Testing the Robustness of Machine Learning Models Through Mutations

Manuel Méndez(✉), Miguel Benito-Parejo, and Mercedes G. Merayo

Design and Testing of Reliable Systems Research Group, Universidad Complutense de Madrid, Madrid, Spain
{manumend,mibeni01}@ucm.es, mgmerayo@fdi.ucm.es

Abstract. The reliable performance of machine learning algorithms stands as a critical and foundational concern. Usually, scientific attention centres solely on this aspect when selecting among models. However, in real-world scenarios, datasets are vulnerable to human errors during data input. Consequently, algorithms must display consistency and resilience against such errors. We assert that, especially in real-world applications, the resilience is, akin to the performance, a critical characteristic when selecting one algorithm over another. To address this concern, we propose a novel methodology for assessing model robustness by evaluating models both before and after applying mutations to the dataset. To validate the effectiveness of this methodology, we analyse five commonly used machine learning algorithms in a case study concerning traffic flow forecasting in Madrid. In assessing the robustness of the models, we introduce two metrics derived from well-known regression measurements. The results clearly reveal that the random forest model shows the highest robustness, according to our analysis, and that different models can exhibit very different behaviours in terms of this aspect.

Keywords: Algorithms Robustness · Data Mutation · Deep Learning · Machine Learning

1 Introduction

In recent years, the field of machine learning (ML) has undergone a transformed evolution, fundamentally altering how data is pre-processed and analysed. The continuous advancement and refinement of ML models has revolutionised several fields, such as medicine, autonomous driving technology, weather forecasting, pollutant tracking or facial recognition systems. However, during this technological revolution, the performance and dependability of ML models rely on the quality and reliability of the data. In real-life cases, datasets may contain a variety of errors, ranging from small inconsistencies to omissions, which can affect the accuracy and reliability of predictions. These errors come from many different sources. For example, mistakes when people insert data, optical character recognition problems or accidental damage to files, can make

This work has been supported by the Spanish MINECO/FEDER project AwESOMe (PID2021-122215NB-C31).

N.-T. Nguyen et al. (Eds.): ICCCI 2024, CCIS 2165, pp. 308–320, 2024.
https://doi.org/10.1007/978-3-031-70248-8_24

data less accurate. Although these errors may seem irrelevant when considered independently, when accumulated they can significantly affect the performance of ML models.

In this work, we analyse the impact of the aforementioned errors on the performance of ML models by inserting mutations in the dataset. We aim at determining the extent to which different ML models are affected by such errors. The ML models that are barely affected by errors will be more robust than the ones that are more sensible to variations in the data. In order to do it, we inject *mutations* in datasets. These mutations simulate potential errors, such as omitting data, skipping decimal points, or repeating previous data points, that may have been introduced in datasets. They will be randomly distributed throughout the predictor variables of the training dataset. Then, we will analyse how different models deal with these kinds of errors. The experiments will be conducted on an existing dataset [19] that was used to predict the hourly traffic flow in a main roadway of Madrid. The *mutated dataset* will be used to train five ML models and the results will be compared by using two different metrics. While our work does not involve the direct application of mutation testing, we are clearly inspired by this technique, particularly in the context of applying mutations to the data. Upon reviewing the scientific literature, we have identified numerous studies that utilise mutation testing in ML, however, they predominantly focus on mutations developed within the code of ML models, as noted by [11,21].

The rest of the paper is organised as follows. In Sect. 2 we review the most relevant and recent contributions related to the topics of the paper. In Sect. 3 we present the dataset in which we will apply our experiments and we briefly describe the ML models that we will compare among them. Moreover, we define the metrics that we will use to assess the quality of the models. In Sect. 4 we describe the mutations and the procedure that we will follow to apply them in our dataset. In Sect. 5 we present the results of our experiments. Finally, in Sect. 6 we present our conclusions and some lines for future work.

2 Related Work

Although our work does not directly apply mutation testing in ML, we consider it important to highlight some milestones achieved through the combination of these two fields. Classification algorithms have been employed to detect whether a given test would detect a mutant [23]. In another investigation, they were also used to solve the known Equivalent Mutant Problem directly with classification algorithms [20]. Similarly, in [4], the authors explore the use of ML algorithms to classify mutants into minimal sets or equivalent categories. Conversely, the combination of both fields has also been common, specifically in the application of mutation testing in the code of ML models, as reflected in [21]. Moreover, in another interesting investigation, a tool named DeepMutation++ enables the identification of vulnerable segments of sequential input through runtime analysis in both feed-forward and recurrent neural networks by applying mutation testing [11].

Another testing technique, metamorphic testing [5,18,22], also involves controlled modifications to the dataset. However, these modifications are based on metamorphic relations, which are defined as relationships between the input and output of a system that remain consistent across these modifications. In [25], algorithms like KNN

and Naive Bayes are evaluated with metamorphic testing. Metamorphic testing has also been included in the analysis of Support Vector Machine [7], to verify classification models using a set of combined statistical test methods in [27] or even to analyse unsupervised ML models [26].

However, it is noteworthy that none of these investigations evaluate the robustness of a model. To address this, various techniques are employed, such as classical cross-validation or bootstrap methods [14], stratified for imbalanced data [29], bias (highly related to underfitting) and variance (high variance related to overfitting) analysis [6, 30] or analysis of the learning curve. The learning curve [24] allows us to appreciate how the model improves as the dataset increases the number of observations. In [12], error detection is performed by leveraging co-occurrence statistics from large corpora of clean tables.

It is our understanding that the utilisation of mutations within the dataset to assess the robustness of the model represents a novel approach in this work.

Finally, concerning the ML models that we have analysed in this work, they all are widely employed in multitude of fields [8, 10, 13, 16, 19].

3 Description of Dataset and Models Employed

In this section, we will outline the main features of the dataset that we have used in our experiments. In addition, we will describe the models used and the metrics against which we will compare them.

The dataset we have used in our experiments is the one utilised in a previous work [19] to forecast hourly traffic flow 12 h in advance on Paseo la Castellana Street, a roadway that runs from north to downtown Madrid, specifically in a north-south direction. This dataset comprises 31380 observations with 9 predictor variables[1] and one target variable associated with each observation. Among the different roadways considered in the previous work, we have chose Paseo la Castellana Street because it experiences the highest traffic volume for most of the day in Madrid. Another reason is that the corresponding dataset contains a very small number of errors, which suits the needs of our experiments. In our case, we want to simulate potential errors in the dataset in a controlled way, not that they are in the original dataset. In Fig. 1, we show the average traffic flow by hour, by day of the week and by month. Note that we omit the first period of the COVID pandemic (from March to June of 2020) because the data during that period were very anomalous.

The experiments have been performed using 5 classical ML models: Linear regression [17], a simple statistical model used to model the relationship between a dependent variable and one or more independent variables by fitting a straight line to the observed data points; Random forest [3], an ensemble learning method that constructs a multitude of decision trees and outputs the mean prediction; Long-short term memory neural networks (LSTM) [28], a type of recurrent neural network architecture designed to capture long-term dependencies in sequential data; Bidirectional LSTM (BiLSTM) [9], a variation of LSTM that processes input sequences in both forward and backward directions

[1] The characteristics of predictor variables can be found at [19].

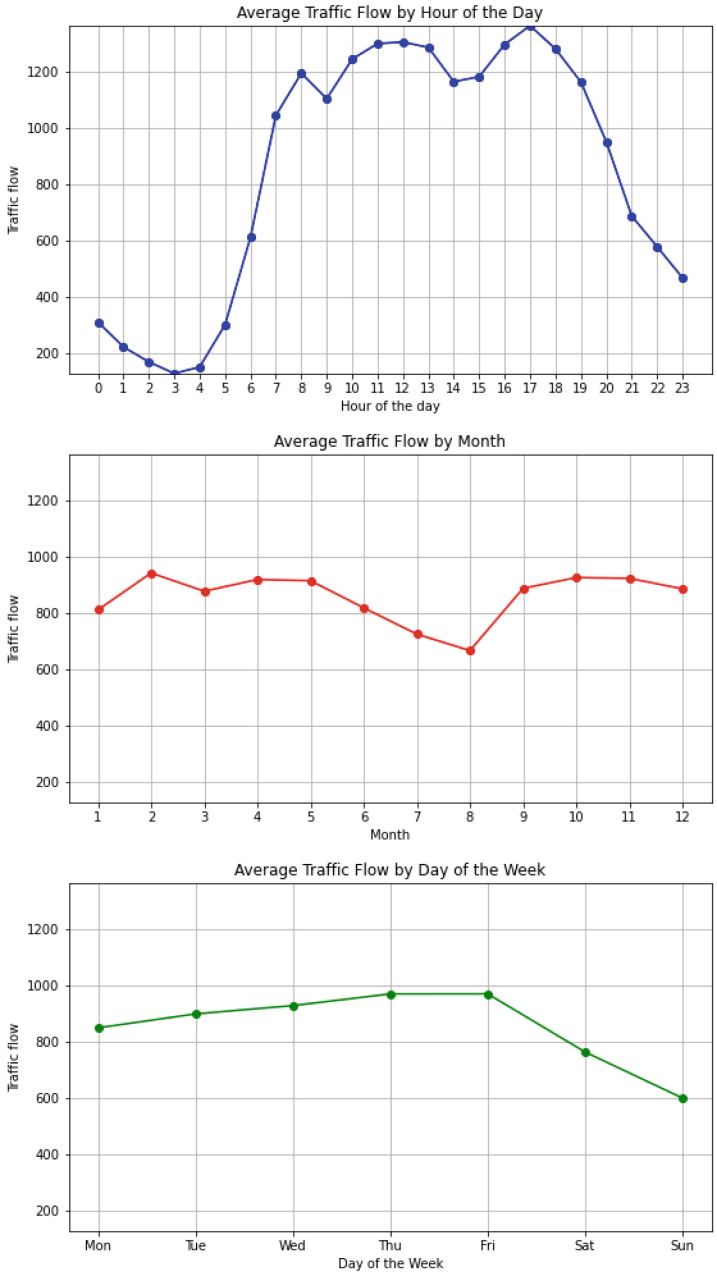

Fig. 1. P/Castellana (N-S) average traffic flow by hour (blue), by month (red) and by day of the week (green), excluding the first quarantine period. (Color figure online)

and the hybrid model developed in [19], Convolutional neural network combined with a Bidirectional LSTM (CNN-BiLSTM), that captures spatial features using convolutional layers and temporal dependencies using bidirectional recurrent layers. In order to assess the robustness of the models, we will utilise two metrics: the relative variance of mean absolute error (MAE) and the relative variance of mean squared error (MSE). In other words, we will examine the percentage variability of each of these metrics before and after applying mutations.

4 Mutations

In this section, we describe the different mutations that we have selected to apply to the dataset, as well as the procedure carried out to inject them. It is worth noting that the mutations have been applied to all the predictor variables in the training set.

In order to define formally the application of the different mutations we have considered to the dataset, we will used the following notation.

Let $(x_1, \ldots, x_p)$ and (y) be the predictor variables and target variable, respectively, of an observation of the training set. Let us consider that the training set contains n observations. We denote by $x_{i,j}$, where $1 \leq i \leq n$ and $1 \leq j \leq p$, the value of the j-th predictor variable associated with the i-th observation and by $x'_{i,j}$ the mutation of that value.

Next, we introduce the 4 kinds of mutations that we have used in our experiments and define how we applied them to the dataset.

– *DEC*: The first mutation simulates mistakes corresponding to either, the omission or the incorrect position of a decimal point. The application of this mutation consists in replacing the original value, $x_{i,j}$, by this value multiplied by a power of 10. This mutation will be only applied on decimal numbers. We have to take into account that if $|x_{i,j}|$ is higher than 1, then $|x'_{i,j}|$ must also be higher than 1. In other case, the error would not be about misplacing or omitting the decimal point, but rather about inserting an additional digit, specifically 0. For example, if you have $x_{i,j} = 18.27$ and the position of decimal point is omitted or changed, then $x'_{i,j}$ could take the values: $1.827, 182.7$ (changing the position of the decimal point) or 1827 (omitting the decimal point). However, if $|x_{i,j}|$ is less than 1, this cannot happen, because if the decimal point is moved to the left, no action (removing the 0) is necessary for the number to have value. For example, if $x_{i,j} = 0.23$ and the decimal point is omitted, then $x'_{i,j}$ will be 023, which represents the same value as 23. In the case that the position of the decimal point is changed and $x'_{i,j} = 02.3$, this value also represents the value 2.3 and it is not necessary to change it. This is the reason that the exponent of the power of 10 that is used to inject the mutations must be less than or equal to the length of the fractional part of $x_{i,j}$. Next we define formally the application of the mutation to the dataset.
  Let $x_{i,j}$ be the value corresponding to a predictor variable of an observation. Let α and β be the length of the whole part and the fractional part of $x_{i,j}$, respectively. Given $e \in \mathbb{Z}$, randomly chosen, such that $-(\alpha - 1) \leq e \leq \beta$ and $e \neq 0$, we define $x'_{i,j} = x_{i,j} \cdot 10^e$.

– *IMP*: The second mutation is related to the values that can be assigned to a variable when the corresponding data is missing. Specifically, there are three options regarding the assigned value: 0, the previous one or the average of the previous data in the same column. Formally, given $x_{i,j}$ the mutated value $x'_{i,j}$ is randomly chosen among the following options:

$$0, \quad x_{i-1,j}, \quad \frac{1}{i-1} \cdot \sum_{k=1}^{i-1} x_{k,j}$$

– *SGN*: This mutation simulates a common mistake: the omission or incorrect placement of the sign. Formally, given $x_{i,j}$ the mutated value $x'_{i,j} = -x_{i,j}$
– *DIG*: The last mutation corresponds to placing a wrong digit in a number. This mutation is only applied to the whole part of the number. This is due to the fact that we consider that if this error affects the fractional part it is irrelevant. However, it would be trivial to extend the application of this mutation to the fractional part if it is necessary, for example, when dealing with normalised data that take values close to 0. Formally, given $x_{i,j}$, Let α be the length of the whole part of $x_{i,j}$ and d_s, where $1 \leq s \leq \alpha$, be the digit corresponding to the sth position of the whole part of $x_{i,j}$, respectively. In this case, we consider d_1 the least significant digit of the whole part and d_α, the most significant digit of the whole part. Given $1 \leq s \leq \alpha$ and $m \in \mathbb{N}$ such that $0 \leq m \leq 9$ and $m \neq d_s$ we have that

$$x'_{i,j} = \begin{cases} x_{i,j} - d_s \cdot 10^{s-1} + m \cdot 10^{s-1} & \text{if } x_{i,j} \geq 0 \\ -x_{i,j} + d_s \cdot 10^{s-1} - m \cdot 10^{s-1} & \text{if } x_{i,j} < 0 \end{cases}$$

Next, we will describe the procedure that we will follow to apply these mutations. In the training set, we will randomly apply mutations to 500, 5000, and 20000[2] (respectively, to the $0.059\%, 0.59\%$ and 2.36% of the total) numerical cells. We will apply the same number of mutations of each type. Specifically, in the case of the *IMP* mutation, we will apply the possible values randomly, therefore, we will not have the same number of mutations for each of them. The code has been written in Python and is available in https://github.com/MMH1997/MutationOnDataset.

5 Experiments

In this section, we will describe our experiments. First of all, it is important to remark that in some rare cases, the mutations may not be applied as expected. For instance, if the *IMP* mutation is applied to a value equal to 0, no change will occur. Another example, if the *IMP* mutation uses the value corresponding to the previous observation and this value is equal to the value to which the mutation is being applied, no change will be observed. Similarly, if the value to apply in the mutation is the average of the values of the previous observations, the original value will not exhibit any change. Moreover,

[2] In accordance with the explanations provided in Sect. 5, when we refer to "20000 mutations", we are describing the application of 2000 mutations to one-tenth of the entire dataset, with the outcomes scaling equivalently to applying 20000 mutations across the entire dataset.

when a value is single-digit, the *DEC* mutation is not possible. These situations only occur about 10% of cases according to our experiments.

The selected hyperparameters for the models are similar but not exactly the same ones as those employed in [19]. In the Random forest model, we will use 25 trees, a maximum depth of 10, and a minimum number of 10 samples in a node before splitting it. In the neural network models, we use the same hyperparameters and structure as in the aforementioned work, except for the number of units in the LSTM or BiLSTM layers. In the previous work, the number of units was 500, but in this analysis, we use 50. In both models we use a seed in order to ensure that the results of the experiments are consistent and comparable across different executions. Finally, the linear regression model does not have any hyperparameter to consider.

We conducted the experiments after preprocessing the data, following nearly all the steps outlined in [19]. These steps involve transforming all data into numerical format. Subsequently, the data is converted from a sequence to pairs of input (predictor variables) and output (target variable), effectively transforming a time series problem into a supervised ML problem. Following this modification, the total number of cells in which mutations can be applied rises to 847152. After conducting preliminary experiments, we observed that the normalisation of data did not significantly impact the results. Therefore, we decided to avoid the normalisation step performed in the original work. This ensures that mutations directly affect the raw data.

In all the experiments, we utilised the same original and mutated training sets, as well as the same testing set, with a $75-25$ split. The results will be presented by model according to the predefined metrics. For the initial set of 500 mutations, we conducted 10 experiments for each model to ensure the consistency of our assessment. Regarding the second set of 5000 mutations, we performed 4 experiments. This distribution is based on the need to analyse whether the mutations directly affect the model, in the first case, while in the subsequent cases, the impact is already established, and we focused on understanding how it affects the model. Finally, for the last set, we also conducted 4 experiments. However, after confirming that the results before and after the mutations are scalable, we decided to proceed with this approach. That is, we randomly selected one-tenth of the dataset and apply 2000 mutations instead of 20000. This allowed us to reduce computational time while obtaining optimal results.

It is worth emphasising that our evaluation focuses on the robustness of the models rather than their performance. While one model may exhibit a lower (better) mean absolute error than another model initially, the percentage difference in mean absolute error of the first model after applying mutations may be greater than that of the second. In such instances, we would consider the second model to be more robust.

5.1 Linear Regression

In the case of the linear regression model, it is noteworthy to mention that due to the large number of predictor variables, the model does not perform well with this data. As can be seen in Fig. 2, the initial set of mutations has minimal impact on the metrics with an increment just due to randomness. However, with 5000 mutations, the model, on average, performs 4.257% worse in MAE and 4.994% in MSE than the original model. This can be considered a significant change. Finally, when the number of mutations is

	Linear Regression			
	MAE		MSE	
	Av.	Std.	Av.	Std.
500	0.516	0.217	0.205	0.159
5000	4,257	2,538	4,994	3,104
20000	9,568	4,972	14,530	7,863

Fig. 2. Percentage difference in the MAE and MSE (average and standard deviation) obtained by Linear Regression model concerning the number of mutations applied to the dataset.

20000, these values are almost tripled. In conclusion, even with less than around 1% of the data mutated, this algorithm exhibits significant changes.

5.2 Random Forest

	Random Forest			
	MAE		MSE	
	Av.	Std.	Av.	Std.
500	0.116	0.212	-0.837	0.442
5000	-0.704	0.683	-2,278	1,541
20000	2,016	1,034	3,811	2,715

Fig. 3. Percentage difference in the MAE and MSE (average and standard deviation) obtained by Random Forest model concerning the number of mutations applied to the dataset.

Unlike the previous case, the random forest model performs well with this data. Figure 3 shows that the application of the two initial sets of mutations has no impact on the performance of the model. In fact, the impact is so small that in some cases, the average value after these applications is better than the value in the original dataset. After the application of 20000 mutations (mutations in around 2.35% of the total data), the change starts to be significant, with a MAE 2.016% and a MSE 3.811% worse than in the original case. In summary, the significant resistance to overfitting, the tree diversity, and the effective handling of noisy or outlier-laden data due to the ability to average the results of this model, make it highly robust.

5.3 LSTM Neural Network

The LSTM model performs well with this data. As shown in Fig. 4, a pattern can be detected: the results obtained with the original dataset are better than the ones produced in the case of the mutated dataset regardless of the number of mutations. This implies that even when mutating only 0.059% of the data, the performance of the model undergo significant modifications. The changes observed in MSE are consistently higher than those observed in MAE. We conclude that the robustness of the LSTM model is extremely poor. The high sensitivity to hyperparameters, combined with its propensity for overfitting, may be the reasons behind the poor robustness exhibited by LSTMs.

	LSTM			
	MAE		MSE	
	Av.	Std.	Av.	Std.
500	8,816	7,906	17,001	14,053
5000	15,097	11,783	29,040	23,335
20000	9,544	10,352	17,133	14,380

Fig. 4. Percentage difference in the MAE and MSE (average and standard deviation) obtained by LSTM model concerning the number of mutations applied to the dataset.

5.4 BiLSTM Neural Network

	BiLSTM			
	MAE		MSE	
	Av.	Std.	Av.	Std.
500	2,177	5,342	4,971	9,473
5000	1,986	1,468	1,914	1,551
20000	5,298	3,438	6,086	8,114

Fig. 5. Percentage difference in the MAE and MSE (average and standard deviation) obtained by BiLSTM model concerning the number of mutations applied to the dataset.

The BiLSTM also performs well with this data. Initially, in the case of the initial set of mutations, Fig. 5 exhibits certain inconsistencies because the results when 5000 mutations are applied are better than the ones obtained when 500 mutations are applied. However, this can be explained by overfitting, which also accounts for the high standard deviations, and randomness. These results indicate that although BiLSTM model shows worse results when 500 mutations are applied than in the original case, the variation between 500 and 20000 mutations is not significantly high, indicating a moderate variation. This variation is much lower than in the case of LSTM, likely due to the ability of the bidirectional layer to capture complex contextual information, thereby enhancing robustness compared to LSTM.

5.5 CNN-BiLSTM Neural Network

This model presents the best performance among those analysed [19]. Similarly to the previous case, Fig. 6 shows that the increase in percentage difference in MAE and MSE is not directly proportional to the increase in the number of mutations applied to the dataset. However, in this case, the values are smaller. The high robustness of CNN-BiLSTM compared to BiLSTM, or even more so compared to LSTM, can be attributed to the improvement obtained by the CNN layer in capturing local characteristics in the data.

	CNN+BiLSTM			
	MAE		MSE	
	Av.	Std.	Av.	Std.
500	1,993	3,460	3,970	2,469
5000	2,712	2,621	3,691	3,757
20000	4,477	3,823	3,459	4,031

Fig. 6. Percentage difference in the MAE and MSE (average and standard deviation) obtained by the hybrid CNN-BiLSTM model concerning the number of mutations applied to the dataset.

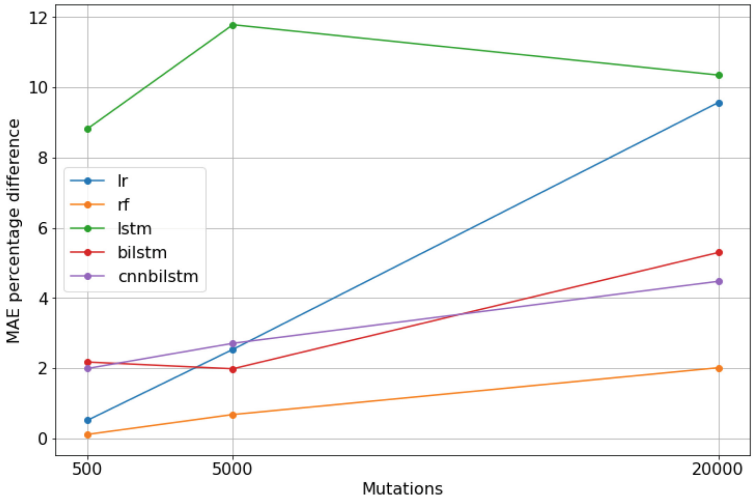

Fig. 7. MAE percentage difference by number of mutations applied and models.

5.6 Discussion

In this section, we compare the results obtained from our experiments for the different models. As depicted in Fig. 7 and Fig. 8, that summarise the percentage difference of MAE and MSE results, respectively, we observe a relative correlation between both metrics. However, there are variations in the results of different models.

In the case of linear regression, the percentage difference increases linearly as the number of mutants increases. Random forest remains almost constant, and the increment of the percentage difference as the number of mutants increases is very slight, not exceeding 2 in the MAE case and 4 in the MSE. LSTM obtains the highest percentage difference in both MAE and MSE. These values are very high, making it the least robust model among those analysed. BiLSTM and CNN-BiLSTM show similar behaviours, with slightly smaller percentage differences in the hybrid case.

In conclusion, LSTM is clearly the least robust model, regardless of the number of mutations applied to the data. Linear regression shows a high dependence on the number of mutations, being very robust when the number of mutations applied is small and much less robust when it is high. Finally, random forest, BiLSTM, and CNN-BiLSTM

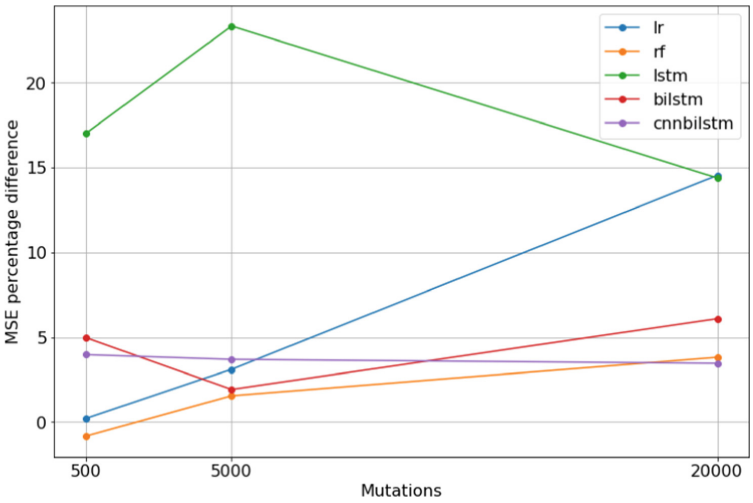

Fig. 8. MSE percentage difference by number of mutations applied and models.

do not seem to have much dependence on the number of mutations applied, at least in the terms we are considering. Random Forest is the most robust model, followed by the hybrid CNN-BiLSTM and the BiLSTM model.

6 Conclusions and Future Work

In this paper, we have proposed a method to evaluate the robustness of machine learning models by applying mutations to the datasets on which the models are trained. Our experiments show that there are significant differences in the robustness among different models. Across the models analysed, we conclude that Random Forest is the most robust model, while LSTM is the least robust.

We consider several avenues for future work. First, we aim to extend this study to other models by analysing datasets of different types and conducting a larger number of experiments. Additionally, we seek to enhance the characteristics of the mutations performed, adapting them to real-world scenarios more effectively. For instance, we would prioritise errors where a '3' is mistaken for an '8' over errors where a '3' is mistaken for a '7'. Similarly, we would prioritise errors where the decimal point is omitted or shifted slightly from its original position over errors where the decimal point is shifted by several places from the original. Moreover, it would be interesting to differentiate between mutations in terms of their detrimental effects in order to delve deeper into the behaviour regarding errors of the models.

Finally, we would like to translate this research into real-world applications [15], as well as formal validation ones [1,2]. Specifically, we aim to develop an interface where users can choose between different models, weighing the well-performance and robustness of the analysed models according to their preferences.

References

1. Benito-Parejo, M., Merayo, M.G.: An evolutionary algorithm for selection of test cases. In: 2020 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (2020)
2. Benito-Parejo, M., Merayo, M.G.: Using genetic algorithms to select test cases for finite state machines with timeouts. In: 2021 IEEE Congress on Evolutionary Computation (CEC), pp. 2403–2410 (2021)
3. Breiman, L.: Random forests. Mach. Learn. **45**, 5–32 (2001)
4. Brito, C., Durelli, V.H.S., Durelli, R.S., de Souza, S.R.S., Vincenzi, A.M.R., Delamaro, M.E.: A preliminary investigation into using machine learning algorithms to identify minimal and equivalent mutants. In: 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 304–313 (2020)
5. Chen, T.Y., et al.: Metamorphic testing: a review of challenges and opportunities. ACM Comput. Surv. **51**(1) (2018)
6. Dietterich, T.G.,, Kong, E.B.: Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Department of Computer Science, Oregon State University, Corvallis (1995)
7. Dwarakanath, A., et al.: Identifying implementation bugs in machine learning based image classifiers using metamorphic testing. In: 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, pp. 118–128. ACM (2018)
8. Galkina, A., Grafeeva, N.: Machine learning methods for earthquake prediction: a survey. In: Proceedings of the Fourth Conference on Software Engineering and Information Management SEIM 2019 (2019)
9. Graves, A., Fernández, S., Schmidhuber, J.: Bidirectional LSTM networks for improved phoneme classification and recognition. In: Duch, W., Kacprzyk, J., Oja, E., Zadrożny, S. (eds.) ICANN 2005, pp. 799–804. Springer, Heidelberg (2005)
10. Himeur, Y., et al.: AI-big data analytics for building automation and management systems: a survey, actual challenges and future perspectives. Artif. Intell. Rev. **56**, 1–93 (2022)
11. Hu, Q., Ma, L., Xie, X., Yu, B., Liu, Y., Zhao, J.: Deepmutation++: a mutation testing framework for deep learning systems. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 1158–1161 (2019)
12. Huang, Z., He, Y.: Auto-detect: data-driven error detection in tables. In: Proceedings of the 2018 International Conference on Management of Data, SIGMOD 2018, pp. 1377–1392. Association for Computing Machinery, New York (2018)
13. Kaur, K., Kaur, P.: The application of AI techniques in requirements classification: a systematic mapping. Artif. Intell. Rev. **57**, 02 (2024)
14. Lasfar, R., Tóth, G.: The difference of model robustness assessment using cross-validation and bootstrap methods. J. Chemomet. **2014**, e3530 (2024)
15. Méndez, M., et al.: Combining metamorphic testing and machine learning to enhance openstreetmap. IEEE Trans. Reliab. 1–15 (2024)
16. Méndez, M., Merayo, M.G., Núñez, M.: Machine learning algorithms to forecast air quality: a survey. Artif. Intell. Rev. **56**(9), 10031–10066 (2023)
17. Montgomery, D.C., Peck, E.A., Vining, G.G.: Introduction to Linear Regression Analysis. Wiley (2021)
18. Méndez, M., Benito-Parejo, M., Ibias, A., Núñez, M.: Metamorphic testing of chess engines. Inf. Softw. Technol. **162**, 107263 (2023)
19. Méndez, M., Merayo, M.G., Núñez, M.: Long-term traffic flow forecasting using a hybrid CNN-BiLSTM model. Eng. Appl. Artif. Intell. **121**, 106041 (2023)
20. Naeem, M.R., Lin, T., Naeem, H., Liu, H.: A machine learning approach for classification of equivalent mutants. J. Softw. Evolut. Process **32**(5), e2238 (2020)

21. Panichella, A., Liem, C.C.S.: What are we really testing in mutation testing for machine learning? a critical reflection. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), pp. 66–70 (2021)
22. Segura, S., Fraser, G., Sanchez, A.B., Ruiz-Cortés, A.: A survey on metamorphic testing. IEEE Trans. Softw. Eng. **42**(9), 805–824 (2016)
23. Strug, J., Strug, B.: Machine learning approach in mutation testing. In: Testing Software and Systems, vol. 7641, pp. 200–214. Springer, Heidelberg (2012)
24. Viering, T., Loog, M.: The shape of learning curves: a review. IEEE Trans. Pattern Anal. Mach. Intell. **45**(6), 7799–7819 (2023)
25. Xie, X., Ho, J.W.K., Murphy, C., Kaiser, G., Xu, B., Chen, T.Y.: Testing and validating machine learning classifiers by metamorphic testing. J. Syst. Softw. **84**(4), 544–558 (2011). The Ninth International Conference on Quality Software
26. Xie, X., Zhang, Z., Chen, T.Y., Liu, Y., Poon, P.-L., Xu, B.: Mettle: a metamorphic testing approach to assessing and validating unsupervised machine learning systems. IEEE Trans. Reliab. **69**(4), 1293–1322 (2020)
27. Xu, L., Towey, D., French, A.P., Benford, S., Zhou, Z.Q., Chen, T.Y.: Using metamorphic relations to verify and enhance artcode classification. J. Syst. Softw. **182**, 111060 (2021)
28. Yu, Y., Si, X., Hu, C., Zhang, J.: A review of recurrent neural networks: LSTM cells and network architectures. Neural Comput. **31**(7), 1235–1270 (2019)
29. Zeng, X., Martinez, T.R.: Distribution-balanced stratified cross-validation for accuracy estimation. J. Exp. Theor. Artif. Intell. **12**(1), 1–12 (2000)
30. Zhou, Y., Wu, J., Wang, H., He, J.: Adversarial robustness through bias variance decomposition: a new perspective for federated learning. In: Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM 2022), pp. 2753–2762. Association for Computing Machinery, New York (2022)