coverage. We did write some of our own tests for key functions in the code base like parsing functions.

3.4 Version Control

3.4.1 *Semantic Versioning, Building and Deployment.*

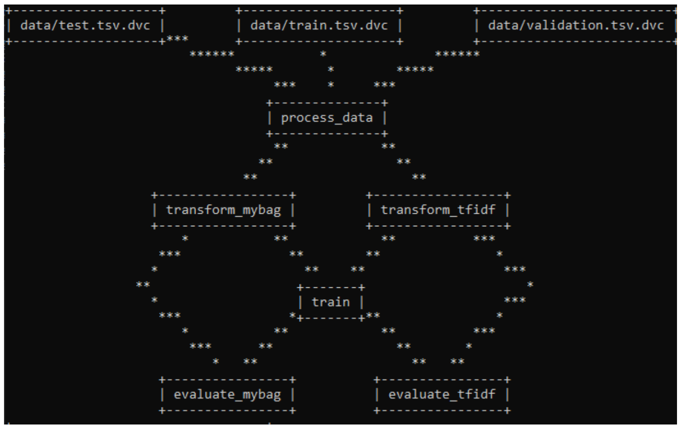
The use of Git and Github is common when building software. The practice of continuous development means that software is constantly changing when using these tools. For users of the software it is good to be able to have access to builds of the code base that are known to be ready for general use. With the use of Git Tags and Github actions the automatic build and deployment of versioned code was implemented. After a commit is tagged with a release number structure e.g. *v0.0.1* , Github actions are used to build and deploy the current state of the code-base. Users can then access these release builds while the development team continues working on the software.

[4]https://pylint.pycqa.org/en/latest/index.html
[5]https://docs.pytest.org/en/7.1.x/

3.4.2 *Data Version Control.*

Any resource that is continuously updated needs a system that can track changes. For code, Git is often used. Simply speaking, Data Version Control (DVC) is Git for data. When the data is modified or added frequently, multiple versions of data are created. DVC is especially useful for this case by tracking these changes. Another advantage is that 'dvc repro' regenerates data pipeline results defined by the stages listed in *dvc.yaml* [6].

To share the dataset with other Google Driver remote users, Stackoverflow data is stored in Google Drive remotely. This allows to save space on our local environment. DVC also provides an implementation with Google Cloud, so for a complex, big size projects, DVC can still be used together with the cloud service. As the dataset is relatively small and the project is lightweight, we chose to use Google drive.



```
+-------------------+          +--------------------+         +-------------------------+
| data/test.tsv.dvc |          | data/train.tsv.dvc |         | data/validation.tsv.dvc |
+-------------------+***       +--------------------+         +-------------------------+
                        ******            *              ******
                              *****        *        *****
                                   ***     *     ***
                                   +--------------+
                                   | process_data |
                                   +--------------+
                                    **            **
                                  **                **
                                **                    **
                   +-----------------+          +-----------------+
                   | transform_mybag |          | transform_tfidf |
                   +-----------------+          +-----------------+
                        *           **            **          ***
                    ***               **        **               *
                    *                   **    **                  ***
                 **                    +-------+                     *
                    *                  | train |                  ***
                    ***               *+-------+**               *
                        *           **            **          ***
                         ***      **                **       *
                            *   **                    **   **
                    +----------------+           +----------------+
                    | evaluate_mybag |           | evaluate_tfidf |
                    +----------------+           +----------------+
```

Figure 3: Data Version Control Directed Acyclic Graph

Looking at the DAG in Figure 3, process_data stage reads in the data and pre-process the data. Transform_model transform those preprocessed data into vectors. These transformed vectors are then used in the train stage. Train stage trains the vectors for both models as we need both trained models. The model is then evaluated.

Our system does not require active usage of DVC, but considering the future extension of our dataset and our system, it is definitely required.

3.5 Containerization and Orchestration

3.5.1 *Containerization.*

Similar to many of the other Release Engineering practices, containerization has become an increasingly popular methodology. It shares a common idea with the principle of visualization. They both try to encapsulate a service so that it can run uniformly and consistently across the infrastructure. The difference is that instead of relying on virtual machines, containerization operates with containers. For our project, we identified two distinct services, namely model hosting, and Web application. The model hosting serves the model to the outside. It gets a StackOverflow title as an input, performs inference with both models, and outputs

[6]https://dvc.org/

their predictions. The Web application container includes the code needed for the graphical user interface and the user interactions. It also provides the necessary metrics to the monitoring system - Prometheus. Both of them are implemented using Docker.

3.5.2 Orchestration. When scaling vertically, the number of components comprising the system increases. The goal of the orchestration is to coordinate and manage those resources and services. To solve those needs, the team decided to use Kubernetes as an orchestrator. It provides many features which facilitate the deployment process. Some of them are simple service discovery, load-balancing, storage orchestration, and self-healing. It also allows to easily add services such as monitoring with Prometheus and visualization with Grafana.

3.6 Data Validation

Validation of data is done by using Tensorflow Data Validation (TFDV). It has three main functionalities: the computation of metadata (descriptive statistics), the ability to infer a schema from training data and the ability to detect data anomalies. The statistics are useful to see at a glance what features are present and what values their distribution takes. A schema reports anticipated properties of data, such as datatype and domains of the features. Using such an inferred schema, which describes certain expectations, data anomalies can automatically be found by checking whether the data obeys the schema.

4 ADDED IMPROVEMENTS

To further improve the engineering methods of working on these Machine learning models, it was decided to create a dashboard that illustrates and compares in-depth the performance of the models in the real world. To do this, first, we needed to create a more extensive user interface for users to interact with the models. These interactions were recorded and used to generate data. Finally, this data gets processed and transformed into valuable metrics that are fed to the improved dashboard to display to the Machine Learning engineers.

4.1 Improved User Interface For Model Correction

To collect data on how well these models perform in the real world, an improved user interface must be created.

The current pool of models is still relatively small and contains only two machine learning models, Bag-of-Words and TF-IDF. To allow for real world performance of the models to be tracked, the User Interface (UI) has an option which allows users to enter a title to be labeled by the models. After the models have made their predictions, a union set of tags generated by the two models is displayed to the user. The user can then click on tags they believe are incorrect which are then marked in red. The user is also given the option to add a list of tags they believe the models should have predicted.

They can finally submit the results on how well the models have predicted the StackOverflow tags. An example of selection of a possibly incorrect tag in red is shown in Figure 4.

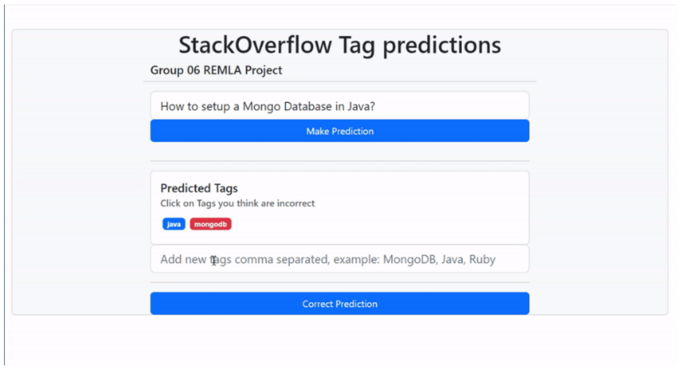

StackOverflow Tag predictions
Group 06 REMLA Project
How to setup a Mongo Database in Java?
Make Prediction
Predicted Tags
Click on Tags you think are incorrect
java mongodb
Add new tags comma separated, example: MongoDB, Java, Ruby
Correct Prediction

Figure 4: User interface with mongoDB tag selected as incorrect

After, the the user presses the button to submit, the tags which the models were able to correctly and incorrectly predict are separated for each model and sent to be displayed by the statistics dashboard.

The UI was improved with a combination of JQuery and Bootstrap for styled elements. Since it is basic we decided a light weight solution like this would be preferred over a full front end framework like React.

4.2 Improved Grafana Dashboard for Deployment and Model Statistics

4.2.1 Motivation. It is hard to assess the model status without any quantifiable metrics. Judging it manually by examining its output is time-consuming and can be swayed by human error. Moreover, the trained model might have shown an outstanding result during training but actually experience an unexpected performance drop when deployed to the real world. That could be caused by many different factors, for example, the use of a highly-unbalance dataset or overfitting due to too much training. Some of the causes can be very complex and deeply rooted, thus, making them hard to notice for the data scientists. Additionally, even if the model is well-trained, it is possible for its performance to diminish due to outside factors. There are many of those, for instance, the addition of a new class in multiclass classification. Nevertheless, a monitoring system with detailed and carefully thought-out metrics can show these to the data scientists, thus, preventing undesired results a lot quicker. Motivated by these reasons, the team proposed to create an advanced monitoring system for the data scientists.

4.2.2 Implementation. To set up the monitoring and the dashboard for our project, Prometheus and Grafana were utilized. Briefly speaking, Prometheus is a monitoring system that can store time-series data metrics, and Grafana visualizes these collected data. To handle the incoming data, a server has been set up using Java and Spring. The data collected in the front-end is sent to the back-end server by Ajax. TagController.class hands over these data to MainController.class where the metrics are generated and mapped to "/metrics". The metrics are written in Prometheus format so that Prometheus can retrieve them correctly.

There are two main types of metrics utilized. The first ones are standard metrics, such as accuracy. They illustrate the performance of the models and can be used to indicate if something is going wrong with them. The second type summaries and tag distributions. Their goal is to identify the cause of the problem. Take the number of tags missed by the models as an example. If the data scientists notice a high value of those, they can understand that the models cannot capture the notion of a particular tag and would likely try to include more such data in the training set.

The Grafana dashboard consists of three rows. The first one is shown in Figure 5. It is meant to provide a comparison between the two models in use and summarize where both of them struggle. On the left, it contains a graph that puts side-by-side the accuracies of both models. It is used to show which of them performs generally better. Despite being extremely one-dimension as a metric, we decided to use only in this row to prevent it from over-crowding (more metrics about the performance of each are included in the detailed rows 2 and 3). The right side of the dashboard contains two tables. The first one outlines the general tag distribution. It provides an overview of how the amount of the different tags compares. The second one aims to show which of the tags are most frequently missed by the models and inputted by the users. Those are the tags that are most problematic and hence important to the monitoring. The table tries to point out more attention to them.

The second and the third rows are conceptually identical but look at different models. Their goal is to show in detail how the models perform. A part of the overview of the Bag-of-words model is illustrated in Figure 6. These rows are designed such that they keep the first type of metrics to the left and the second type to the right. The first graph on the row shows the overall number of predictions. Next to it stands a note which depicts the achieved performance during training. Underneath those are situated a gauge and a time series graph for each of the metrics. The purpose of the gauge is to quickly see the current value of the metric, while the time series provides an overview of the history. Four metrics are employed - accuracy, precision, recall, and F1 score. Considering that the problem is multi-class, we decided to use a weighted average as an aggregation scheme when computing the metrics. All metrics are classical, but we decided to use them namely because of the fact that they are proven in time. Among those, accuracy is the most basic one. It shows the portion of correct predictions compared to the total predictions. Precision and recall are used to provide another dimension to the performance. Precision shows what proportion of the predicted positives are truly positive, while recall illustrates what proportion of the actual positives are correctly classified. The issue with them is that they are connected, and it is hard to maximize both of them simultaneously. Thus, we decided to also include F1 score, as it is good at minimizing both the false positives and false negatives.

5 IMPLICATION

The advanced UI and monitoring systems were set for the ease of data scientists' work. How exactly does this affect the data scientists?

The advanced UI allows to easily imitate the real-life use of the models. By setting up the buttons and making a text field to input additional tags, data scientists themselves can easily create the data and also retrieve the metrics automatically.

As the metrics are visualized on the Grafana dashboard, data scientists can easily see the trend of the data. It is hard to observe the pattern while reading an endless amount of numerical values which are poorly ordered in one excel spreadsheet. The visualization of these numerical metrics certainly reduces time spent to read the trend and also decreases time wasted on debugging the model. Data scientists can quickly notice that there is a weakness in the model using the metrics type one (as defined in the previous section). Then they can identify them using metrics of type two and finally focus more precisely on what needs to be done in order to fix them.

6 CONCLUSION

Through the literature survey, the needs of the release engineering were identified. Based on them, a solution related to an appropriate monitoring dashboard tailored to the needs of the data scientists was selected. Motivated by this, an advanced monitoring dashboard was developed for this project, together with the application of release engineering techniques to our system. The dashboard enables the data scientists to detect the weakness of the model quickly, thus, allowing them to update it swiftly. Thus, it serves the main purpose of each software system, namely to provide the end-user with a better experience of the product. Besides, the advanced UI lets the data scientists try out the system before releasing the product, and assists in collecting the real-life data. Moreover, by utilizing the basic requirements guideline, the concept of release engineering was successfully adopted for the future growth of our application.

The team found out that the new improvements contributed to the understanding of the models and also improved the practicality the models.

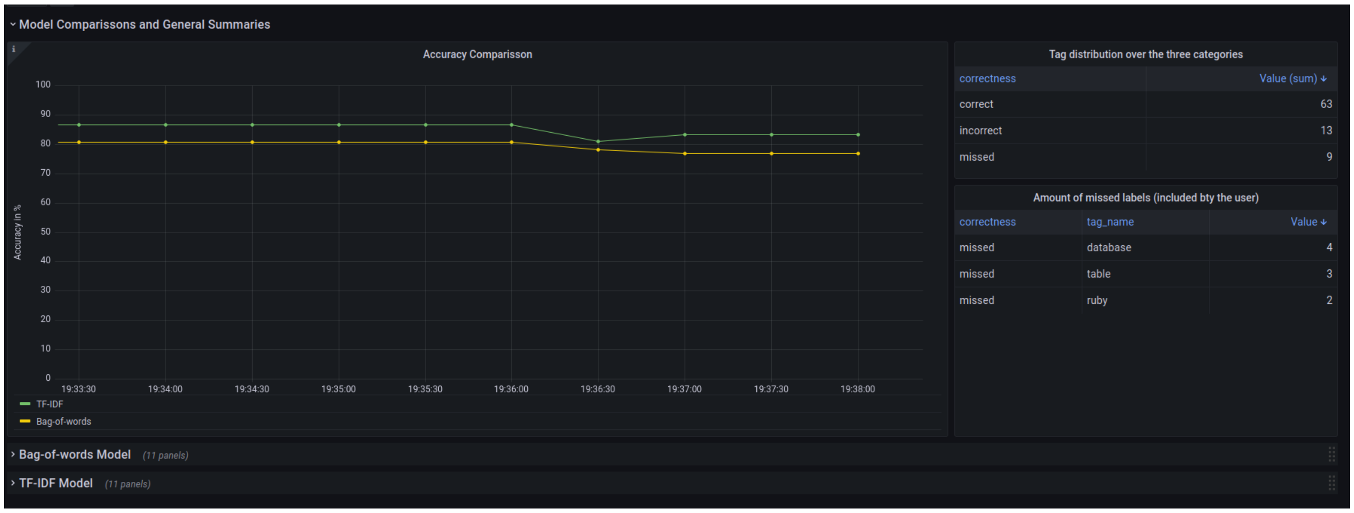
7 FUTURE WORK

The advanced Grafana dashboard was set up to ease the life of the data scientists. As the project spans only one month, the core subset of the features was prioritised.

It is hard for data scientists to monitor the model all the time. Enabling alerting in case of extreme change in metrics allows for productive time management. Apart from that, logging can also be included. It would make the debugging process more straightforward, for example, by allowing the data scientists to look at which titles cause the plummet in the performance.

A different potential improvement is related to the possibility of automatically mining StackOverflow. Our work makes it easy for data scientists to identify the weakness of their models. We think that this can also be done automatically while relying on the metrics. A major improvement would be to then directly mine StackOverflow for relevant data, based on the metrics. This would allow us to close the loop and perform continuous training on the mined data.

While on the note on continuous training, we have identified another potential data source for it. That could be the data generated through the web interface. However, it needs to be used cautiously as it is user-generated and will inherently include errors that need to be mitigated beforehand.



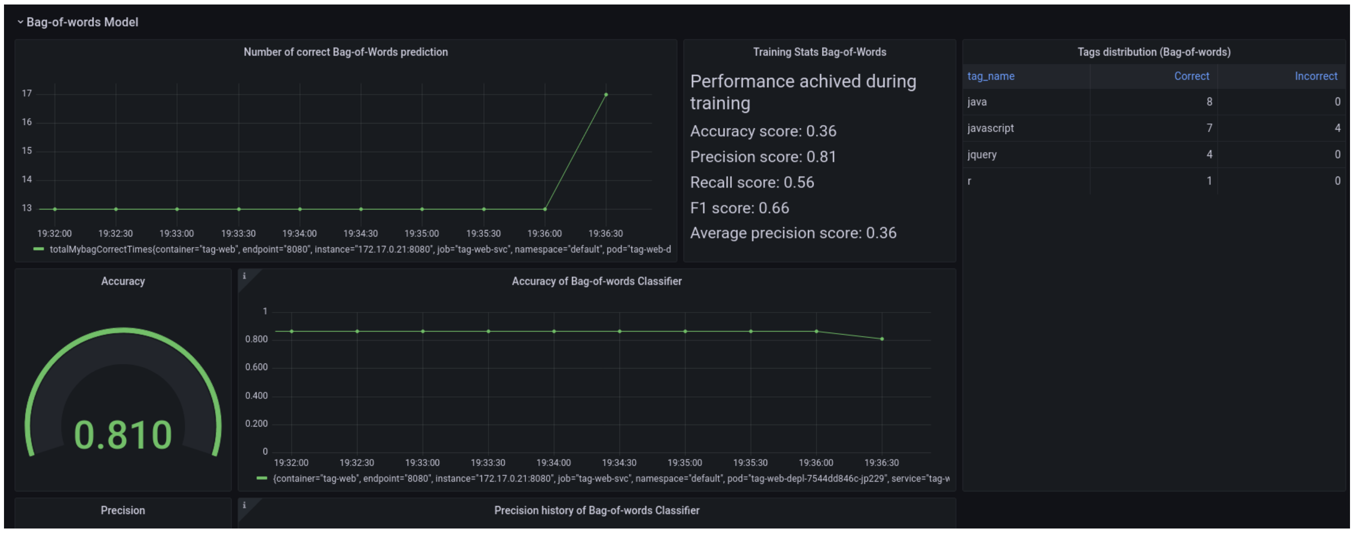
Tag distribution over the three categories

correctness	Value (sum)
correct	63
incorrect	13
missed	9

Amount of missed labels (included bty the user)

correctness	tag_name	Value
missed	database	4
missed	table	3
missed	ruby	2

Figure 5: Dashboard view: Comparison row


Bag-of-words Model
Number of correct Bag-of-Words prediction
17
16
15
14
13
19:32:00
19:32:30
19:33:00
19:33:30
19:34:00
19:34:30
19:35:00
19:35:30
19:36:00
19:36:30
totalMybagCorrectTimes{container="tag-web", endpoint="8080", instance="172.17.0.21:8080", job="tag-web-svc", namespace="default", pod="tag-web-d
Accuracy
0.810
Accuracy of Bag-of-words Classifier
1
0.800
0.600
0.400
0.200
0
19:32:00
19:32:30
19:33:00
19:33:30
19:34:00
19:34:30
19:35:00
19:35:30
19:36:00
19:36:30
{container="tag-web", endpoint="8080", instance="172.17.0.21:8080", job="tag-web-svc", namespace="default", pod="tag-web-depl-7544dd846c-jp229", service="tag-w
Precision
Precision history of Bag-of-words Classifier

Training Stats Bag-of-Words

Performance achived during training

Accuracy score: 0.36

Precision score: 0.81

Recall score: 0.56

F1 score: 0.66

Average precision score: 0.36

Tags distribution (Bag-of-words)

tag_name	Correct	Incorrect
java	8	0
javascript	7	4
jquery	4	0
r	1	0

Figure 6: Dashboard view: Model-specific rows

The last suggested improvement is related to adding more models. The currently used models are all designed for classification tasks. If different types of models were to be added, additional metrics can be used to assess the model. For example, for the regression model, Root Mean Squared Error or Mean Squared Error can be used as metrics. For computer vision tasks, Peak signal-to-noise ratio, Structural Similarity Index, and Intersection Over Union can be used as well. This dashboard can be extended to support all these different types of models and metrics.

REFERENCES

[1] Bram Adams, Stephany Bellomo, Christian Bird, Boris Debić, Foutse Khomh, Kim Moir, and John O'Duinn. 2014 Oct. Testing Scientific Software: A Systematic Literature Review. *PubMed-not-MEDLINE* (2014 Oct). https://doi.org/10.1016/j.infsof.2014.05.006.

[2] Bram Adams, Stephany Bellomo, Christian Bird, Boris Debić, Foutse Khomh, Kim Moir, and John O'Duinn. 2018. Release Engineering 3.0. *IEEE Software* 35, 2 (2018), 22–25. https://doi.org/10.1109/MS.2018.1661327

[3] Bram Adams and Shane McIntosh. 2016. Modern Release Engineering in a Nutshell – Why Researchers Should Care. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 5. 78–90. https://doi.org/10.1109/SANER.2016.108

[4] Bird Marshall-Keim Khomh Moir Adams, Bellomo. March 2015. The Practice and Future of Release Engineering. 32, 46 (March 2015). https://doi.org/doi:10.1109/ms.2015.52

[5] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software Engineering for Machine Learning: A Case Study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 291–300. https://doi.org/10.1109/ICSE-SEIP.2019.00042

[6] Paixão M. Araújo, A.A. 2014. Machine Learning for User Modeling in an Interactive Genetic Algorithm for the Next Release Problem. (2014). https://doi.org/10.1007/978-3-319-09940-8_17

[7] Paixao M.-Yeltsin I. et al. Araújo, A.A. 2017. An Architecture based on interactive optimization and machine learning applied to the next release problem. (2017).

https://doi.org/10.1007/s10515-016-0200-3

[8] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D. Sculley. 2016. What's your ML test score? A rubric for ML production systems.

[9] Ling Wang Chin-Chia Wu Win-Chin Lin Danladi H. Abdulkadir Danyu Bai, Hanyu Xue. 2020. Effective algorithms for single-machine learning-effect scheduling to minimize completion-time-based criteria with release dates. 156 (2020). https://doi.org/10.1016/j.eswa.2020.113445

[10] Foutse Khomh, Bram Adams, Jinghui Cheng, Marios Fokaefs, and Giuliano Antoniol. 2018. Software Engineering for Machine-Learning Applications: The Road Ahead. *IEEE Software* 35, 5 (2018), 81–84. https://doi.org/10.1109/MS.2018.3571224

[11] Christian Murphy, Gail E. Kaiser, and Marta Arias. 2007. An Approach to Software Testing of Machine Learning Applications. In *SEKE*.

[12] M. Openja. 2021. An Empirical Study of Testing and Release Practices for Machine Learning Software Systems (Masters thesis, Polytechnique Montréal). (2021). Retrieved from https://publications.polymtl.ca/9177/.

[13] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. 2019. Machine Learning Testing: Survey, Landscapes and Horizons. https://doi.org/10.48550/ARXIV.1906.10742