Automatic log analysis with NLP for the CMS workflow handling

Lukas Layer[1,*], *Daniel Robert* Abercrombie[2,], *Hamed* Bakhshiansohi[3,], *Jennifer* Adelman-McCarthy[4,], *Sharad* Agarwal[5,], *Andres Vargas* Hernandez[6,], *Weinan* Si[7,], and *Jean-Roch* Vlimant[8,]

[1]INFN, National Institute of Nuclear Physics, Naples, Italy
[2]MIT, Massachusetts Institute of Technology, Cambridge, USA
[3]DESY, Deutsches Elektronen-Synchrotron, Hamburg, Germany
[4]FNAL, Fermi National Accelerator Laboratory, Batavia, USA
[5]CERN, European Organization for Nuclear Research, Geneva, Switzerland
[6]Catholic University of America, Washington DC, USA
[7]University of California, Riverside, USA
[8]California Institute of Technology, Pasadena, USA

Abstract. The central Monte-Carlo production of the CMS experiment utilizes the WLCG infrastructure and manages daily thousands of tasks, each up to thousands of jobs. The distributed computing system is bound to sustain a certain rate of failures of various types, which are currently handled by computing operators a posteriori. Within the context of computing operations, and operation intelligence, we propose a Machine Learning technique to learn from the operators with a view to reduce the operational workload and delays. This work is in continuation of CMS work on operation intelligence to try and reach accurate predictions with Machine Learning. We present an approach to consider the log files of the workflows as regular text to leverage modern techniques from Natural Language Processing (NLP). In general, log files contain a substantial amount of text that is not human language. Therefore, different log parsing approaches are studied in order to map the log files' words to high dimensional vectors. These vectors are then exploited as feature space to train a model that predicts the action that the operator has to take. This approach has the advantage that the information of the log files is extracted automatically and the format of the logs can be arbitrary. In this work the performance of the log file analysis with NLP is presented and compared to previous approaches.

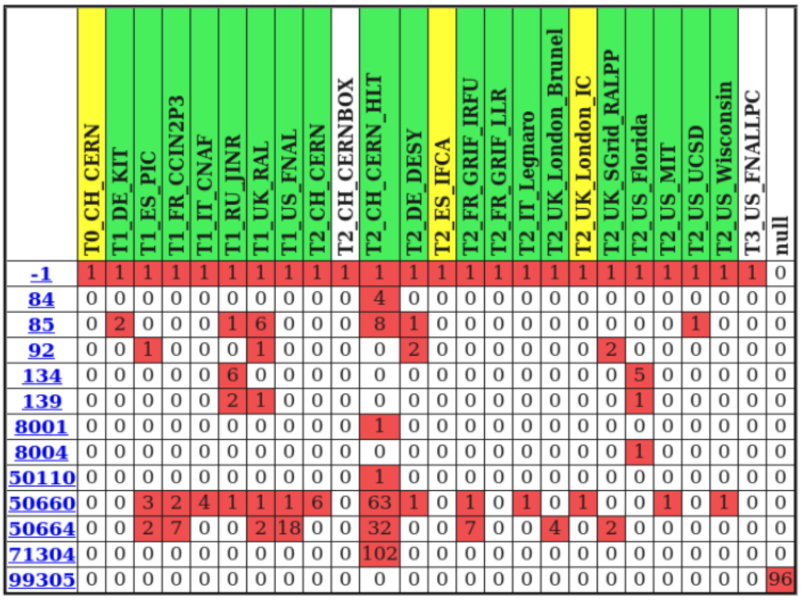
1 Introduction

The central Monte-Carlo production of the CMS experiment [1] utilizes the LHC grid and manages thousands of workflow tasks, each with thousands of jobs on over a hundred computing centers worldwide. A certain rate of the jobs fail due to various issues, such as missing input files or high memory usage. Despite effort of reducing the rate of failure, there remains a fraction of workflows that requires non trivial intervention. This work has to be done by computing operators that look at the error codes and error log files to decide the appropriate

[*]e-mail: lukas.layer@cern.ch

actions to take on the workflows. Machine Learning is a natural solution to move the error handling from manual operation to automated operation. In recent years a framework has been developed that stores the decision and the information available to the operator for taking that decision. It is therefore possible to employ Machine Learning techniques to learn from the operator by training on the labeled data. Several models have been trained based on the error codes of the workflow failures, but the predictions of these models are not yet sufficiently precise to replace the operator. Thus new approaches are explored to add the information that is available in the error log files of the failed workflows in order to improve the precision.

2 Dataset and Strategy



	T0_CH_CERN	T1_DE_KIT	T1_ES_PIC	T1_FR_CCIN2P3	T1_IT_CNAF	T1_RU_JINR	T1_UK_RAL	T1_US_FNAL	T2_CH_CERN	T2_CH_CERNBOX	T2_CH_CERN_HLT	T2_DE_DESY	T2_ES_IFCA	T2_FR_GRIF_IRFU	T2_FR_GRIF_LLR	T2_IT_Legnaro	T2_UK_London_Brunel	T2_UK_London_IC	T2_UK_SGrid_RALPP	T2_US_Florida	T2_US_MIT	T2_US_UCSD	T2_US_Wisconsin	T3_US_FNALLPC	null
-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
84	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
85	0	2	0	0	0	1	6	0	0	0	8	1	0	0	0	0	0	0	0	0	0	1	0	0	0
92	0	0	1	0	0	0	1	0	0	0	0	2	0	0	0	0	0	0	2	0	0	0	0	0	0
134	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0
139	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
8001	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8004	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
50110	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
50660	0	0	3	2	4	1	1	1	6	0	63	1	0	1	0	1	0	1	0	0	1	0	1	0	0
50664	0	0	2	7	0	0	2	18	0	0	32	0	0	7	0	0	4	0	2	0	0	0	0	0	0
71304	0	0	0	0	0	0	0	0	0	0	102	0	0	0	0	0	0	0	0	0	0	0	0	0	0
99305	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96

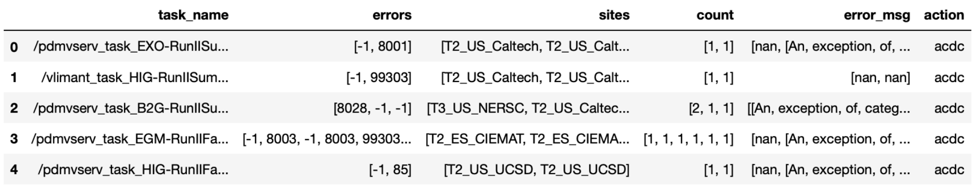
Figure 1: Example for a sparse error-site matrix. The entries of the matrix are the number of times an error code has been thrown.

The dataset used for the Machine Learning consists of failing workflow tasks in the time period between January 2017 and August 2019, which corresponds to approximately 33 000 failed tasks. The data is pulled from the CMS services using *Workflow Team Web Tools*. The target for the Machine Learning are the actions of the operators. In Table 1 the most important actions are shown. The most common action "ACDC without modification" is a retry of only failed jobs without any modification of the memory or the job splitting. In the scope of this work, the goal of the Machine Learning is a binary classification to predict whether the action taken by the operator is an "ACDC without modification" or a different action.
The input for the Machine Learning is a sparse matrix of error codes and sites. For each task the number of times a possible error code is thrown at each site is known. With this information a sparse matrix of error codes and sites can be built that contains the number of times an error has been thrown. An example is shown in Figure 1. Several models to predict the operator's action based on this input have been studied in the last years [2].

Table 1: Actions taken by the operator

Action	Fraction
ACDC without modification	87 %
ACDC with modification	6%
clone	5 %
other	2 %

Additionally, for each thrown error code a snippet of the error log that contains the occurred exception is stored by the CMS *WMArchive* service [3]. The *WMArchive* entries are analyzed with *Apache Spark* on the CERN SWAN platform for interactive computing [5]. The *pandas* library [4] is used to merge the information of the log snippets from *WMArchive* and the information from *Workflow Team Web Tools*. A *pandas* frame that contains the raw input for the Machine Learning is shown in Figure 2.



	task_name	errors	sites	count	error_msg	action
0	/pdmvserv_task_EXO-RunIISu...	[-1, 8001]	[T2_US_Caltech, T2_US_Calt...	[1, 1]	[nan, [An, exception, of, ...	acdc
1	/vlimant_task_HIG-RunIISum...	[-1, 99303]	[T2_US_Caltech, T2_US_Calt...	[1, 1]	[nan, nan]	acdc
2	/pdmvserv_task_B2G-RunIISu...	[8028, -1, -1]	[T3_US_NERSC, T2_US_Caltec...	[2, 1, 1]	[[An, exception, of, categ...	acdc
3	/pdmvserv_task_EGM-RunIIFa...	[-1, 8003, -1, 8003, 99303...	[T2_ES_CIEMAT, T2_ES_CIEMA...	[1, 1, 1, 1, 1, 1]	[nan, [An, exception, of, ...	acdc
4	/pdmvserv_task_HIG-RunIIFa...	[-1, 85]	[T2_US_UCSD, T2_US_UCSD]	[1, 1]	[nan, [An, exception, of, ...	acdc

Figure 2: *Pandas* frame that contains the information to build the sparse error-site matrix and the labels for the Machine Learning.

3 Natural Language Processing

To use the error log snippets in the Machine Learning, numerical representations of the words (word embeddings) have to be obtained and subsequently have to be converted into a numerical representation of the text snippet. This can either be done using unsupervised algorithms, like *word2vec*, or by learning the word representations directly in the supervised training. In both cases the error log snippets have to be preprocessed.

3.1 Preprocessing of the error log snippets

The preprocessing consists out of the following steps and is done with *pandas* and the *Natural Language Toolkit* [7].

- Tokenization: the text string is split up in single words with the *NLTK* treebank tokenizer, which is a tokenizer that uses regular expressions and is suited for error logs.
- Cleaning: low frequency words, special characters and single characters that are not important for the meaning of the error message are filtered out.
- Selection: since sometimes multiple snippets of the error logs have been stored, ad-hoc rules have been introduced to select the most meaningful snippet.
- Indexing: every unique word in the corpus of the error log snippets is associated with an unique integer.

3.2 Unsupervised learning of word embeddings

Word2vec [6] is an unsupervised algorithm to learn word embeddings. The model is a shallow, two-layer Neural Network that takes a text corpus as input. The output is a high-dimensional vector space with each unique word in the corpus being assigned a corresponding vector in the space. Semantically similar words are mapped to nearby points in the vector space. This captures relations between the words and reduces the dimensionality compared to one-hot encoding.
The *word2vec* algorithm is used to learn word embeddings for the words of the error log snippets. Subsequently the word vectors of the words in each error log snippet are averaged to get a numerical representation of each snippet. The resulting high-dimensional vectors can be visualized in two dimensions with the t-SNE algorithm, that maps similar objects to nearby points and dissimilar objects to distant points. In Figure 3 the output of the t-SNE algorithm for the averaged vectors of 5000 error log snippets is shown. Clusters are forming that correspond to similar error codes. This is natural, since error log snippets from the same error code share several words and thus the averaged vectors are close in the high dimensional vector space.

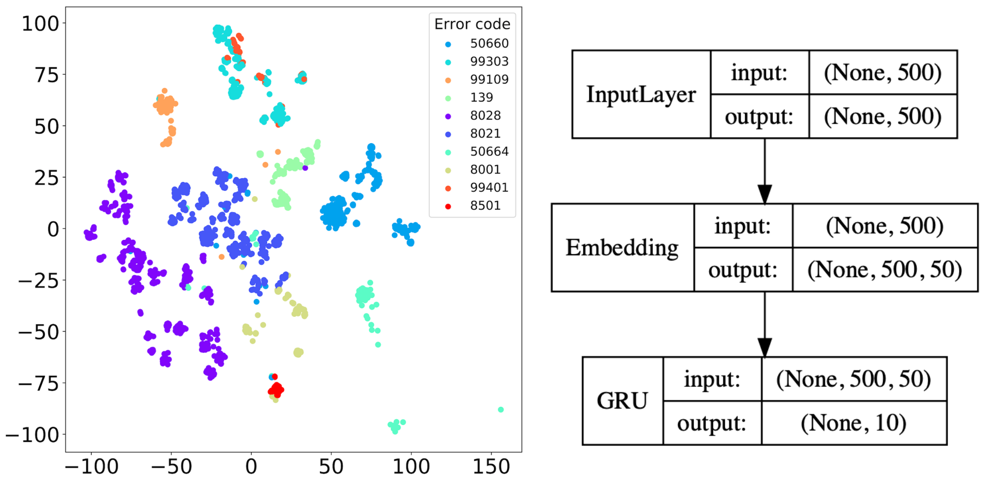


Figure 3: Left panel: output of the t-SNE algorithm for 5000 error log snippets after averaging the word vectors of each snippet. Right panel: sketch of a model that uses a GRU to obtain a numerical representation of a text snippet.

3.3 Supervised learning of word embeddings

The unsupervised training does not make use of the sequential nature of language. This can be exploited by using a Recurrent Neural Network, such as a Long Short-Term Memory (LSTM) or a Gated Recurrent Unit (GRU). In the right panel of Figure 3 an example for a model using a GRU is shown. The input layer of the model takes a vector with a maximum of 500 indexed words. The following embedding layer picks out the word vector corresponding to the respective integer of each word. In case that the text snippet is shorter than 500 words, the integer sequence is filled up with zeros. The zeros are masked in the embedding layer, such that the vectors corresponding to zero are not considered in further calculations. It should be noted that the embedding layer can be initialized with vectors obtained in unsupervised trainings, e.g. by training on the whole corpus of Wikipedia. The GRU takes as input the sequence of word vectors and outputs a 10-dimensional vector that encodes the meaning of the text snippet. This model can then be used in a larger model to train the word embeddings depending on the objective to optimize.

4 Models and Training

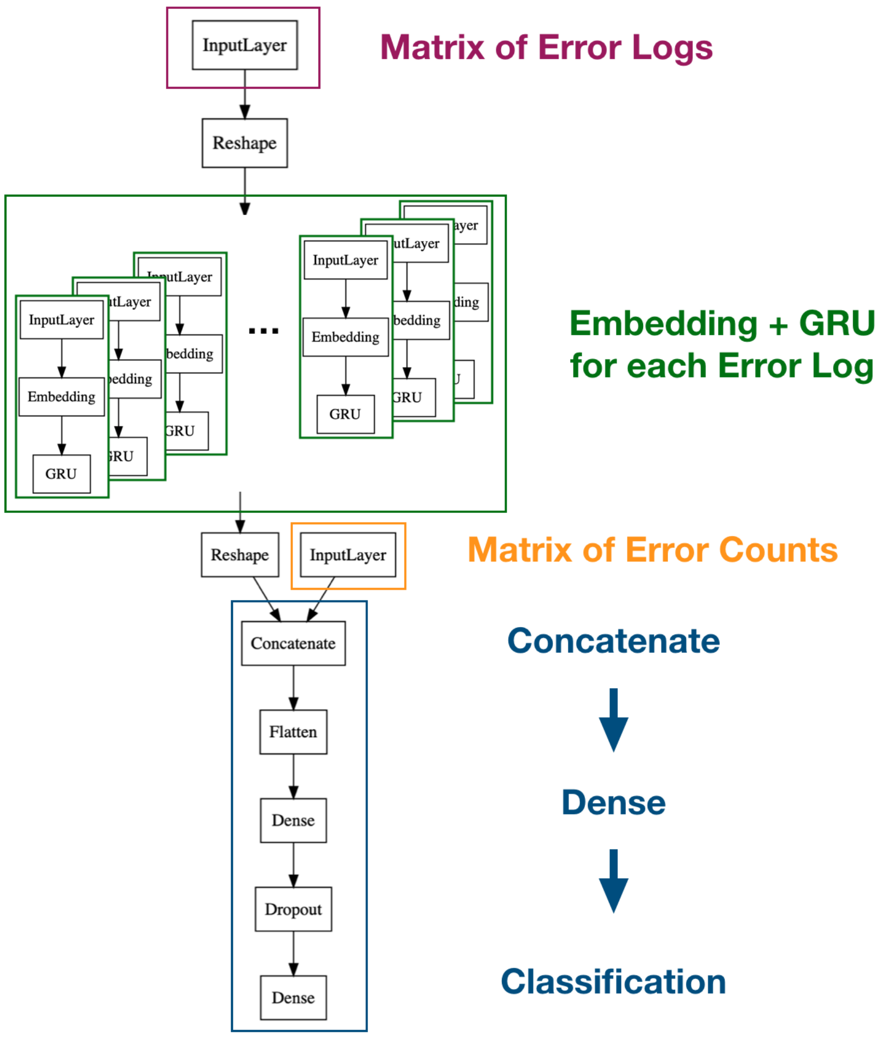


Figure 4: Sketch of a model that uses a GRU to encode the error log snippets of the sparse error-site matrix. Also the error count information is added. The final target is a binary classification.

The models are implemented using the *Keras* framework [8]. Three different classes of models are considered:

- Baseline model: a simple Feed-Forward Neural Network that only uses the error-site matrix with the number of times an error code has been thrown on each site.
- Model that uses an RNN for the error log snippets (RNN): a model that uses the error-site matrix with the number of times an error code has been thrown as first input branch and the RNN model described in Section 3.3 to represent the error log snippets as second input branch. The word embeddings are learned in the supervised training. A sketch for the architecture of this model is shown in Figure 4.
- Model that uses averaged *word2vec* vectors (AVG): a model that uses the error-site matrix with the number of times an error code has been thrown as first input branch and the averaged *word2vec* vectors of the error log snippets as second input branch.

Table 2: Model parameters

Model	Number of parameters	Batch size	Training time (1 epoch)
Baseline	$O(500\,000)$	500	$O(1\ ms)$
AVG	$O(1\,000\,000)$	100	$O(1\ s)$
RNN	$O(5\,000\,000)$	4	$O(1\ h)$

The order of magnitude of the number of parameters, the batch size and the training time for one epoch is shown in Table 2. In particular the use of RNNs results in long training times and a small batch size has to be used due to the high memory usage of the model. The training has been done on the Caltech GPU Cluster with 2-8 NVidia GeForce GTX and Titan X/Xp. Bayesian optimization with multiple GPUs has been used to optimize the hyperparameters of the models. Two further methods for the training have been explored: the NNLO framework [9] that allows to train one model distributed on multiple GPUs and the spark_sklearn library on the CERN SWAN platform that allows to train up to 60 3-fold cross-validated models in parallel.

5 Results

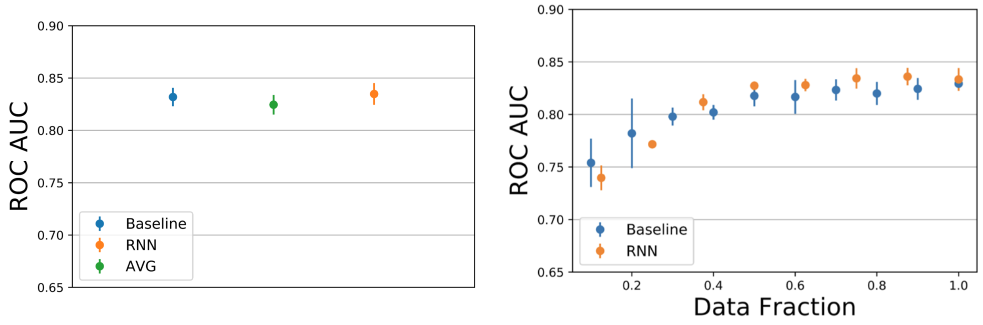


Figure 5: Left panel: 3-folded ROC AUC for the discrimination of "ACDC withouth modification" vs. other actions for the three models. Right panel: ROC AUC as a function of the training data for the baseline model and the RNN model.

Due to the small amount of training data and the large class-imbalance, 3-fold cross-validation of the area under the ROC curve (ROC AUC) is used as metric for the evaluation of the performance. As described in Section 2, the target is to discriminate the most common action "ACDC without modification" against other actions. In the left panel in Figure 5 the optimized ROC AUC for the three models defined in the previous section is shown. The different models give similar results and the ROC AUC value is around 0.83. In the right panel the ROC AUC metric is shown as a function of the training data for the baseline model and the most promising NLP model that uses an RNN to represent the error log snippets. This plot illustrates that the performance of the models is still improving with more data.

6 Conclusions

A pipeline for the Data Acquisition and Machine Learning has been implemented to predict the operator's action for failing CMS workflow tasks. First models using Natural Language Processing have been trained successfully. The addition of the error log snippets in the training makes the models more complex and in particular the use of RNNs results in long training times. At this point of the work, the models have a similar performance.
Further optimization of the pipeline is required to exploit the full potential of the error logs. In general, the performance of the models will improve over the next years with more data. In the near future the models will be deployed and feedback from the operators on the suggestions of the AI will be collected. A more refined training will be performed to move from binary classification to multiclass classification in order to predict multiple actions.
Within the *Rucio Scientific Data Management* program [10] the development of a common system to collect and categorize errors, provide operators with actions and collect feedback on the AI suggestions is foreseen and the acquisition and analysis of error log snippets with Machine Learning, as explored in this work, will be part of it.

References

[1] CMS Collaboration, *"The CMS experiment at the CERN LHC"*, J. Instrum **3** (2008)
[2] C. Contreras et al., *"CMS Workflow Failures Recovery Panel, Towards AI-assisted Operation"*, CHEP 2018, https://indico.cern.ch/event/587955/contributions/2937424/
[3] V. Kuznetsov and N. Fischer and Y. Guo, *"The Archive Solution for Distributed Workflow Management Agents of the CMS Experiment at LHC"*, Comput. Softw. Big Sci **2** (2018)
[4] W. McKinney, *"Data Structures for Statistical Computing in Python"*, Proceedings of the 9th Python in Science Conference, 51-56 (2010)
[5] D. Piparo et al., *"SWAN: a service for interactive analysis in the cloud"*, Future Gener Comput Syst 78 Part 3 (2018)
[6] T. Mikolov et al., *"Efficient Estimation of Word Representations in Vector Space"*, arXiv:1301.3781
[7] E. Loper and S., Bir, *"NLTK: The Natural Language Toolkit"*, In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics, 2002
[8] F. Chollet and others, *"Keras"*, 2015, https://keras.io
[9] J. Vlimant et al., *"MPI-based tools for large-scale training and optimization at HPC sites"*, CHEP 2019, https://indico.cern.ch/event/773049/contributions/3474799/
[10] M. Barisits, T. Beermann, F. Berghaus et al., *"Rucio: Scientific Data Management"*, Comput Softw Big Sci (2019) **3**: 11