POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Data and Information Quality Project

Student: Davide Esposito
Assigned task: Completeness/Regression

UID: 10907319
Project_ID: 49

Academic year: 2023-2024

---

1. Task

The following project aims to study and analyze the Completeness Data Quality Dimension for a regression task developed on a set of different regressors. In particular, the project involves the setting up of 10 different experiments, within which it is necessary to simulate distributions of missing values (MCAR and MNAR values) and analyze the results before and after the imputation phase by varying the number of missing values injected.

1.1. MCAR and MNAR simulation

As will be described in the section relating to the implementation of the different functions, the MCARs will be generated through a uniform distribution between the different characteristics (columns) of the dataset generated based on the type of experiment in question, given that in MCAR, the probability that a subject has a missing value on a certain feature does not depend on its unobserved value nor on the observed values of other variables.
Instead, the MNARs will be generated following a very precise rule: once a certain limit (threshold) and a characteristic (column) of the data set are defined, the values of that column that are greater than the pre-established limit will be missing with a certain probability. This rule was implemented to simulate the fact that, in MNAR data, the probability of missing values depends on the variable's own value.

2. Setup choices

Within this section, the main implementation decisions taken for the entire project and for the individual experiments carried out will be discussed.

2.1. Data collection phase

The different datasets used for each experiment were generated using the *datasets.make_regression* function of the *sklearn* library. Below is a brief description of the prototype of the function with the parameters involved and used during the development of the experiments.

```
klearn.datasets.make_regression(n_samples, n_features, n_informative, n_targets, bias,
                      effective_rank, tail_strength, noise, shuffle, coef, random_state)
```

Compared to the settings originally provided, the only parameters not modified are *n_targets*, *bias*, *shuffle* and *random_state* (in paragraph **seed** the use and importance of this parameter is explained). The remaining

parameters, however, were modified appropriately during the initialization of the different experiments to vary the datasets generation as much as possible.
Below is a brief interpretation of the parameters and some default settings for all experiments:

- *n_samples* and *n_features*. To allow an easier distribution of the missing values generated among the features, the total number of features has been increased to 4. Consequently, by generating more complex datasets, the number of total samples generated has also been increased to 5000.
- *effective_rank* controls the number of singular vectors that have non-zero singular values.Actually, it controls the effective rank of the feature matrix, representing the number of informative features that are linearly independent. It affects the multicollinearity among features. In particular, if *effective_rank* is set to None (default value), then there is no multicollinearity. This parameters are modified only during the last three experiments.
- *noise* parameter represents the standard deviation of the Gaussian noise applied to the output. It allows to control the level of noise in the generated dataset. it was set to 1 for all the experiment to introduce a small noise that reflects variations in the data that may occur in real datasets.
- *coef* parameter, if set to TRUE, the coefficients of the underlying linear model are returned. As we will see later, this parameter will be very useful in the analysis phase to understand which of the generated features are informative and which are not.

2.2. Experiments settings

Each experiment is characterized by the following decisions:

- **Data generation**
- **Data pollution**
- **Data Imputation**

As regards data generation, a table is shown below with the different parameter settings of the *make_regression* function (some of the default parameters are inserted for greater completeness and to allow you to view the parameters as a whole).

Experiments Data generation

	n_samples	**n_features**	**n_informative**	**effective_rank**
E1	5000	4	4	None
E2	5000	4	2	None
E3	5000	4	2	None
E4	5000	4	2	None
E5	5000	4	2	None
E6	5000	4	2	None
E7	5000	4	2	None
E8	5000	4	4	3
E9	5000	4	4	2
E10	5000	4	4	1

Table 1: *make_regression* settings for experiment

To better understand the reason for these configurations, we must take a step forward and consider the different types of data pollution associated with the different experiments. The experiments are all separate from each other, however the main forms of pollution were evaluated simultaneously on multiple experiments to evaluate their effectiveness and draw important results in the subsequent analysis phase. In particular, the following table presents the type of missing values injected in each experiment.

Experiments Data pollution

	Missing values	
	MCAR	**MNAR**
E1	X	-
E2	X	-
E3	X	-
E4	-	X
E5	-	X
E6	X	X
E7	X	X
E8	X	-
E9	X	-
E10	X	-

Table 2: Data pollution for each experiment. The **X** indicates if the typology is injected.

After having reported the main settings for each experiment in tabular form, here is an overall summary for each experiment trying to understand the reasons for the choices and how to then group the different experiments:

- **Experiment 1:** Simulation of MCAR values through an uniform distribution across all the features (for this experiment the features are all informative), assigning 0.25 of probability to each of them.
- **Experiment 2:** Simulation of MCAR values through an uniform distribution across all the **informative features** (for this experiment two features are informative, two non-informative), assigning 0.50 of probability to each of them.
- **Experiment 3:** Simulation of MCAR values through an uniform distribution across all the **non-informative features** (for this experiment two features are informative, two non-informative), assigning 0.50 of probability to each of them.
- **Experiment 4:** Simulation of MNCR values on a **non-informative** feature.
- **Experiment 5:** Simulation of MNCR values on an **informative** feature.
- **Experiment 6:** Simulation of both MNCR values on a **non-informative** feature and MCAR through an uniform distribution across all the features, assigning 0.25 of probability to each of them.
- **Experiment 7:** Simulation of both MNCR values on an **informative** feature and MCAR through an uniform distribution across all the features, assigning 0.25 of probability to each of them.
- **Experiment 8:** Simulation of MCAR values across the feature, assigning 0.25 of probability to each of them. Here, *effective_rank* has an high value, than we introduce a **small multicollinearity** among the features.
- **Experiment 9:** Simulation of MCAR values across the feature, assigning 0.25 of probability to each of them. Here, *effective_rank* has a medium value, than we introduce a **moderate multicollinearity** among the features.
- **Experiment 10:** Simulation of MCAR values across the feature, assigning 0.25 of probability to each of them. Here, *effective_rank* has a low value, than we introduce a **strong multicollinearity** among the features.

Remember that all the experiments are repeated many time with an incremental number of missing values injected from 5% to 50% and increasing step of 5%, with the exception of experiments 4 and 5 with an increase of up to 35%.

3. Data Imputation

The imputation techniques and methodologies used for the different experiments are the following, listed from the simplest to the most complex in terms of implementation:

- **Random imputation**
- **Single imputation (w/ mean, median and most_frequent value)**
- **KNNImputer (Model-based Imputation)**
- **KNN_MICE and BayesianRidge_MICE (Multiple Imputation by Chained Equations)**

In the section 5 the imputation methods used for each experiment will be specifically discussed and only the best results will be reported subsequently.

4. Pipeline implementation

The work pipeline used for this project is the same provided by the guidelines and reported below.

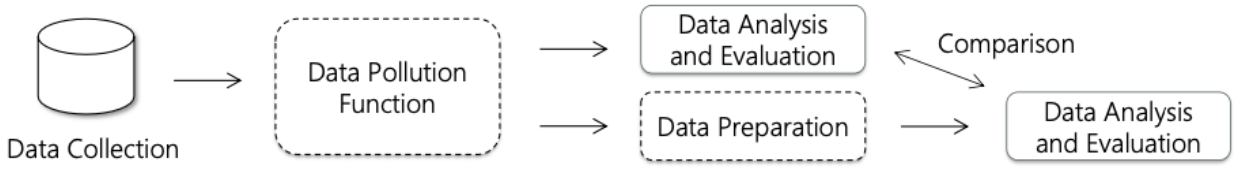


Figure 1: Workflow pipeline

As regards the Data Collection phase and Data Analysis and Evaluation phase, the functions had already been provided, the only thing to change were the different settings for the dataset generation for each experiment (as already described previously).

4.1. The magic of the seed

A very important issue that has influenced the entire workflow is that associated with **seed** parameter. Since we are not working on a real data set, but rather a generated one, it is possible to exploit the setting associated with the seed parameter to always generate the same data distribution and therefore repeat the same work blocks several times without obtaining variations.
Furthermore, in addition to the data generation itself, the seed can also be used to always obtain the same distributions of missing values injected into the datasets. This very powerful tool, therefore, allows us to repeat the same work steps without the risk of carrying out the same operations but on different data. Therefore, the seed was set for the entire duration of the project. This also wants to justify the fact that during the organization of the pipeline, the same data pollution was defined twice for two different analyses, obtaining exactly the same result as the pipeline shown in the figure above.

4.2. Data Pollution Functions

For the different experiments, different pollution functions were implemented to inject the different types of missing values. Below are the main ones with a brief description of their implementation.

- `delete_values_with_uniform_distribution(df, delete_percentage, feature_distribution, seed)`

  This function simulates the MCAR values inside the *df* dataframe, injecting missing values with a frequency equal to *delete_percentage*. In particular, it generates MCAR values across the feature of the dataframe based on the assigned probability distribution *feature_distribution* .

- `generate_MNAR_values_with_uniform(df, delete_percentage, feature, percentile, probability_of_missing, seed)`

  This function simulates the MNAR values inside the *df* dataframe on the specified *feature*. In particular, all the values above a certain threshold computed using the *percentile* value are delete with a certain probability equal to *probability_ of_ missing*. Furthermore, the method takes into account also the number of missing values to inject.
  The following figure shows the result of the function applied to the first feature of the dataset using the *missingno* library.

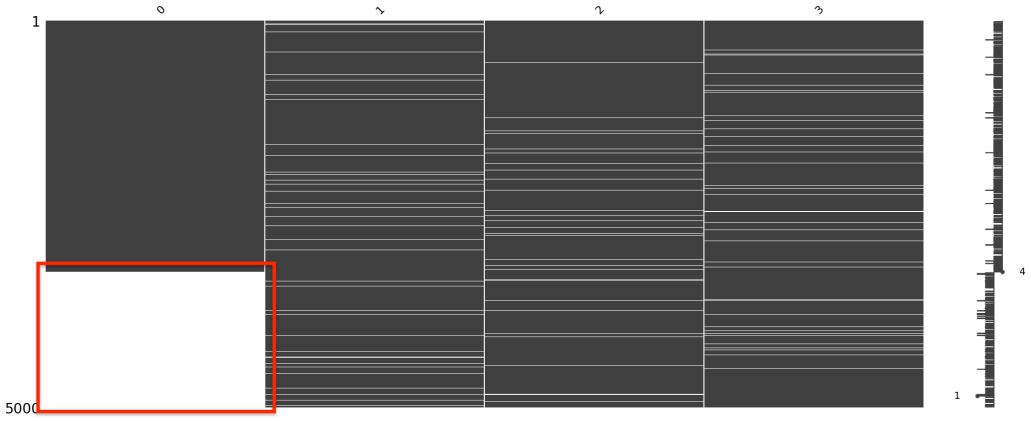


Figure 2: MNAR values visualization

4.3. Data Imputation Functions

Regarding the Data Preparation part, several methods have been defined that use the different algorithms and imputation techniques provided by the *sklearn* library. The presentation of these functions is left aside, as they would be an explanation of the library's methods, which is accessible from the documentation that will be left in the References section.

4.4. Pipeline Structure

All experiments follow the same implementation order, this for greater readability and flexibility, so as to be able to create a basic pipeline for each experiment, which can then modify the individual phases based on the objective you want to achieve. The block structure used for each experiment is therefore presented below.

4.4.1 Variables definition block

In this first block, all variables and settings used throughout the experiment are initialized. For example, the regression models used are defined, the increment on the number of missing values to be injected for each iteration, the probability distributions on the features, etc...

4.4.2 Data Collection block

The data collection block involves the implementation of the *make_ regression* function for the generation of the dataset associated with the experiment in question according to the methods presented in the previous section. Furthermore, within this section the informative features are investigated, which are then used for the various injections of missing values.

4.4.3 Data Pollution block

Once the dataset has been generated, the missing values are injected according to the settings defined for each experiment. Here, the functions described in the paragraph 3.2 and the variables initialized in the first operating block are used.
In particular, the pollution operation is repeated many times with different % of missing values, generating as many datasets as the total number of steps to reach the maximum %. (for example from 5% to 50%, with an increasing step of 5%, 10 datasets will be generated). Obviously, each dataset generated is stored in a list which will then be used by the next phase.

4.4.4 Pollution Evaluation block

As mentioned previously, the evaluation functions are already provided previously. Therefore, this operational block consists in iterating over the different regressors used for the analysis, each of them over all the previously created datasets.
The evaluation functions will return different results for each model and for each dataset with different % missing values. These values will be saved for subsequent plots.

4.4.5 (Repeated) Data Pollution block

Immediately after the pollution evaluation, this block recovers the original dataset (previously saved specifically), and reconstructs the datasets with the same missing values injection carried out previously. The reason for this implementation choice is explained in the paragraph 3.1 and also because the evaluation functions provided did not allow the use of the same datasets constructed in the first pollution phase.

4.4.6 Data Imputation and Evaluation block

In the same way as the datasets were evaluated within the Pollution Evaluation block, here the reconstructed datasets are evaluated by the different regressors, based on the data imputation carried out. In particular, the various data imputation functions are implemented and the results are saved immediately after the evaluation.

4.4.7 Results Evaluation block

Within the last block, pre-provided functions are called to plot the different results, comparing the performance of the different regressors, both on datasets without data imputation and on those with.

5. Results

In the last section, the main results obtained from the analysis of the different experiments are presented, analyzing and interpreting the different numerical and graphical results. In particular, the results shown will indicate the performance levels on the training set using the Negative Root Mean Squared Error (neg-RMSE) as a metric. Other performance measures were not reported in the report as they are not of fundamental importance for revealing the reported results.

5.1. Data Imputation Results

- A first important result regarding the data imputation phase is that obtained from the implementation of the *drop_row_above_threshold()* function, which eliminates all instances of the dataset that contain a number of NaN values greater than or equal to half the number of features + 1. This is done to avoid the model being able to train on instances entirely generated through imputation.
  This result was verified in the experimental phase, noting a clear improvement in performance, regardless of the generated dataset or imputation technique used. Therefore this operation was validated as a basic empirical rule for the imputation phase and inserted by default within each experiment.

- Random data imputation, as one might expect, drastically reduces the predictive performance of the model, both in the training and testing phases.

- KNNImputer and KNN_MICE have been shown by multiple results to be less effective than the other methods used. Therefore these two were removed from the rating.

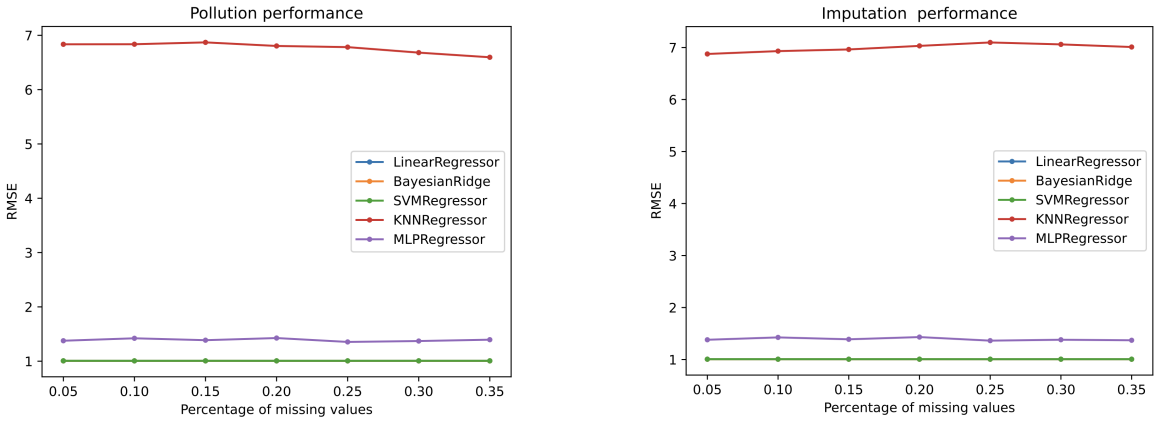
- Simple Imputation with the mean produces slightly better results than the other two (median and most_frequent). Therefore the average is always used as the default statistical value. This result is probably due to the fact that we generated datasets that have a distribution very close to the normal one, therefore the average value is much more effective than the other two. In particular, the results reported for the first 7 experiments are those obtained using the Simple Imputation method with the average.

- BayesianRidge_MICE performs much better than KNN_MICE. In particular, BayesianRidge_MICE achieves the same performance reported by Simple Imputation on the first 7 experiments. Instead, regarding experiments 8,9 and 10 it reports greater performance compared to Simple Imputation method.

5.2. Experiments {4,5,6,7} analysis

As already mentioned in the previous paragraph, simple imputation with averaging was used for this set of experiments, being the method that provided the best results.
On this set of experiments, the imputation method did not prove to be an optimal method for managing MNAR, as it did not bring any effective improvement (if not minimal for high percentages of missing values). This is to point out that when faced with missing values of the MNAR type, to obtain consistent improvements, it could be more useful to inspect these missing values and understand the reasons for them using knowledge and information of the context in question. Obviously this type of approach, given that we are working on generated data, and which therefore have no real meaning, is not applicable.
Despite this, some interesting considerations emerged on how these missing values influence performance on the type of dataset we have in front of us.

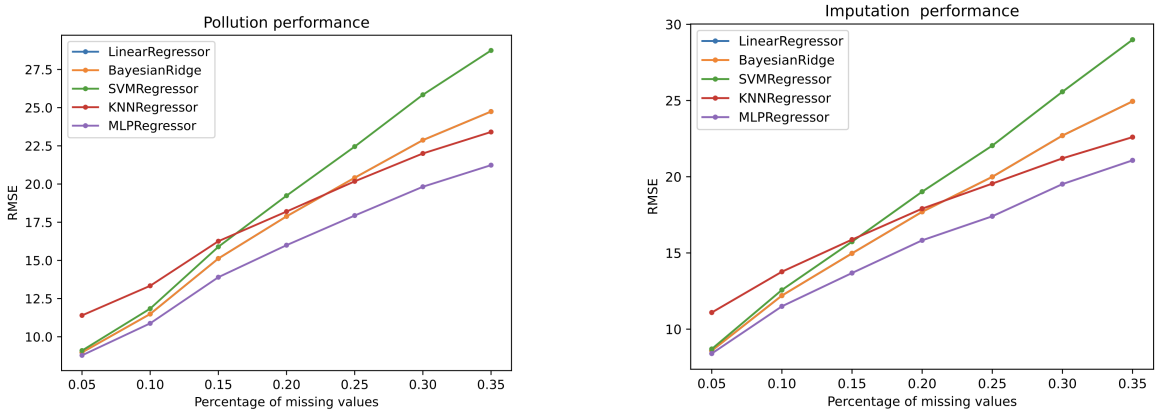
Experiment 4 shows a slightly better level of performance in the training phase after performing imputation on the missing values. In experiment 4, having missing values on a non-significant feature does not greatly affect performance (as shown in figure 3, the error remains constant at a very low value for almost all regressors).



(a) performance on pollution phase (b) performance on imputation phase

Figure 3: Training performance experiment 4

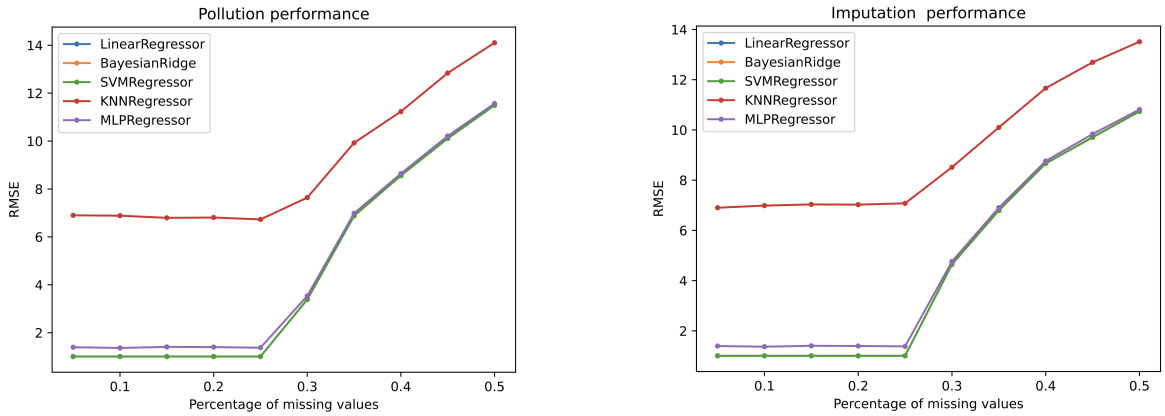
However, by inserting missing values along an informative feature, performance drops dramatically as the % of missing values increases, as is the case in Experiment 5 (Figure 4).



(a) performance on pollution phase (b) performance on imputation phase

Figure 4: Training performance experiment 5

As proof of this, in experiment 6, where we have a mix generation between MNAR and MCAR (in particular as the percentage of missing values increases, the amount of MCAR also increases), we have an explosive increase in error. This is justified because, at the beginning of the experiment, we are generating MNAR only along the non-informative feature and when we start generating MCAR values along all the features (including the informative one) the error starts to increase (Figure 5).

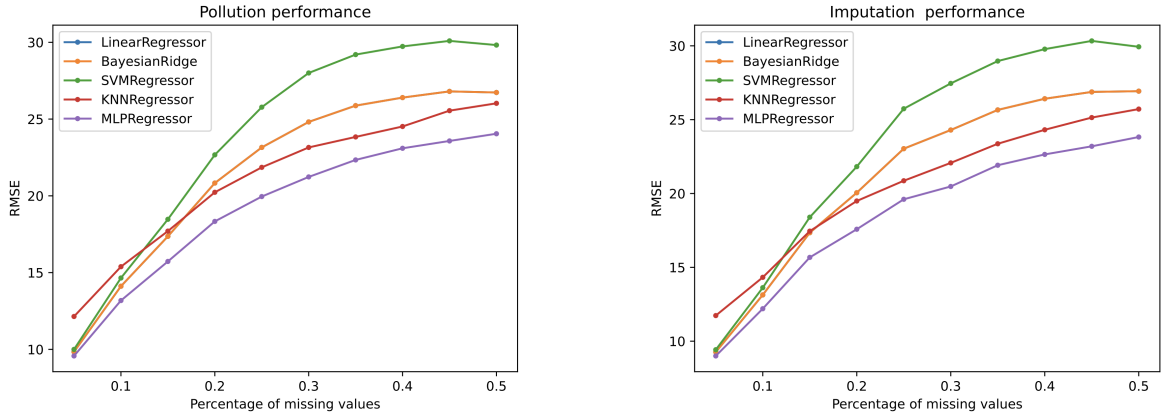


(a) performance on pollution phase

(b) performance on imputation phase

Figure 5: Training performance experiment 6

In experiment 7, however, the MNARs are generated on an informative feature and for this reason the error explodes from the beginning of the iteration and gradually increases as the missing values generated increase (Figure 6).



(a) performance on pollution phase

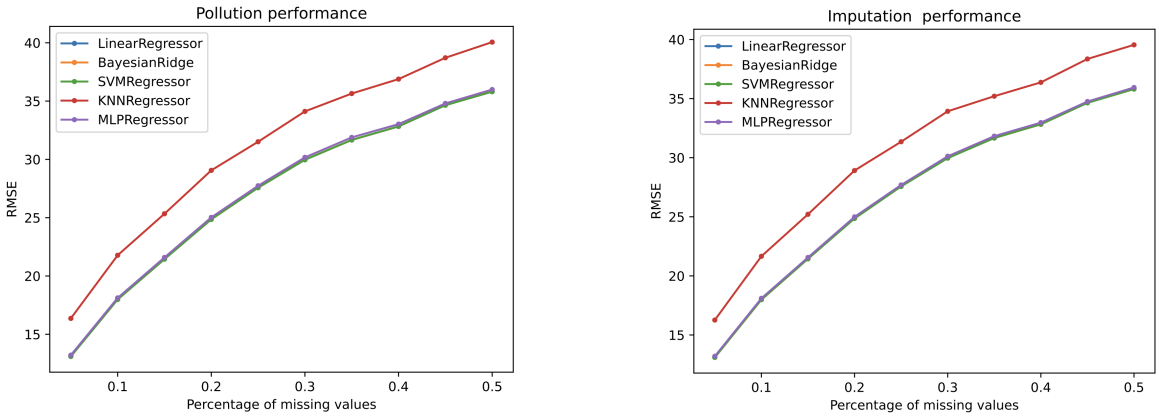
(b) performance on imputation phase

Figure 6: Training performance experiment 7

The result found is that it does not matter the quantity of missing values present, but rather the quality of these. In particular, the weight associated with a missing value for an informative feature is worth much more than missing values for non-informative features.

5.3. Experiments {1,2,3} analysis

Another interesting aspect is the fact that the error measured with the simulations of MCAR values on the informative features of the dataset in experiment 2 (Figure 7) is greater than the error measured with the simulation of MNAR values also on informative features in experiment 5 (Figure 4).



(a) performance on pollution phase

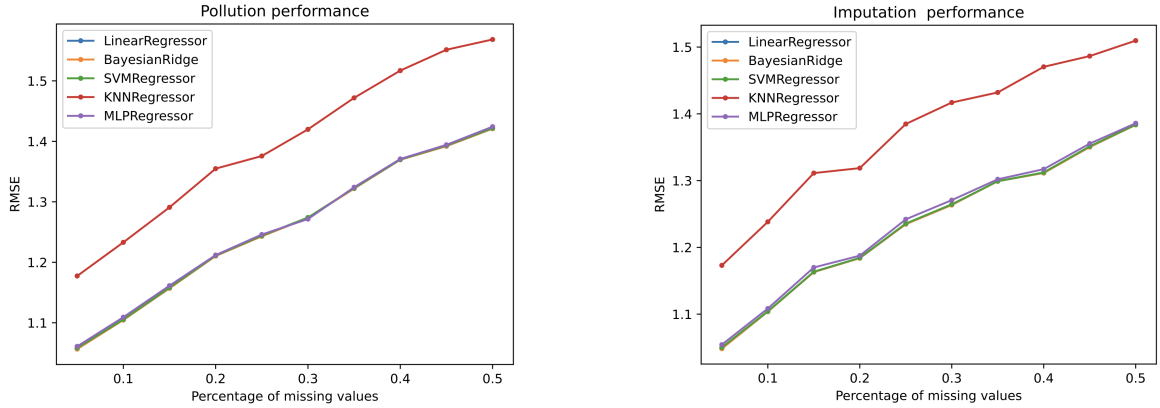
(b) performance on imputation phase

Figure 7: Training performance experiment 2

Most likely, this result is due to the fact that another important aspect is related to how these missing values are distributed. Infact, MCAR values present a completely random distribution compared to MNAR values, increasing the possibility to lose a lot of different values that contain different, but important information.

5.4. Experiments {8,9,10} analysis

Compared to the first two sets of experiments, this one gives more attention on the imputation part, since the generation of the datasets for these experiments looks only at the multicollinearity property among the features. First of all, as previously mentioned, BayesianRidge_MICE is the only imputation method that improved performance between the two phases. Instead, all the others tended to maintain the same result, if not worse. Another interesting result is that although for all three experiments there are only informative features and the missing values were injected uniformly across all of these, the error level remains particularly low (Figure 8). The results of the remaining experiments are not plotted since they have similar behaviors.



(a) performance on pollution phase

(b) performance on imputation phase

Figure 8: Training performance experiment 8