A Comparative Performance Analysis of Supervised, Unsupervised, and Reinforcement Learning Algorithms in Containerized Cloud Environments Using Docker and Kubernetes

Andreea - Oana RADU, Sergiu - Alexandru IONESCU, Ioana NAGÎȚ
Bucharest University of Economic Studies, Romania
raduandreea22@stud.ase.ro, ionescusergiu14@stud.ase.ro, nagitioana15@stud.ase.ro

Nowadays, the selection of the implementation infrastructure is an important decision from an operational point of view for any organization that uses machine learning in production, this directly influencing the cost, response time and system scalability. With the support of a public dataset which contains both fake and true news, this paper proposes a benchmarking flow, reproducible and auditable, structured as a comparison between ten machine learning algorithms from the three types of learning: supervised learning, unsupervised learning and reinforcement learning. These algorithms were identically deployed in containerized infrastructure and were selected to represent three levels of complexity interpretation: classic linear models, non-linear models and neural networks, and for reinforcement learning, it was used the idea of tabular control versus approximation through neural network. All algorithms were implemented in Python using scikit-learn and TensorFlow for data processing and for their containerization and orchestration, Docker and Kubernetes were used. The ten algorithms were experimentally compared in five different seeds on a local environment, with the help of the Docker containers and a Kubernetes cluster, managed in cloud with Azure Kubernetes Services. The results show that the main factors influencing the calculation cost of an algorithm are represented by its internal mechanism (simple table versus complex neural network) and the execution environment (local, Docker or Kubernetes), these proving to be more relevant than the learning type to which the algorithm belongs. A comparison of execution environments highlights a significant increase in execution time per run, from the local environment to the cloud cluster. This difference reflects a combination of factors such as different hardware, the simultaneous execution of multiple tasks on AKS, and orchestration costs; these factors cannot be completely separated within the current experimental design and are presented and discussed in this paper.

Keywords: *machine learning benchmarking, Docker, Kubernetes, cloud computing, reinforcement learning*

1 Introduction

The container-based solutions, especially Docker, and in combination with the Kubernetes orchestration platforms, have become a standard without which the machine learning tasks cannot be implemented in the cloud environments. Containers offer reproducibility, portability between heterogeneous infrastructures and simplified management of dependencies. These are very important and valuable in the data science research, where experiments must be resumed on hardware, operating systems and different cloud suppliers [1], [2].

In the specialty literature, there is research about the containerization of certain individual work flows, with the help of machine learning or about Kubernetes optimization with the help of the machine learning techniques, especially reinforcement learning, applied for the planning of the cluster resources [3], but there are few empirical studies which directly compared the manner in which various algorithms from the three types of learning: supervised, unsupervised

and reinforcement learning, behave when they are identically implemented in a containerized infrastructure. Because each learning type presents different computing features: supervised learning uses classification based on selected labels, unsupervised learning groups data in clusters, based on similarity or density calculations, and the reinforcement learning involves sequential interaction, with state, with an environment, often combined with approximation with the help of neural networks in more advanced variants, as Deep Q-Networks (DQN), the agent learning from rewards [4]. Therefore, these differences suggest that the infrastructure overhead, the resource consumption and the scaling behavior, may substantially vary not only between individual algorithms, but also between the learning types. In this paper, it is presented a comparative performance analysis for ten machine learning algorithms, distributed as follows: four supervised learning algorithms Support Vector Machine (SVM), Random Forest, Logistic Regression and a convolutional neural network inspired from the architecture of the algorithm DeepText, used by Facebook, two unsupervised learning algorithms, K-Means and DBSCAN, and, last but not least, there were assessed four variants of reinforcement learning SARSA and DQN, each being assessed both native as well as reformulated as "contextual bandit", with a single step, to provide compatibility directly on a common classification task. These algorithms were run in identical Docker containers and orchestrated with the help of Kubernetes, initially on a local cluster, Minikube, and subsequently on a cloud cluster managed by Azure Kubernetes Service.

In this paper, three important aspects were considered, which can represent a starting point for future research in the sense of containerization with Docker, Kubernetes and cloud. Therefore, it was established an empirical benchmark with different seeds for the ten algorithms assessed from the point of view of accuracy, training time, CPU use and memory consumption. These were done with the help of a container stratified architecture and their orchestration including secured Docker images and Kubernetes Indexed Jobs for the parallel execution of the five seeds, in order to be a reproducible benchmark. Last but not least, this paper presents a quantified analysis of differences in execution times across various infrastructure layers, ranging from the local virtual environment to the containerized and orchestrated cloud environment, explicitly showing that the effects of hardware, concurrency, and orchestration could not be completely separated within the scope of this study (see Section 7.1), and provides practical guidance for resource allocation when containerizing heterogeneous machine learning tasks. Overall, this paper contributes to the decision-making process regarding both the infrastructure a company can use to run algorithms from different types of machine learning and the choice of a machine learning method based on expected performance.

2 Literature review

In the current research regarding containerization for machine learning, it can be noticed a focus on treating Docker containers as a solution for the experimental reproducibility issue. In the paper [1], the authors largely discuss the reproducibility idea for machine learning and propose, as a solution, the containerization as a mechanism capable of encapsulating in a single package the code, software dependencies and the execution environment of an experiment, proving these through a dedicated platform of replication of the machine learning models, on a large scale. Although it is a paper about containerization, it does not empirically compare the resource behavior of certain learn-

ing algorithms from various learning types inside the containerized infrastructure. The authors of [5] propose a scalable benchmarking platform for deep learning models, but this, too, does not extend the comparison to different types of learning.

A second research direction encountered in the papers published on this topic, is the one of applying reinforcement learning techniques for the optimization of Kubernetes itself. In paper [3], the authors propose a planning algorithm of the pods for Kubernetes clusters used in edge computing environments, based on proximal policy optimization, therefore proving a significant reduction of the average response time, compared to the implicit scheduler of the Kubernetes orchestrator. A similar approach is also proposed in [6], where the authors propose a customized Kubernetes scheduler, based on a neural network trained to predict the planning times. Thus, in these papers, reinforcement learning is the infrastructure optimization means, and it is not used as a comparison component [7], [8]. Therefore, this paper brings novelty by comparing the algorithms, including the ones of reinforcement learning type, with the algorithms of the other learning types.

With respect to the comparison of the algorithms assessed in this paper, the comparison of the supervised learning methods to detect fake news, was the subject of several independent studies. In paper [9], we have compared four supervised learning algorithms on the same dataset used also in this paper for fake news detection, but this paper represents a continuation of the research by comparing several learning types and their containerization in the local virtual environment, as well as in the cloud environment; thus, this study does not reuse any results, text, or figures from [9].

In paper [10], the authors compare the performance of algorithms SVM, Naïve Bayes, Random Forest and Logistic Regression on several datasets for the detection of disinformation, obtaining an accuracy of up to 97% with SVM on one of the tested datasets. Also, the authors of paper [11], presented the Deep Q-Network (DQN) algorithm, the reinforcement learning algorithm used in this paper within the comparisons performed. They proved for the first time the capacity of a neural network to reach performances comparable to the human ones, on a wide range of sequential control tasks.

There is research on the machine learning algorithm benchmarking, both regarding the comparison made between the classification methods algorithms as well as regarding the infrastructure optimization through reinforcement learning, but this research tends to treat the topics separately. Therefore, this paper extends the analysis of the classification methods comparison to the three learning types and, also, containerization is used as measurement object within the research, and not as an optimization target. A related conceptual direction is proposed in paper [12], which introduces a Docker and Kubernetes containerization theoretical architecture for the integration of the machine learning algorithms in educational platforms, covering all three learning types, but without experimental validation of the resources' behavior. This paper completes this theoretical framework through an empirical study, with concrete performance measurements and infrastructure overhead; [12] contains no experimental results, so there is no overlap between the two papers.

3 Methodology

For this paper, the observation methodology was used to select the dataset for algorithm testing and their containerization. It was also used the experimental methodology to test algorithms and to create containerized infrastructure in local environment, as well

as in cloud. Therefore, the objective of the research is to empirically measure the behavior of the machine learning algorithms from different paradigms: supervised learning, unsupervised learning and reinforcement learning, when they are identically run in containerized infrastructure with Docker and Kubernetes or in a cloud environment. And, in the end, to be able to see what influences the most their infrastructure cost: time, CPU or memory.

The dataset is a public one, taken from the Kaggle platform, with the title „Fake and Real News Dataset". It contains 44,898 press articles, binary labeled, with 1 the true ones and with 0 the fake ones. Each article contains a title, the text of the news, subject and publishing date. The text processing consisted of the preprocessing stage, through the conversion of the letters from uppercase to lowercase, the removal of URLs and punctuation marks, and, subsequently, the representation of the vectorial text with TF-IDF, for classical algorithms. For DeepText, trainable tokenization and embedding were used.

For each algorithm, the datasets for training and testing were obtained by dividing the existing dataset as follows: 80% for training and 20% for testing. All these data were divided using five preselected, fixed seed values, and this determined the initialization of the model's parameters, the sampling order and the initialization of the simulation environment for the reinforcement learning algorithms. The values of the seeds were identical for all trained and tested algorithms, the values of these seeds being: 42, 123, 2024, 7 and 99. Also, it can be noticed that a single seed controls, at the same time, several things: the way the data is split into training and testing, the way the model's parameters are initialized, the order in which the data is used for training and, for the reinforcement learning algorithms, the way the simulation environment starts. This means that the standard deviation reported across the five seeds reflects, at the same time, all these sources of variation, and not each one separately, and this was not an aspect that the present research aimed to separate. They helped to obtain an estimated average and a standard deviation for all metrics considered within the algorithm testing process. In order to compare algorithms, we used metrics as: accuracy, precision, F1 score and AUC (Area Under the Curve). There were also used the standard metrics, used for the assessment of the clustering algorithms, such as: Silhouette, which measures how good the found clusters are formed, and ARI (Adjusted Rand Index) which measures the extent of the overlapping of the clusters found by the algorithm with the known real labels. It uses real labels, but only for testing, not for training. And, last but not least, execution time, CPU usage, and memory usage are used as comparison metrics, measured using a custom monitor built on the Python psutil library, which samples the CPU usage percentage and the RSS (Resident Set Size) of the training process every 0.5 seconds in a background thread, reporting the average and peak values used throughout this study. In Table 1, the hyperparameters used for the ten algorithms evaluated within this research can be noticed, and, also, the sampling instrument and the interval used for the CPU and memory measurements, described in section 3, are presented.

Table 1. Algorithm configuration and hyperparameters

Algorithm	**Key Hyperparameters**
Text vectorization (TF-IDF)	unigrams + bigrams, max_features = 20,000
SVM	LinearSVC, C = 1.0, max_iter = 5000, Platt-

	scaled via 3-fold CV
Random Forest	n_estimators = 300, max_depth = None, n_jobs = -1
Logistic Regression	solver = lbfgs, C = 1.0, max_iter = 2000
DeepText (CNN)	embedding dim = 128; 3 parallel Conv1D branches (kernel sizes 3, 4, 5; 100 filters each); dropout = 0.5; dense = 64; epochs = 3; batch size = 64; Adam optimizer; vocab = 20,000; sequence length = 200
K-Means	n_clusters = 2, n_init = 10; TF-IDF reduced to 100 dimensions via TruncatedSVD
DBSCAN	eps = 0.3, min_samples = 10; TF-IDF reduced to 50 dimensions via TruncatedSVD
SARSA (native)	alpha = 0.1, gamma = 0.99, epsilon 1.0→0.01 (decay 0.9995), 5000 episodes, FrozenLake-v1 (is_slippery = True)
DQN (native)	gamma = 0.99, lr = 1e-3, batch = 64, replay buffer = 20,000, epsilon 1.0→0.01 (decay 0.995), target-network update every 10 episodes, 300 episodes, CartPole-v1
SARSA (bandit)	state = K-Means cluster id (100 clusters) on a 50-dim SVD representation; alpha = 0.2; 3 epochs over the training set
DQN (bandit)	state = 100-dim SVD vector; single-step return (gamma = 0); lr = 1e-3; batch = 64; 2 epochs over the training set
Resource monitoring instrument	Custom monitor based on the Python psutil library, sampling interval = 0.5 seconds

Also, in order to support the reproducibility of the results presented in this paper, the source code, the Dockerfiles and the Kubernetes manifests used within this research are publicly available at [link GitHub].

4 Presentation of algorithms

For algorithm selection, two criteria were considered. First of all, each learning type: supervised, unsupervised and reinforcement learning had to be represented, including both classical methods and methods based on deep learning. The second selection criterion was represented by the diversity of the usage profiles of the relevant resources, to assess the containerized infrastructure behavior under heterogeneous tasks.

4.1 Supervised algorithms

Within the research, four supervised learning algorithms were selected, from which three are classic algorithms used for data classification. SVM (Support Vector Machine) uses a hyperplane that separates the fake articles from the real ones as well as possible. The algorithm identifies the decision boundary that maximizes the margin between the two classes [13]. Random Forest, within the research paper, builds 300 decision trees, each one being trained on a different selection of data and features, and, then, the final decision is the one chosen by the majority of trees [14]. Logistic Regression uses a mathematical function that helps to transform the text into a probability for the news to be fake or real [15]. And the fourth algorithm is DeepText which represents a neural network, which can automatically learn the features from the text. This type of algorithm learns patterns and relations between words and their representation during the training process, and, therefore, it

is not necessary, as in the case of the classical algorithms, to define in an explicit manner certain rules to be able to differentiate the fake news from the real ones.

4.2 Unsupervised algorithms

This paper tested two unsupervised algorithms widely recognized in the academic research as effective for text clustering: K-Means [16], which groups articles in two clusters based on the similarity of their vectorized text, without previously knowing which news is fake and which is true. The second algorithm is called DBSCAN [17], which also groups similar articles. This algorithm lets the number of clusters be determined automatically and can mark certain news as not belonging to any existing cluster.

4.3 Reinforcement learning algorithms

In this research, we used two reinforcement learning algorithms, SARSA and DQN, which are among the best known in the literature. To enable a direct comparison of accuracy, precision, and recall with classification algorithms, we reformulated SARSA [18] and DQN [11] as single-step contextual bandits, applied to the task of classifying news as fake or real, using the same reward and penalty logic: the agent selects a label for each article and receives a reward if it correctly predicts whether the news is fake or real. In the case of SARSA (bandit), the state observed by the agent is the identifier of a K-Means cluster (100 clusters) calculated based on a 50-dimensional SVD-reduced representation of the article, rather than based on the full text representation of the article. This discretization is necessary because SARSA relies on a tabular representation of Q-values, which cannot directly index a continuous, high-dimensional state space. DQN (bandit) uses a neural network instead of a table to approximate the Q-values, which allows it to take as its state the full text representation, reduced via SVD to 100 dimensions, without discretizing it into clusters.

To correctly assess the reinforcement learning algorithms on their canonical tasks as well, allowing a fair comparison of containerization behavior, we also implemented both algorithms in their native environments. SARSA was trained on FrozenLake, where the agent learns, through trial and error, to cross a virtual frozen lake without falling into holes, receiving a reward or penalty at every step. The second algorithm, DQN, was trained on CartPole, using a neural network instead of a simple table, to learn how to keep a pole balanced on a cart, a task with a continuous state space, more complex than the discrete FrozenLake environment on which SARSA was tested.

5 Containerized infrastructure

For each algorithm category, a dedicated Docker image was created, built with the help of python:3.11-slim image and secured by defining a dedicated, non-root user. These Docker images were orchestrated with the help of Kubernetes, using Job type resources, in which data shared through persistent volumes were used. In order for it to be run in parallel, each repetition for each algorithm, it was used the Kubernetes native mechanism, of Indexed Job type, which helped the automatic mapping of the completion index to one of the five pre-selected, fixed seed values. It was also implemented a final job, with the help of an initialization container, which interrogates the status of the other jobs through an API of the Kubernetes orchestrator, and when all other jobs, related to the Docker images of the supervised, unsupervised and reinforcement learning were finalized, it is triggered to start the aggregation and generation stage of the diagrams obtained following the running of the learning algorithms.

These experiments were executed pro-

gressively, in four testing environments: a virtual local environment Python, subsequently we have containerized the algorithms with the help of Docker Desktop on the same hardware and, subsequently, we have orchestrated with Kubernetes, initially on a local cluster with a single Minikube node, 2vCPU with 6 GB RAM allocated, and, subsequently, we ran on a cluster managed by Azure Kubernetes Service (AKS) with two Standard_D4s_v3 type nodes, with a 4 vCPU processor and 16 GB RAM for each node. For everything to operate well in cloud, we selected the West Europe region for resources and we used the student subscription to test the algorithms and containers.

6 Results and discussions

The results of this research are presented in this section with the help of standard deviation and average, for the five independent repetitions obtained with the help of the parallel execution on Azure Kubernetes Service, described in the previous section.

6.1 Performance of the classification algorithms

In Table 2 there are presented the classification metrics of the six algorithms assessed on the dataset for the detection of fake news for the five repetitions with different seeds.

Table 2. Metrics of the classification algorithms

Algorithm	Accuracy (%)	Precision (%)	Recall (%)	F1 score (%)	AUC (%)
SVM	99.63 ± 0.04	99.64 ± 0.08	99.59 ± 0.07	99.62 ± 0.04	99.98 ± 0.01
Random Forest	99.78 ± 0.05	99.77 ± 0.08	99.77 ± 0.14	99.77 ± 0.06	99.99 ± 0.01
Logistic Regression	99.07 ± 0.08	98.87 ± 0.09	99.19 ± 0.13	99.03 ± 0.08	99.91 ± 0.01
DeepText (CNN)	99.86 ± 0.02	99.90 ± 0.02	99.80 ± 0.04	99.85 ± 0.02	100.00 ± 0.00
SARSA (bandit)	86.68 ± 0.32	84.61 ± 1.16	88.14 ± 1.33	86.33 ± 0.30	92.82 ± 0.36
DQN (bandit)	98.71 ± 0.13	98.93 ± 0.23	98.36 ± 0.39	98.64 ± 0.14	99.72 ± 0.04

It can be noticed that DeepText obtained the best accuracy of all algorithms 99.86%, as well as the highest AUC 100.00% ± 0.00%, this algorithm being followed at a very small difference by algorithm Random Forest which obtained 99.78% and the following one was SVM 99.63%. It is notable that the Logistic Regression algorithm, which is the simplest linear model from the ones assessed, remains competitive, with a score of 99.07%, only 0.56 percentage points below SVM. From the two reinforcement learning models of bandit type, DQN algorithm obtained the accuracy of 98.71%, a performance close to the one of Logistic Regression algorithm, while SARSA obtains a pretty low score compared to the rest of the assessed algorithms, 86.68%, with a difference of about 12.4 percentages attributed to the discretization of the state in only 100 clusters K-Means, as described in Section 4.3. As can be noticed, the standard deviation obtained small percentages for all algorithms, below 0.15 percentages for most metrics, with the exception of the SARSA algorithm, which obtained the highest standard deviation for precision and recall compared to the other algorithms, 1.16 and 1.33 percentages, and this aspect indicates stability for the results of the five seeds. It can also be noticed that the accuracies close to the maximum obtained by the assessed algorithms are con-

sistent with a well-known property of this type of dataset, where the true news articles often carry systematic surface elements, such as the source or the formatting used, which make the two classes easier to separate, and, therefore, the results obtained should not be assumed to remain the same on other, less easily separable, fake news datasets. For a more visual representation of the results, these can be seen in Figure 1.

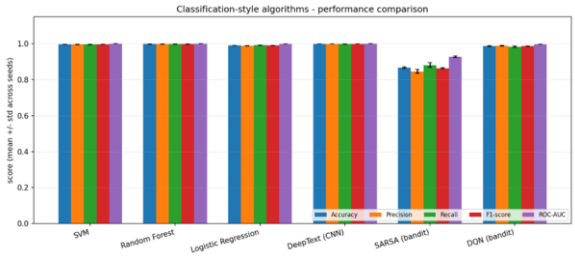

Fig. 1. Classification algorithms metrics

In Figure 2 the ROC curves are presented for the six classification algorithms. These are obtained from a representative run, seed 42, together with the average AUC value on the five repetitions used in the algorithm comparison. It can also be noticed that five algorithms out of the six trained algorithms have almost identical curves, clustered to the upper left corner of the graph, this suggesting the idea that the separation of the fake news from the real ones is almost perfect, a fact confirmed by the AUC values which are close to 1. SARSA algorithm is the only one whose curve visibly moves away from this corner, having a more rounded shape and closer to the dotted diagonal line, which represents the performance of a random classifier. This is also confirmed by the low AUC, 0.928, with a lower accuracy, from Table 2. But, also in this case, it can be noticed that the curve remains above the diagonal, which indicates that the algorithm learned a real structure from the data, but with a lower discriminative performance compared to the other algorithms.

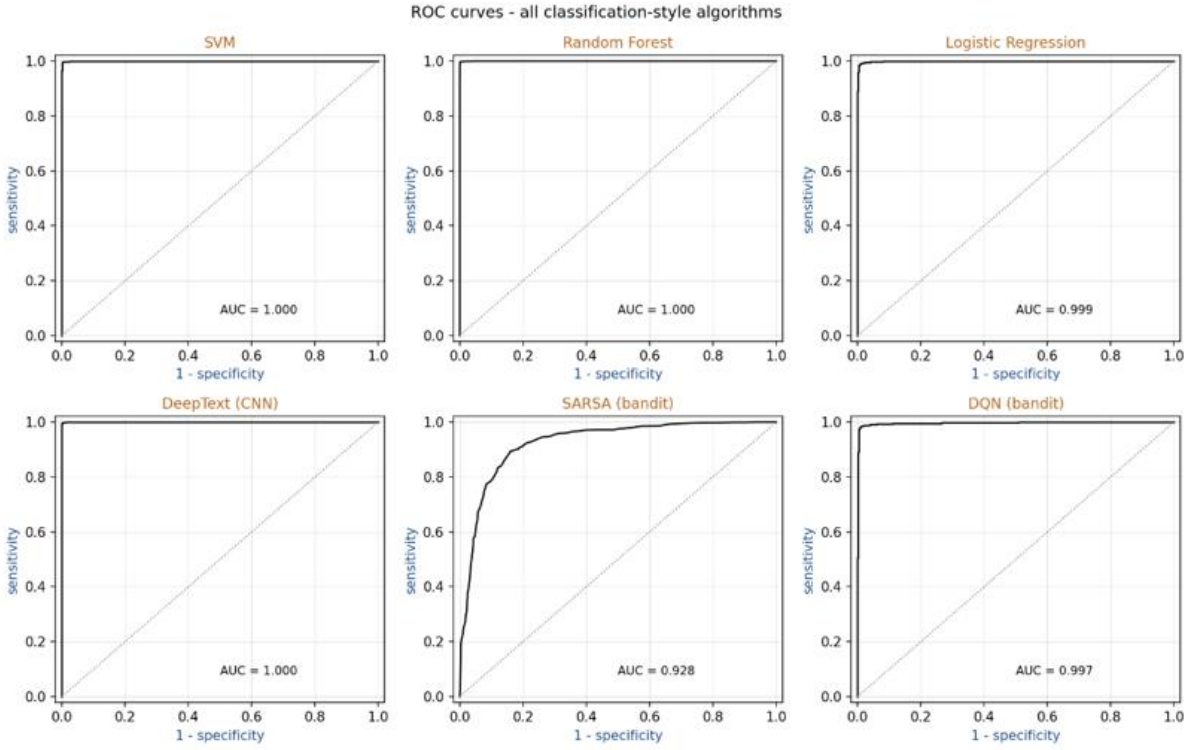

Fig. 2. ROC curves for classification algorithms

6.2 Clustering algorithms performance

Both clustering algorithms, K-Means and DBSCAN tested, produce values of the Rand index, adjusted close to zero. As can be noticed in Table 3, K-Means obtains 0.0237 and DBSCAN obtains -0.0007, with a significant low standard deviation. Therefore, it is confirmed that the unsupervised clustering, applied directly on the lexical representation of the text, does not differentiate between fake and real, even if it was trained on five different seeds. This is also confirmed by the Normalized Mutual Information values, 0.0162 for K-Means and 0.0081 for DBSCAN, both close to zero. Also, the low silhouette score of K-Means, 0.0565, indicates once more that this type of unsupervised learning creates clusters weakly separated or less compact, this being consistent with its inability to differentiate fake news from real ones. In contrast, DBSCAN obtains a considerably higher silhouette score, 0.4967, indicating that it forms well-separated and compact clusters; however, this does not translate into a better differentiation between fake and real news, as shown by the near-zero Rand index, this suggesting that DBSCAN finds a genuine structure in the text, but one which does not correspond to the fake and real news labels.

Table 3. Metrics for clustering

Algorithm	Silhouette	ARI	NMI
K-Means	0.0565 ± 0.0001	0.0237 ± 0.0010	0.0162 ± 0.0007
DBSCAN	0.4967 ± 0.0001	−0.0007 ± 0.0000	0.0081 ± 0.0000

6.3 Performance of the reinforcement learning algorithms

Additionally to the reinforcement learning algorithms transformed in classification and applied to the text with news, these two reinforcement learning algorithms were implemented also in native environment. Therefore, as can be seen in Table 4, DQN, in the CartPole environment reached an average reward of 111.89 on the last 50 episodes, a result that is under the maximum theoretical level of 500 of the CartPole-v1 environment, in this case the episode automatically stops if the stick remains in balance for 500 steps. But it can be said that the result is above the minimum possible level, with a partial learning, not completely convergent, at a number of 300 episodes. SARSA algorithm reached a success rate of 38.16% for the last 500 episodes, in the FrozenLake environment, the result being below the threshold of 70%, which represents the reward level at which this environment is considered solved, according to the Gymnasium registry, this indicating a partial, non-convergent learning within the 5000 training episodes used. Table 4 also shows the total average reward for the entire training, of 0.1664 for SARSA and 63.54 for DQN, both values being lower than the ones calculated only for the last episodes.

Table 4. Performance of SARSA and DQN on native environments

Algorithm	Environment	Main metrics	Total average reward
SARSA	FrozenLake	Success rate (the last 500 ep.) 38.16% ± 3.09%	0.1664 ± 0.007
DQN	CartPole	Average reward (the last 50 ep.) 111.89 ± 40.71	63.54 ± 12.98

6.4 Container performance assessment

In Table 5a, it can be noticed the training time, average and standard deviation for the five repetitions applied for the

eight text-classification algorithms; Table 5b presents the same metrics for the native SARSA and DQN algorithms, which are not directly comparable to Table 5a, since they are trained on small-scale control environments, unrelated in size and nature to the 44,898-document classification task. The native SARSA algorithm was the fastest, with only 10.29 ±1.29 seconds and DQN bandit, the one transformed in classification, obtained a time of 9578.07 ± 751.52 seconds, a significantly high difference from the running time point of view. SVM, Logistic Regression and DBSCAN are grouped in an interval relatively close to 227-313 seconds, but Random Forest obtained a running time of 845.58 seconds, and DeepText obtained 786.32 seconds, this being the fourth highest time used by algorithms for running, after DQN bandit, native DQN and Random Forest. And the native DQN obtained a time of 6297.66 seconds and DQN bandit obtained 9578.07 seconds, these two algorithms confirming that the approximation mechanism with the help of the neural networks and combined with training at every step represents the most expensive computation model from all assessed algorithms.

Also, in Table 5a and Table 5b, it can be noticed the average and maximum use of the processor, as well as the maximum consumption of memory for all algorithms trained in the research. Therefore, the highest memory consumption was registered by algorithm DBSCAN 3293.40 ± 22.67 MB, almost double compared to the following algorithm, DQN bandit, which obtained 1733.36 ± 69.08 MB. This consumption is a consequence of the vicinity calculation, performed by the algorithm for each pair of points in the dataset. The most reduced memory consumption and also the CPU consumption is obtained by SARSA native, this proving the simplicity of the tabular representation without having dependencies of text representation. DeepText and DQN bandit present a similar pattern of CPU use, using 99.81%, respectively 99.74% average CPU, because of the sequential nature of the neural networks training with the help of TensorFlow in the case of this experiment; DeepText's high standard deviation for this metric, ±47.45 percentages, is also notable, and possibly related to the same node contention discussed below for Logistic Regression. Random Forest has a moderate average consumption 75.85% of CPU, but the maximum CPU consumption is high, 187.16%, which proves a pattern of use in stages, because of the blocking on several nuclei and activated only in certain training stages. Logistic Regression has a mean maximum CPU of 561.46%, with a standard deviation, 1082.34 percentages, that exceeds this mean, and this situation can be considered an isolated peak of CPU usage, probably caused by temporary contention on the shared cloud node and not by a structural property of the algorithm.

Based on the image on the whole of the results in Table 5a, it can be said that the training time and the resource consumption are not always correlated, for instance algorithm DBSCAN has a moderate training time of 227.35 seconds, but reached the highest memory consumption of all algorithms, and, on the opposite side, the algorithm DQN bandit has the highest training time, but the memory consumption is only 30% above the one of the classification algorithms. These results indicate that the time and resource consumption, as memory and CPU, must be separately assessed, to determine the infrastructure cost.

Table 5a. Use of resources and execution time of algorithms

Algorithm	Average CPU (%)	Maximum CPU (%)	Maximum memory (MB)	Training time (s)
SVM	47.76 ± 3.86	81.58 ± 5.55	1420.52 ± 53.96	313.03 ± 24.56
Random Forest	75.85 ± 23.52	187.16 ± 25.45	1399.38 ± 70.68	845.58 ± 230.14
Logistic Regression	45.33 ± 2.34	561.46 ± 1082.34	1409.00 ± 38.64	256.72 ± 20.21
DeepText (CNN)	99.81 ± 47.45	214.32 ± 51.58	1220.73 ± 4.63	786.32 ± 397.47
SARSA (bandit)	53.32 ± 6.48	87.12 ± 12.81	1384.07 ± 66.44	359.53 ± 43.83
DQN (bandit)	99.74 ± 3.04	230.20 ± 90.47	1733.36 ± 69.08	9578.07 ± 751.52
K-Means	52.60 ± 8.10	82.16 ± 8.91	1678.64 ± 48.02	367.71 ± 42.73
DBSCAN	82.37 ± 2.40	137.14 ± 18.50	3293.40 ± 22.67	227.35 ± 24.91

Table 5b. Use of resources and execution time of native reinforcement learning algorithms

Algorithm	Average CPU (%)	Maximum CPU (%)	Maximum memory (MB)	Training time (s)
SARSA (native)	34.19 ± 3.78	46.52 ± 7.53	133.97 ± 0.20	10.29 ± 1.29
DQN (native)	97.73 ± 4.66	115.42 ± 1.65	937.80 ± 78.82	6297.66 ± 1578.61

In Figure 3, it can be noticed the graphical representation of the CPU and memory resources consumed during algorithm running assessed for this research.

Fig. 3. CPU and memory usage

In Figure 4 it is represented the running time of the algorithms tested within the research, in order to observe the differences between algorithms more easily, after being containerized with Docker and Kubernetes and trained on Azure Kubernetes Service.

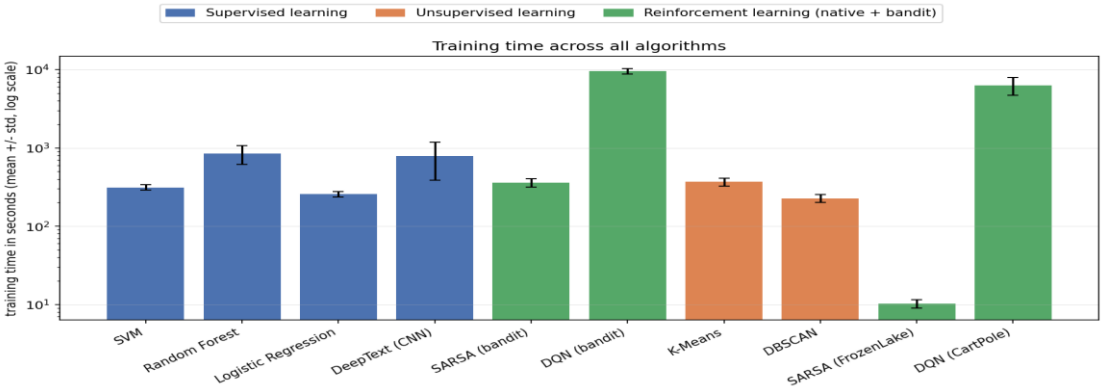

Fig. 4. Algorithm running time

In Table 6 it can be seen a comparison of the execution time for five representative algorithms, ran in four successive testing environments. The first environment is virtual, Python, locally named venv, subsequently the algorithms were run after being containerized with Docker, but without orchestration. Subsequently, the algorithms were run with the Kubernetes orchestrator, on local cluster with a single node Minikube, and, finally, they were run with Kubernetes on the cloud cluster from Azure, with five different seeds for each algorithm. From the table it can be noticed that execution time increased progressively from venv to Docker, and then to Kubernetes locally and, finally, to Kubernetes cloud, for most algorithms; however, this comparison combines several factors that this study cannot fully separate, since the Azure measurements were obtained on different hardware than the local ones, and under concurrent execution, with several parallel repetitions sharing the same cluster resources. The most visible is at SARSA native algorithm, which was the fastest algorithm in this study: it started from a time of 1.02 seconds on venv, remained practically unchanged at 1.01 seconds in Docker, subsequently increasing to 7.00 seconds on Minikube, and, finally, in cloud reaching 10.29 seconds on average. This accentuated increase for this algorithm, in which case, normally, the execution time is not higher than 2 seconds, indicates that the overhead comes from the orchestration costs which contain the pod planning, container starting, attachment of the network volumes, this being at least as plausible an explanation as resource contention on the shared cloud nodes, discussed in Section 6.4 for Logistic Regression and DeepText. And for the algorithms with longer execution time, SVM, K-Means, DBSCAN, SARSA bandit, the overhead introduced by Docker is moderate, reaching about 40% at K-Means, while for SARSA native the difference between venv and Docker remains within measurement noise, while passing to Kubernetes cloud multiplies the execution time compared to the local running. K-Means is a notable exception, being the only algorithm in Table 6 for which Minikube, 443.36 seconds, was slower than Azure, 367.71 seconds on average, the opposite pattern observed for the other four algorithms, this suggesting that the limited resources allocated to the local Minikube cluster, 2 vCPU, 6 GB RAM, may become the dominant factor for computationally heavier algorithms. Another aspect is represented by the fact that, in cloud, we ran the algorithms in five different seeds and we have an average of the running time for each algorithm.

Therefore, the comparison of the four environments suggests a relevant rule for

selecting the infrastructure: for individual or short term tasks, the orchestration cost may exceed the effective algorithm running time. But the real benefits of orchestration, parallelism, reproducibility and tolerance to errors, become useful and visible when the aggregate scale is applied on several simultaneous runs, like the 50 executions, five for each algorithm of the 10 algorithms tested in this study. On a per-run basis, every cloud execution in Table 6 remains slower than the corresponding local execution; but, at the aggregate level, in cloud, with the Kubernetes orchestration, all 10 algorithms were run five times with different seeds, and everything lasted 8 hours, while, on local, only one run of DQN native lasted about 100 minutes, this meaning that five sequential runs of this algorithm alone would have required close to 9 hours. Also, tolerance to errors is an orchestration benefit noticed in this study, when the DBSCAN algorithm failed at the first run attempt, and, subsequently, the execution series was automatically restarted, without human intervention, everything being automatically performed by Kubernetes.

Table 6. Execution time comparison on several environments

Algorithm	Venv (s)	Docker (s)	Kubernetes local – Minikube (s)	Kubernetes cloud – Azure 5 runs on average (s)
SVM	39.80	49.80	71.03	313.03 ± 24.56
K-Means	53.70	74.60	443.36	367.71 ± 42.73
DBSCAN	50.50	63.30	77.00	227.35 ± 24.91
SARSA (native)	1.02	1.01	7.00	10.29 ± 1.29
SARSA (bandit)	62.10	68.50	71.00	359.53 ± 43.83

Figure 5 presents the effective cost of the experiment in Azure cloud, of the used type of resources, in detail. The total cost incurred was 7.75 EUR for the entire 8 hours running period of the experiment. Therefore, it can be noticed that the most expensive were the Virtual Machines, 6.73 EUR, this confirming the fact that the compute resources represent the main cost factor in this type of experiment. The remaining cost is represented by Storage, 0.84 EUR, Container Registry, 0.10 EUR, Virtual Network, 0.07 EUR, and other minor components, as Bandwidth and Load Balancer, both under 0.01 EUR.

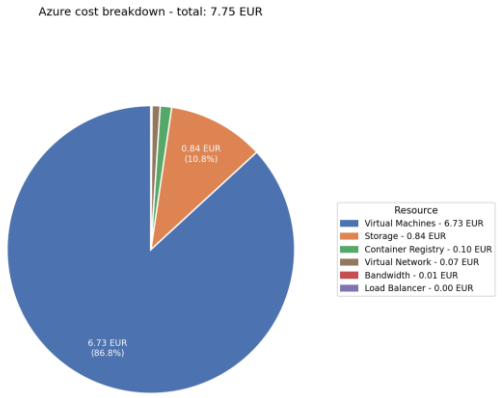

Fig. 5. Cost of resources in Azure cloud

From the experimental study, it can be seen that, in order to have a benefit from the time and cost point of view when running machine learning algorithms, the duration and the number of repetitions of the task matter, because for the fast algorithms, containerization overhead may exceed the effective computation time, but for slow running

algorithms, with more repetitions to do, containerization and orchestration are very useful. For experiment algorithms, as the ones used in this research, running can be done both locally or only with the help of containers, but, in the case of companies, where the data volume is very high and requirements must be completed within a very short time, the solutions based on containers in cloud can be very useful in order to have efficiency and fast benefits, although this extrapolation was not directly tested at that scale in the present study.

7 Conclusions

In this paper, it was presented an empirical comparative analysis of ten algorithms of machine learning, within the three types of supervised, unsupervised and reinforcement learning, through a consistent experiment, using five different seeds: 42, 123, 2024, 7 and 99. Subsequently, these algorithms were run identically in a layered containerized infrastructure: initially they were run in a local virtual environment, Python, then with Docker, subsequently orchestrated with Kubernetes, on a local cluster Minikube and, finally, on a cloud cluster, managed by Azure Kubernetes Service (AKS).

The results obtained show that, in order to calculate the infrastructure cost of an algorithm, it is more important to consider its internal calculation mechanism, rather than the learning type to which it belongs. Also, the tested algorithms of unsupervised learning type, applied directly on lexical representations of the text, cannot identify the fake news from the real ones, without an explicit supervision. Both K-Means algorithm and DBSCAN produced values of the Rand index, adjusted, close to zero, the values for all five algorithm runs remaining unchanged, which shows that these algorithms need supervision to correctly group information. Another aspect is represented by the fact that the overhead of containerization and orchestration is not an exact percentage, but it depends on the duration of the calculation task; if the algorithms are extremely fast, the cloud infrastructure cost can multiply the execution time by approximately tenfold, but for work tasks on a longer period, it can bring significant benefits as the overhead decreases.

In this research there were practically demonstrated the benefits of the Kubernetes orchestration, which include parallelization, tolerance to errors and reproducibility. These can be better noticed when the experiment is complete with several tasks, in our case 50 runs, five for each of the ten algorithms, which were completed in just 8 hours, at a total cost of 7.75 EUR, therefore confirming the economic feasibility of the systematic benchmarking on managed cloud infrastructure; this can be a model that can be replicated in future research with the aim to compare various algorithms in containerized environments.

As a conclusion, the results of this research provide a practical benchmark for the organizations that need to decide what learning algorithm to use for a task and what infrastructure to implement in production, to be more efficient in solving the work tasks. Also, this research represents a starting point for future research, both about the learning types and also about containerized infrastructure.

7.1 Limitations

This study has several limitations. The comparison of execution time between venv, Docker, Minikube and Azure, presented in Table 6, combines the orchestration overhead with two other factors which this research could not fully separate, the different hardware used locally compared to the AKS nodes, and the fact that several repetitions were run in parallel on the cloud cluster, sharing its resources, and a single, controlled sequential run on Azure, which

would isolate the real orchestration overhead from these factors, was not performed within this study.

Also, it can be noticed that the accuracies obtained by the top four classification algorithms, SVM, Random Forest, Logistic Regression and DeepText, are very close to each other, with differences smaller than their combined standard deviation across the five seeds, and since no formal statistical test was applied within this research, these algorithms should be considered as having similar performance on this dataset, and not as being strictly ranked one above the other.

Another aspect is represented by the scale of the experiment, which remains modest, given the fact that the AKS cluster used had only two nodes and eight vCPUs in total, that the dataset contained 44,898 articles, and that the protocol used only five seeds, and it was not verified if the observed patterns, at a larger scale, with more nodes, more data or more repetitions, remain the same.

As a last aspect, the security of the containerized infrastructure was assessed only through hardening practices at image level, such as the non-root user, and through minimal RBAC permissions for the final job, without a formal scan of vulnerabilities.

7.2 Future trends

In the future, we want to extend the analysis beyond the hardening practices used in this study, non-root user, RBAC minimal, to a more serious security assessment, such as automatic scanning of vulnerability of the Docker images, applied to the various Kubernetes admission policies, which block unsecured deployments, and network isolation between tenants in a multi-user cluster.

Also, the cluster used in this research used two nodes with 8 vCPU total and the dataset was of 44,898 articles, which represents a modest scale. In the future, we want to use more data and a cluster with more nodes, in order to verify if the observed patterns are maintained at production scale.

A third research trend we want to explore in greater depth, is the comparison of the containerized infrastructure with other platforms that can perform approximately the same tasks and the discovery of the most efficient solution that can be implemented within companies.

Acknowledgment

This paper was co-financed by the Bucharest University of Economic Studies during the PhD program.

References

[1] J. Gardner, Y. Yang, R. Baker, and C. Brooks, “Enabling End-To-End Machine Learning Replicability: A Case Study in Educational Data Mining,” arXiv preprint arXiv:1806.05208, Jul. 2018.

[2] C. Carrión, “Kubernetes as a Standard Container Orchestrator - A Bibliometric Analysis,” J. Grid Comput., vol. 20, no. 4, p. 42, Dec. 2022, doi: 10.1007/s10723-022-09629-8.

[3] X. Wang, K. Zhao, and B. Qin, “Optimization of Task-Scheduling Strategy in Edge Kubernetes Clusters Based on Deep Reinforcement Learning,” Mathematics, vol. 11, no. 20, p. 4269, Oct. 2023, doi: 10.3390/math11204269.

[4] Z. Zhong, M. Xu, M. A. Rodriguez, C. Xu, and R. Buyya, “Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions,” ACM Comput. Surv., vol. 54, no. 10s, pp. 1–35, Jan. 2022, doi: 10.1145/3510415.

[5] C. Li, A. Dakkak, J. Xiong, and W. Hwu, “The Design and Implementation of a Scalable Deep Learning Benchmarking Platform,” in 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), IEEE, Oct. 2020, pp. 414–425. doi: 10.1109/CLOUD49709.2020.00063.

[6] V. Dakić, G. Đambić, J. Slovinac, and J. Redžepagić, “Optimizing Kubernetes

Scheduling for Web Applications Using Machine Learning," Electronics (Basel)., vol. 14, no. 5, p. 863, Feb. 2025, doi: 10.3390/electronics14050863.

[7] Y. Zhou, Y. Yu, and B. Ding, "Towards MLOps: A Case Study of ML Pipeline Platform," in 2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE), IEEE, Oct. 2020, pp. 494–500. doi: 10.1109/ICAICE51518.2020.00102.

[8] V. Dakić, M. Kovač, and J. Slovinac, "Evolving High-Performance Computing Data Centers with Kubernetes, Performance Analysis, and Dynamic Workload Placement Based on Machine Learning Scheduling," Electronics (Basel)., vol. 13, no. 13, p. 2651, Jul. 2024, doi: 10.3390/electronics13132651.

[9] A.-O. Radu and S.-A. Ionescu, "Comparison of Supervised Learning Algorithms for Fake News Detection," Proceedings of the International Conference on Business Excellence, vol. 19, no. 1, pp. 2139–2148, Jul. 2025, doi: 10.2478/picbe-2025-0166.

[10] D. Choudhury and T. Acharjee, "A novel approach to fake news detection in social networks using genetic algorithm applying machine learning classifiers," Multimed. Tools Appl., vol. 82, no. 6, pp. 9029–9045, Mar. 2023, doi: 10.1007/s11042-022-12788-1.

[11] V. Mnih et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[12] A.-O. Radu, "CONTAINERIZATION OF MACHINE LEARNING ALGORITHMS FOR INTELLIGENT EDUCATIONAL PLATFORMS," ICERI2025 Proceedings, Nov. 2025, pp. 7214–7224. doi: 10.21125/iceri.2025.1974.

[13] C. Cortes and V. Vapnik, "Support-vector networks," Mach. Learn., vol. 20, no. 3, pp. 273–297, Sep. 1995, doi: 10.1007/BF00994018.

[14] L. Breiman, "Random Forests," Mach. Learn., vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324.

[15] D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant, Applied Logistic Regression. Wiley, 2013. doi:10.1002/9781118548387.

[16] M. Ahmed, R. Seraj, and S. M. S. Islam, "The k-means Algorithm: A Comprehensive Survey and Performance Evaluation," Electronics (Basel)., vol. 9, no. 8, p. 1295, Aug. 2020, doi: 10.3390/electronics9081295.

[17] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "DBSCAN Revisited, Revisited," ACM Transactions on Database Systems, vol. 42, no. 3, pp. 1–21, Sep. 2017, doi: 10.1145/3068335.

[18] A. K. Shakya, G. Pillai, and S. Chakrabarty, "Reinforcement learning algorithms: A brief survey," Expert Syst. Appl., vol. 231, p. 120495, Nov. 2023, doi: 10.1016/j.eswa.2023.120495.

Andreea - Oana RADU is a Ph.D. student in Economic Informatics at the Bucharest University of Economic Studies (ASE), Romania. She holds a bachelor's degree and a master's degree in Economic Informatics and currently works in the field of cybersecurity at an institution in the financial sector. Her research interests include artificial intelligence, cloud computing, containerization and orchestration technologies such as Docker and Kubernetes, MLOps, cybersecurity, and Big Data, with a particular focus on security, reproducibility, and resource efficiency in machine learning tasks deployed in containerized infrastructures.

Sergiu - Alexandru IONESCU is a PhD student in Economic Informatics at the Bucharest University of Economic Studies (ASE), Romania. He holds a master's degree in Database Management and Business Intelligence and is a senior professional in the financial sector, with experience in data analytics and controlling. His research interests cover data storage and processing technologies, modern data architectures, Big Data, cloud computing, data governance, and machine learning, with a particular focus on their use in financial institutions and data-intensive environments.

Ioana NAGÎȚ is a PhD candidate at the Bucharest University of Economic Studies, specializing in Business Informatics. Her research focuses on Big Data, Artificial Intelligence, Machine Learning and Blockchain technologies, with a particular interest in applications related to data-driven decision-making, disaster management and smart coordination systems. Alongside her academic work, she has a professional background in software engineering and technical support for enterprise technologies. Her research interests also include data privacy, security, and emerging digital technologies.