Data Quality Report Project

Salvatore Buono

December 2023

Student code: 10600540

Assigned DQ issue: **Variable type**

Assigned ML task: **Regression**

1 Introduction

The primary objective of this project is to develop a sophisticated data pollution function designed to introduce variable type errors into a given dataset. These errors will extend various data types, including categorical, numeric, and boolean types. By injecting these intentional inaccuracies, we aim to assess and analyze the robustness and resilience of different regression algorithms when confronted with such data pollution.

The data pollution function will be meticulously crafted to simulate real-world scenarios where datasets may be prone to inaccuracies or inconsistencies in variable types. This intentional introduction of errors will allow us to evaluate the perfomance, speed and overfitting of regression algorithms in handling diverse data scenarios.

2 Setup choices

In the data collection phase, I decided to build a dataset for regression using the scripts provided by teachers. These scripts utilized the `make_dataset_for_regression` function from *scikit-learn*. I created a dataset with 12 features to ensure that the pollution function is more understandable and to reduce computational cost. Out of these 12 features, only 4 are informative, meaning that only these 4 features are utilized during the training of regression algorithms.

In order to gain a better understanding of how different algorithms behave as the number of samples increases, I decided to conduct 10 different experiments using my data pollution functions. These experiments ranged from a dataset containing 1000 tuples to a dataset with 2000 tuples. This approach helps me comprehend the correlation between regression algorithms and pollution function, since for each iteration I apply the pollution function with same percentage but with increasing number of tuples.

3 Pipeline implementation

3.1 Data analysis

In preparation for the development of the data pollution function, a crucial preliminary step involves gaining a comprehensive understanding of the underlying data structure. To accomplish this, I initiated an analysis of the dataset, delving into key aspects such as the diverse data types present, the range of values embedded, and the distribution of these values throughout the dataset.

Upon completing this preliminary examination, it became evident that all features within the dataset are of a numerical nature, specifically floating-point variables. Notably, each value within the dataset is unique, and collectively, they exhibit a range from -1 to 1.

While these insights provide valuable information about the dataset's basic characteristics, they offer only a surface-level comprehension. To delve deeper and introduce a layer of complexity for a more fine-grained analysis, the next step involves injecting noise or pollution into the dataset. This deliberate introduction of variability will enable a more thorough exploration of the dataset's resilience and the robustness leading to a more comprehensive understanding of the data's behavior in realistic, imperfect conditions.

3.2 Data pollution

3.2.1 Completely at random data pollution function

This function takes a percentage factor of pollution that should be injected and returns a new polluted dataset. The pollution function injects different variable types (boolean, categorical, and numeric) according to a random number that selects the pollution type to be injected. I created various categorical features such as name, surname, favorite color, age, and a boolean feature which can represent if is subscribed to a service or newsletter (true) or not (false).

This function operates entirely at random, meaning that values are not assigned based on corresponding values in the original dataset. Since there is no correspondence between the original value and the value injected into the dataset, this results in outcomes that are not easily readable, as explained in the subsequent section.

To address this readability issue, I have decided to create a new pollution function that injects values depending on the range in which the original value falls.

3.2.2 Range-based pollution function

In order to build this function is necessary to **scale** value because I want to ensure that the function behaves consistently across different ranges of values. If data isn't scaled, the same amount of "pollution" could have a big impact on smaller values and a small impact on larger values. This could lead to inconsistent results and make it harder to understand the effects of the pollution. I used `MinMaxScaler` which scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one. This function is built using the following criteria:

- Maps a given value to a corresponding color based on predefined ranges (fixed to 10%).

- Maps a given value to a corresponding name based on predefined ranges (fixed to 20%).
- Maps a given value to a corresponding surname based on predefined ranges (fixed to 20%).
- Maps a given value to *True* if it is higher than 0.5, otherwise *False*.
- Create a new feature *full name* which is the concatenation of name and surname, if both are present. This introduce correlation between features.

Moreover, this function allows people to have multiple names and surname, so when create the *full name* feature, it concatenates all the names and surnames.

4 Results

The pollution function introduces variable type errors into a dataset, which can affect the performance, speed, and overfitting of different regression algorithms. The behavior of different regression algorithms can vary when considering different pollution factors and the use of completely at random data pollution function and range-based pollution function. In particular the regression algorithms deepened in this project are:

- *K-Nearest Neighbors (KNN) Regression*: KNN regression is a non-parametric method that approximates the association between independent variables and the continuous outcome by averaging.
- *Linear Regression*: Linear regression is a statistical procedure used to predict the value of a dependent variable based on the value of one or more independent variables. It models the relationship between these variables by fitting a linear equation to the observed data.
- *Bayesian Ridge Regression*: Bayesian Ridge Regression is a variant of linear regression where we start with an estimate (the prior value) and as we include additional data points, the accuracy of our model improves. It differs from Ordinary Least Squares (OLS) regression in that it incorporates a prior probability for the model parameters, which is updated as more data points are collected.
- *Multi-Layer Perceptron (MLP) Regression*: MLP regression is a type of neural network model that can be used for regression tasks. It consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Each node in the layers (except for the input layer) uses a nonlinear activation function, which allows the model to learn complex patterns in the data.
- *Support Vector Machine (SVM) Regression*: SVM regression, also known as Support Vector Regression (SVR), is a type of SVM that supports linear and non-linear regression. The model produced by SVR depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction.

- *Gaussian Process Regression (GPR)* is a kernel method, in particular we can say that is the kernel version of Bayesian Linear regression approach. A Gaussian Process is defined as a probability distribution over functions $y(x_i)$: given an input we have a Gaussian distribution as target and not a deterministic point.

When using the completely at random data pollution function 1, which injects pollution randomly without considering the original values, the performance of the regression algorithms may be affected. The pollution can introduce inaccuracies in the dataset, leading to a decrease in the performance of the regression models.

Algorithm	Mean Performance	Distance	Speed
	MSE	$MSE_{pred} - MSE_{fit}$	s
Bayesian Ridge	77.5778	3.5619	0.1321
GPRegressor	103.9500	103.3332	1.6564
KNNRegressor	106.2096	19.6152	0.1586
LinearRegressor	87.5540	5.9666	0.0595
MLPRegressor	92.2433	23.0518	9.2118
SVMRegressor	77.1604	4.1079	0.0552

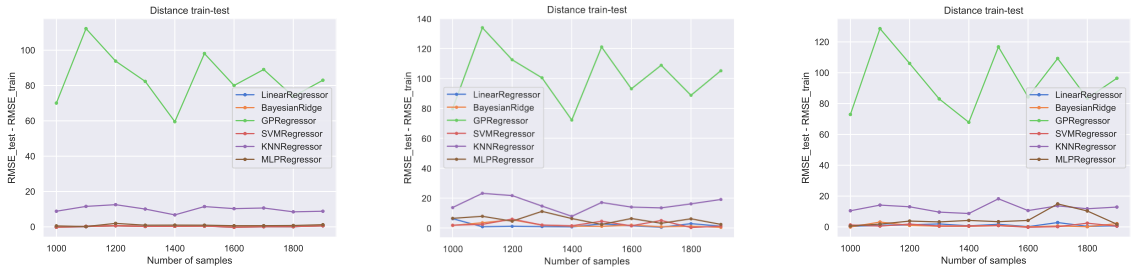
Table 1: Summary Results of Random pollution function

On the other hand, when using the range-based pollution function 2, which injects pollution based on the range of the original values, the behavior of the regression algorithms may differ. This pollution function considers the original values and injects pollution accordingly, which can result in a more understandable and interpretable dataset with better performances with respect to completely at random data pollution function.

Algorithm	Mean Performance	Distance	Speed
	MSE	$MSE_{pred} - MSE_{fit}$	s
Bayesian Ridge	34.142216	4.348499	0.435829
GPRegressor	99.756613	99.102938	2.514167
KNNRegressor	84.621472	15.884865	0.155992
LinearRegressor	36.279888	4.477498	0.088263
MLPRegressor	57.784192	19.032268	12.417402
SVMRegressor	32.178972	2.717416	0.418856

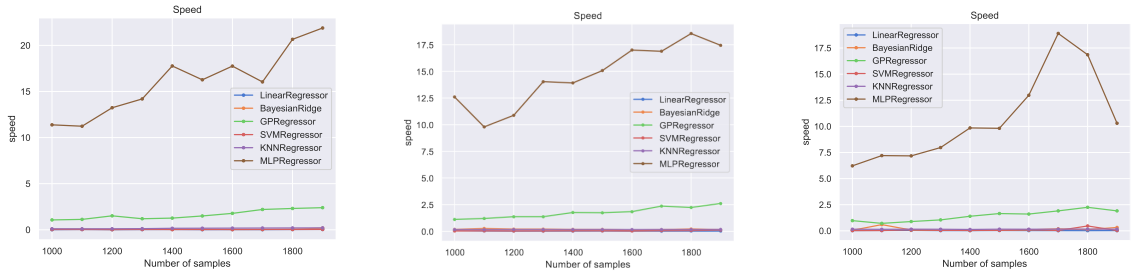
Table 2: Summary Results of Range-based pollution function

4.1 Results with low pollution factor



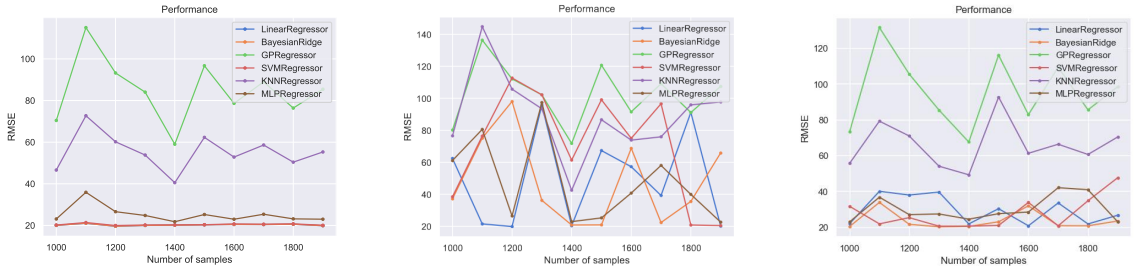
(a) No pollution (b) Random pollution (c) Not random pollution

Figure 1: Difference in distance train-test with 30% factor



(a) No pollution (b) Random pollution (c) Not random pollution

Figure 2: Difference in speed train-test with 30% factor



(a) No pollution (b) Random pollution (c) Not random pollution

Figure 3: Difference in performance with 30% factor

The pollution function introduces variable type errors into a dataset, which can affect the performance, speed, and overfitting of different regression algorithms.

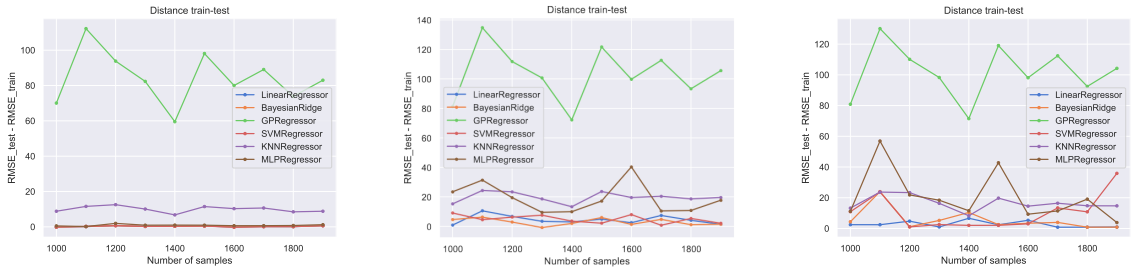
When the pollution function is low, **the distance between the train and test error** (Figure 1) for different regression algorithms can vary. For instance, the *LinearRegressor* and *BayesianRidge* algorithms show a smaller distance between train and test error, indicating that these models may be less prone to overfitting. For example Bayesian Ridge regression, which uses Bayesian inference and includes regularization parameters, can be more robust to noise and outliers introduced by the pollution function. On the other hand, the *SVMRegressor*, *KNNRegressor*, and *MLPRegressor* show a larger distance, suggesting a higher risk of overfitting. This is because they are more complex, since they have a higher capacity to capture

intricate patterns in the training data, potentially leading to overfitting when the model becomes too flexible and starts fitting noise in the data. The *GPRegressor* is already overfitting (Figure: 1a) and the pollution function increase the distance between train and test error. This is because we have a small dataset and the GP regressor may struggle to discern true patterns from noise. The pollution function, by introducing errors, further increase overfitting, causing the model to memorize noise patterns during training. The completely at random pollution function introduces pollution without considering the original values, which can lead to a larger distance between train and test error. This is because the pollution may introduce inconsistencies that the algorithm has not encountered during training. On the other hand, the range-based pollution function injects pollution based on the range of the original values, which can result in a smaller distance between train and test error. This is because the pollution is introduced in a way that is more consistent with the original data distribution.

The **speed** (Figure: 2) of the regression algorithms can also be slightly affected by the pollution function. In general, simpler models like *LinearRegressor* and *BayesianRidge*, *SVMRegressor* and *KNNRegressor* tend to be faster, while more complex models like *GPRegressor* and *MLPRegressor* can be slower due to their computational complexity. The completely at random pollution function injects pollution randomly, without considering the range of the original values. This can result in a faster pollution process since there is no need to analyze the original values. On the other hand, the range-based pollution function requires analyzing the range of the original values to determine the pollution to be injected. This additional analysis can slow down the pollution process. However, the impact on the overall speed of regression algorithms will depend on the specific implementation and computational resources available.

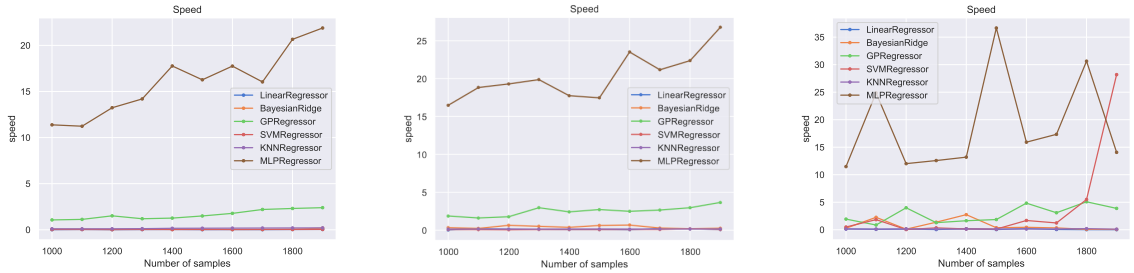
The **performance** (Figure: 3) of the regression algorithms, measured by the Root Mean Square Error (RMSE), can also be influenced by the pollution function. For instance, the *MLPRegressor* shows a relatively small decrease in performance, indicating that it may be more robust to data pollution. *MLPRegressor*, a type of neural network, can model complex, non-linear relationships, potentially making it more robust to the effects of high pollution. The impact on performance will depend on the specific regression algorithm and the nature of the pollution introduced. In general, the range-based pollution function may provide a more realistic representation of data inconsistencies, which can help evaluate the performance of regression algorithms in handling diverse data scenarios.

4.2 Results with high pollution factor



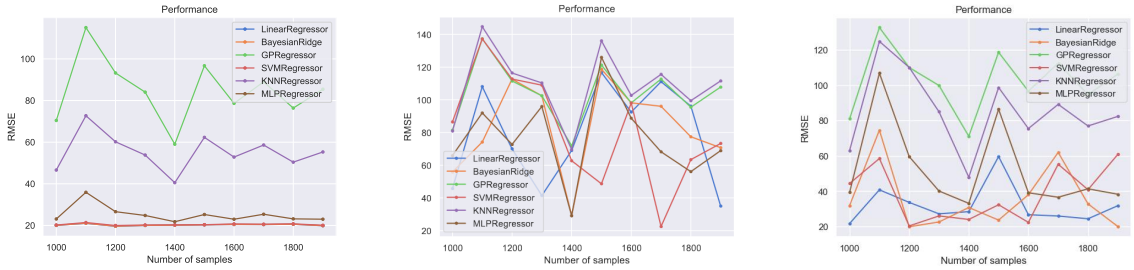
(a) No pollution (b) Random pollution (c) Not random pollution

Figure 4: Difference in distance train-test with 60% factor



(a) No pollution (b) Random pollution (c) Not random pollution

Figure 5: Difference in speed train-test with 60% factor



(a) No pollution (b) Random pollution (c) Not random pollution

Figure 6: Difference in performance with 60% factor

In the high pollution factor section, the graphs reveal distinct behaviors of regression algorithms when confronted with increased data pollution. Unlike the low pollution factor scenario, where algorithms like *LinearRegressor* and *BayesianRidge* showed a smaller **distance between train and test error** (Figure 4), suggesting better generalization and less overfitting, the high pollution context presents a different picture. With a 60% pollution factor, the distance between train and test error for all algorithms, including *LinearRegressor*, *BayesianRidge*, *GPRegressor*, *SVMRegressor*, *KNNRegressor*, and *MLPRegressor*, is notably larger. This increased distance indicates a higher likelihood of overfitting, where the models are fitting not only the underlying data patterns but also the noise introduced by the pollution.

The **speed** (Figure: 5) of the regression algorithms is also impacted by the high pollution factor. While simpler models like *LinearRegressor* and *BayesianRidge* maintain relatively faster speeds, more complex models such as *GPRegressor* and *MLPRegressor* experience a slowdown. This is due to the additional computational complexity required to process the polluted data, which is more pronounced at higher pollution levels.

Performance (Figure: 6), measured by RMSE, deteriorates across all regression algorithms with high pollution. The *MLPRegressor*, which previously showed resilience to data pollution at lower levels, now exhibits a significant performance drop, indicating that even robust models like neural networks are not immune to the effects of high pollution. This performance degradation is a result of the algorithms' struggle to make accurate predictions when the data is heavily corrupted by pollution.

In conclusion, the high pollution factor leads to larger gaps between train and test errors, indicating a greater risk of overfitting, slows down the speed of more complex regression algorithms, and generally decreases the performance of all models due to the increased difficulty in learning from highly polluted data.