ACADEMIC YEAR 2023/2024

POLITECNICO
MILANO 1863

Data and Information Quality Project

Dimensionality and Completeness

Flaminia Telese - 10931641
Francesco Colabene - 10675235
Group Id 24

Prof. Cinzia Cappiello
Prof. Camilla Sancricca

Dimensionality....4
Setup Choices....4
Results - fixed # of features....6
Experiment 1....6
Experiment 2....6
Experiment 3....7
Experiment 4....7
Experiment 5....7
Experiment 6....8
Experiment 7....8
Experiment 8....8
Experiment 9....9
Experiment 10....9
Results - fixed # of samples....10
Experiment 1....10
Experiment 2....10
Experiment 3....10
Experiment 4....11
Experiment 5....11
Experiment 6....11
Experiment 7....12
Experiment 8....12
Experiment 9....12
Experiment 10....13
Completeness....14
Setup Choices....14
Pipeline Implementation....15
Results....16
Experiment 1....16
Experiment 2....17
Experiment 3....17
Experiment 4....18
Experiment 5....19
Experiment 6....19
Experiment 7....20
Experiment 8....20
Experiment 9....21
Experiment 10....21

Dimensionality

Setup Choices

To assess the effect of the dimensionality for a regression task, we studied two scenarios: fixed number of features and variable number of samples and vice versa.

For the first case, we initially tried to understand if there was any upper limit about the number of samples. After that, we selected a wide range of sample values, to represent different situations, and created polluted datasets to see the effect of the dimensionality in different conditions.

# of samples	`[300, 400, 550, 750, 100, 1300, 1600, 2000, 2500, 3000]`

Each experiment after the first two is repeated twice: every dataset has 5 features, the first time the number of informative features is equal to the number of features while the second time the number of informative features is 2. Each experiment uses the same seed, 2023.
With a fixed number of features and a variable number of samples we conducted ten different experiments:

1. Standard dataset with a low number of samples: from 100 to 1500.
2. Standard dataset with a high number of samples: from 2000 to 20000.
3. Standard dataset with the array of values above.
4. Same as 3, but with different informative features.
5. Dataset with parameter effective rank=500 and tail strength=0.5.
6. Same as 5, but with different informative features.
7. Dataset with parameters bias=1.0, noise=2.0.
8. Same as 7, but with different informative features.
9. Dataset with parameters bias=1.0, noise=2, effective rank=500 and tail strength=0.8.
10. Same as 9, but with different informative features.

For the second case, we selected a fixed number of samples (1000) and a standard vector of features and tested similar situations. This time the number of informative features is not fixed, since the number of features change as well. Each experiment uses the same seed: 2023.

# of features	`[2, 3, 4, 6, 8, 10, 14, 18, 24, 30]`
# of informative features	`[1, 2, 3, 4, 5,  6,  9, 12, 17, 22]`

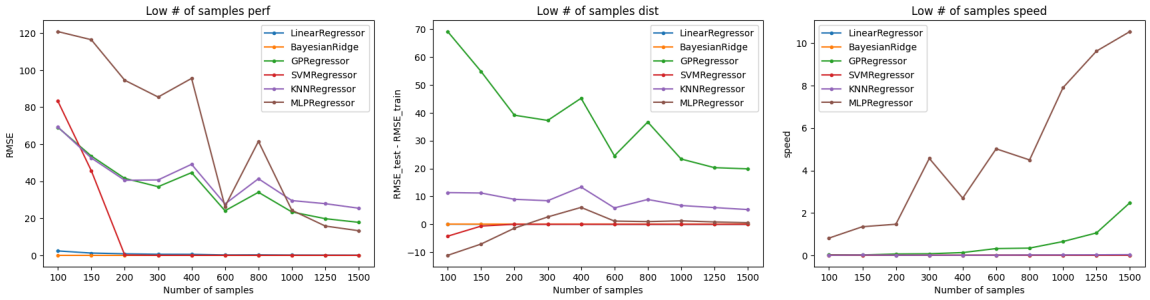
With a fixed number of samples and a variable number of features we conducted ten

different experiments:

1. Standard dataset with the number of informative features equals one of the features.
2. Same as 1, but with different informative features
3. Dataset with parameter noise=2.0.
4. Same as 3, but with different informative features.
5. Dataset with parameters bias=1.0.
6. Same as 5, but with different informative features.
7. Dataset with parameter effective rank=500 and tail strength=0.5.
8. Same as 7, but with different informative features.
9. Dataset with parameters noise=2.0, bias=1.0.
10. Same as 9, but with different informative features.

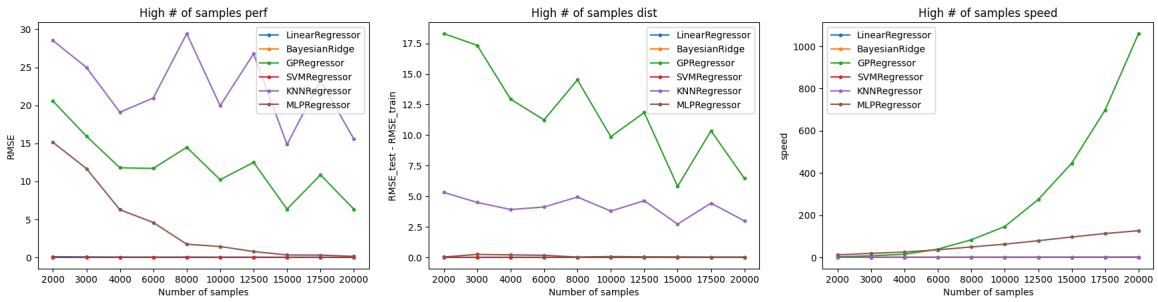
Results - fixed # of features

Experiment 1

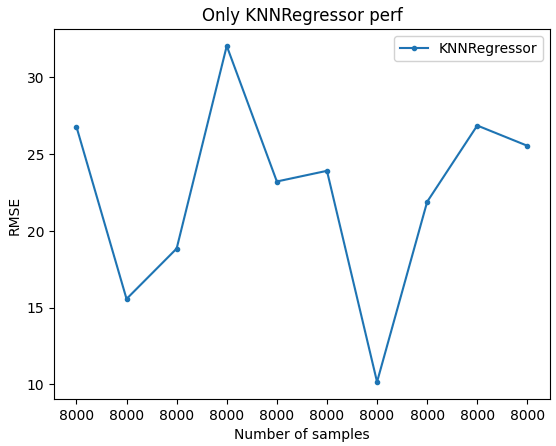


In the above figures we can observe the results of the analysis performed on the dataset of the first experiment. We can see that the RMSE lowers as the number of samples increases for every algorithm, with the biggest improvement coming from the MLP regressor. The MLP is also the worst performer with a lower sample size, because neural networks usually require more training data.

Experiment 2



Here we increased again the number of samples. The RMSE of the MLP approaches zero, while the KNN and the GP regressor bounces up and down because of randomness. We can see it better in this example:



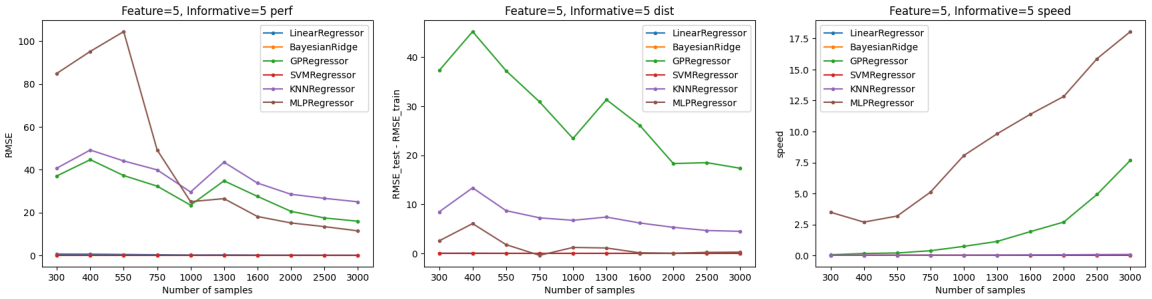
When repeating the same experiment with different seeds to generate the dataset, the RMSE varies a lot.

The time required for the GP regressor is growing exponentially, while the MLP is growing linearly. We can see that the RMSE lowers as the number of samples increases for every algorithm, with the biggest improvement coming from the MLP regressor.

Every test has been conducted on Codalab, and values after 21000 made the server crash, so we stopped here with the analysis. As expected, the more we increase the number of samples, the lower the RMSE becomes, with biggest improvements for the MLP regressor.

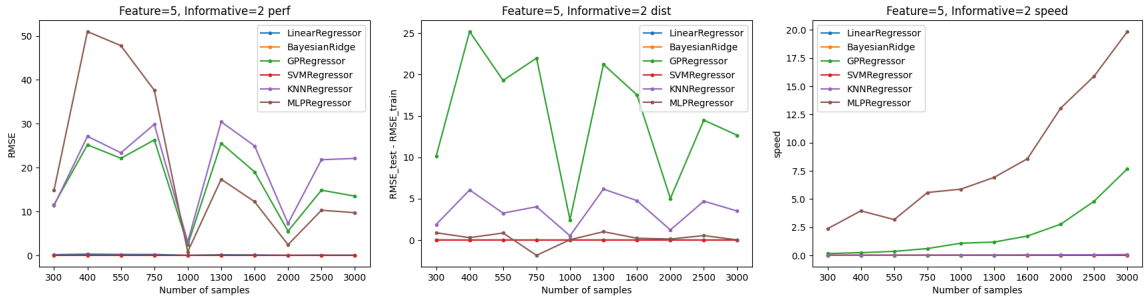
From now on, we will use the previously defined array, as it's a good compromise between required time and general representation of different sample numbers.

Experiment 3



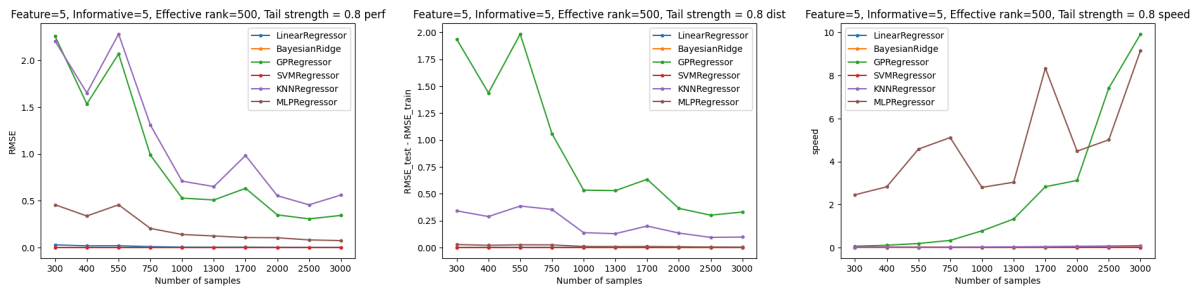
This is a dataset created without any additional parameter and will be the baseline for every successive analysis.

Experiment 4



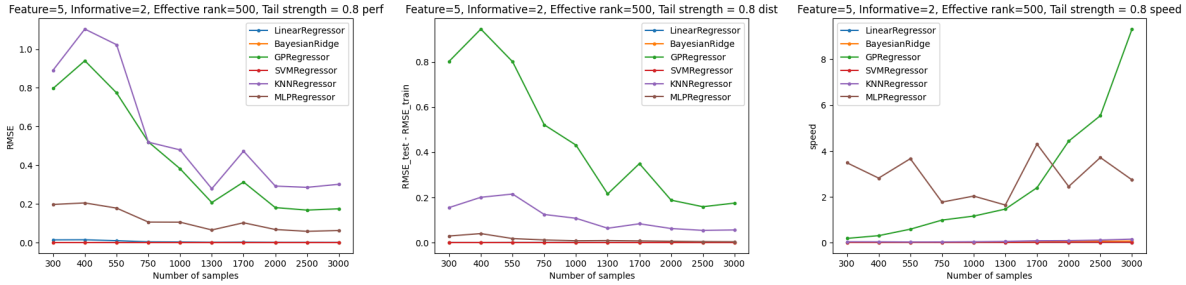
This is the first experiment with less informative features. Error values seem generally lower, but as the number of samples increases, they tend to the same values as the third experiment. The dataset generated for 1000 values scores performances much lower than the rest, it's the same issue seen in the example of the Experiment 2.

Experiment 5



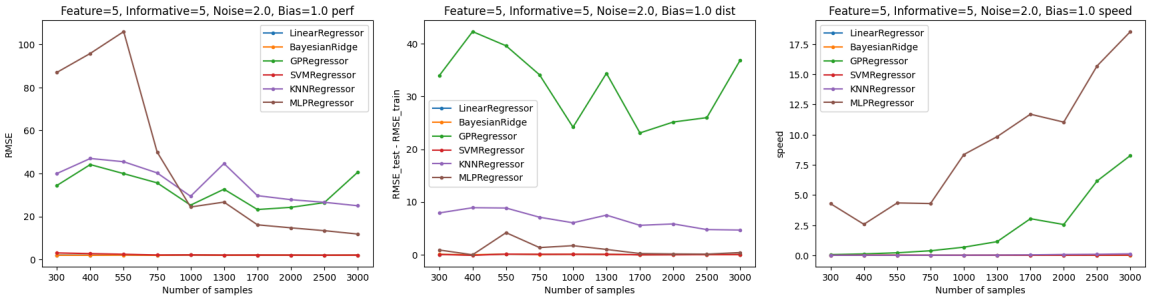
Here we introduce linear dependency across variables. Even if both parameters are high, the RMSE is significantly lower compared to the previous experiments. This is probably because the regressor might recognize the dependencies thus predicting with higher accuracy. In this case, the MLP regressor is not slower than the GR regressor.

Experiment 6



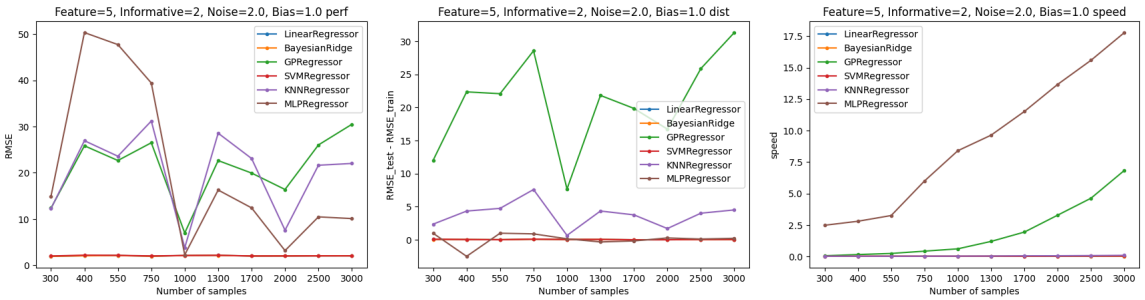
Reducing the informative features doesn't really make an impact in this case.

Experiment 7



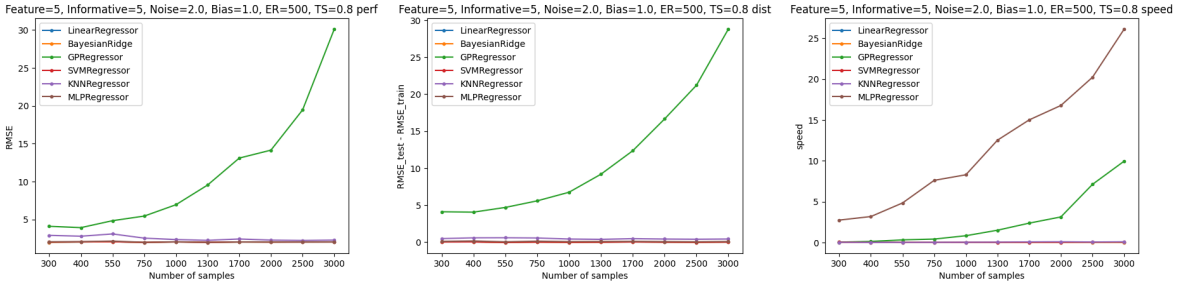
Adding both noise and bias creates similar graphs to the third experiment, except for one thing: the last values of the RMSE start to increase for the GP regressor. Initially it lowers, as always, but in the end it increases.

Experiment 8



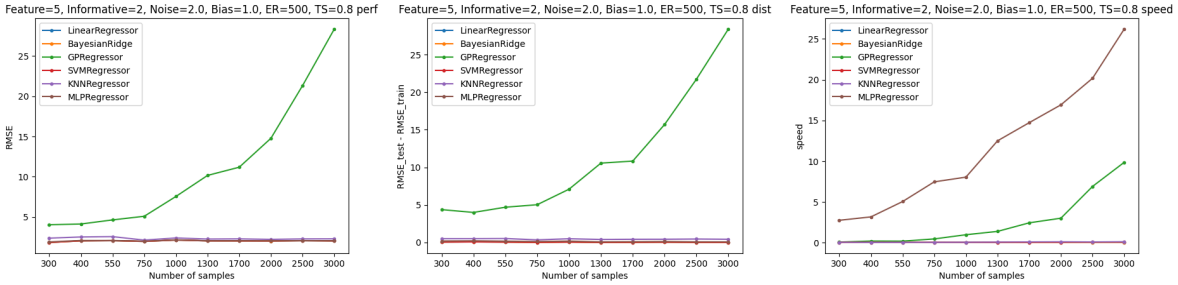
We can notice again the same patterns seen in Experiment 7 and 4, combined.

Experiment 9



Just like in experiment 7, the GP regressor increases with the number of samples. This time it's even worse than before: it starts higher and grows right from the beginning, contrary to the other regressors, which shows a pattern similar to before.

Experiment 10

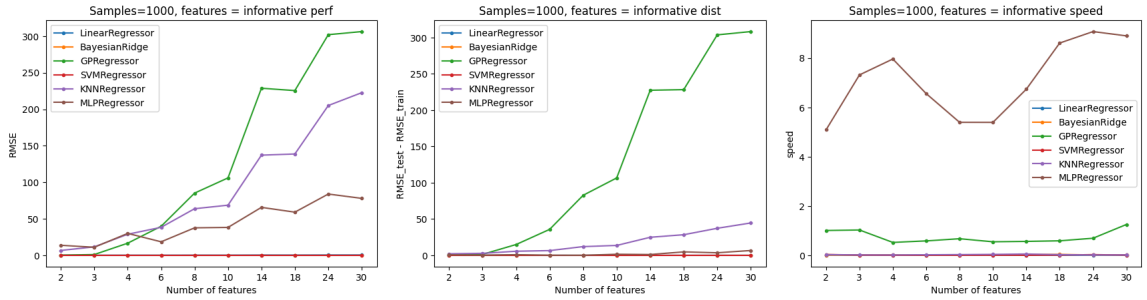


The results are analogous to those of Experiment 9. The only notable difference compared to previous experiments is that reducing the number of informative features didn't improve the RMSE of the GP regressor. This is probably because lowering the complexity makes the noise and the bias stand out even more than before.

In general, we can see that increasing the number of samples improves the training almost always. Between the 3 regressor that shows an important improvement, the MLP regressor is usually improving the most: NN usually requires more training data compared to ML. There are particular cases (Experiment 5 and 6) where the GP regressor takes is slower than the MLP, and other cases (Experiment 7, 9 and 10) where increasing the number of samples is actually worse for the GP regressor.

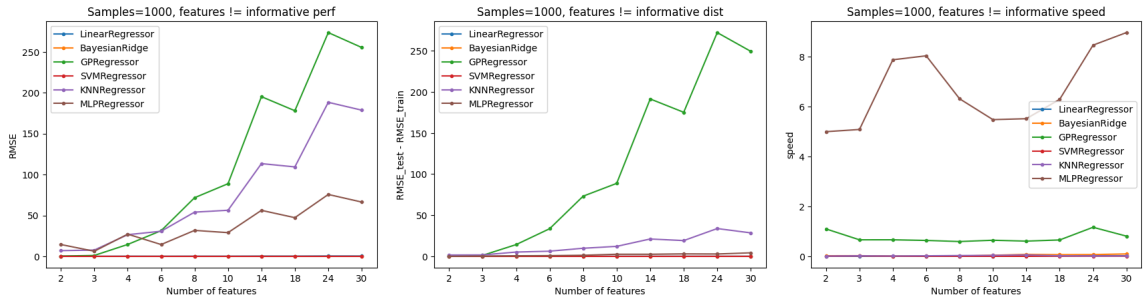
Results - fixed # of samples

Experiment 1



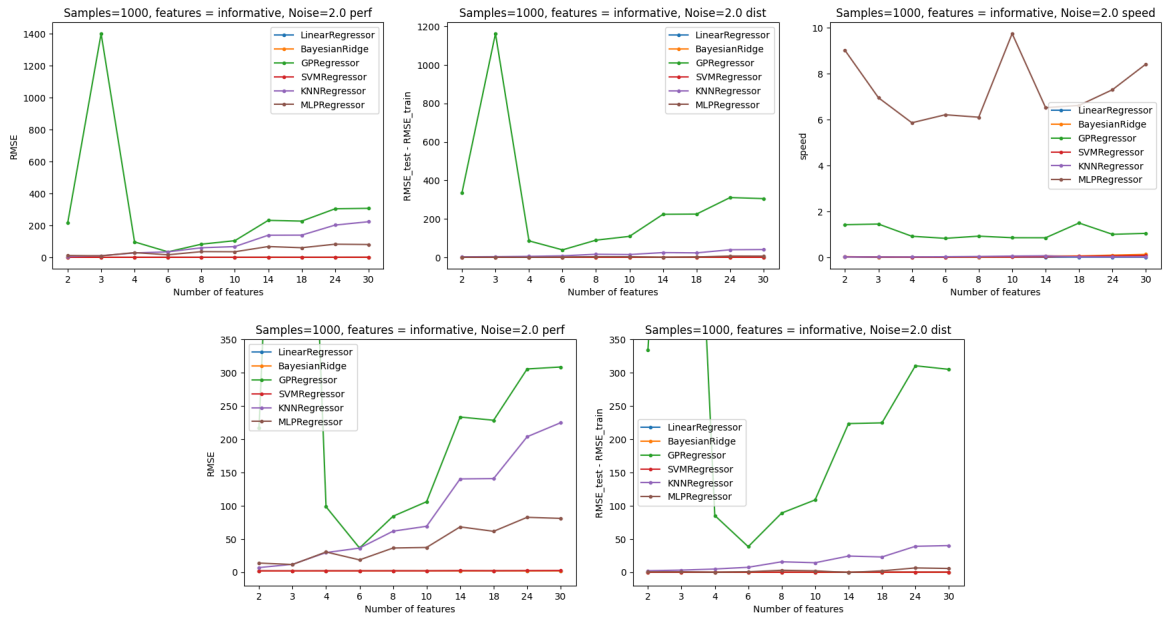
This is a dataset created without any additional parameter and will be the baseline for every successive analysis. Predictably, the RMSE increases a lot as the number of features increases. It's important to note that this time, the KNN regressor isn't the worst base performer.

Experiment 2



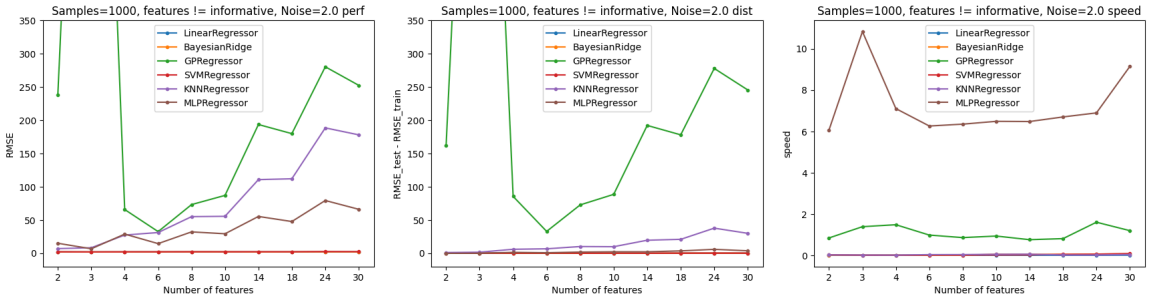
With less informative features the RMSE is slightly reduced. The difference is not as drastic as the difference in the first set of experiments.

Experiment 3



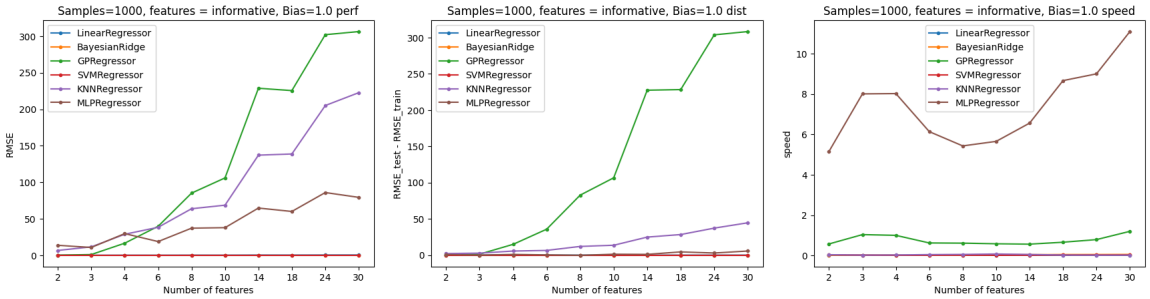
Datasets with certain parameters will have this spike RMSE and distance for the GP regressor. It's the same issue seen in the example of Experiment 2 (fixed number of features). Under the standard graphs are also reported two graphs with a vertical limit to better understand what is really happening. The noise has basically no impact on the results.

Experiment 4



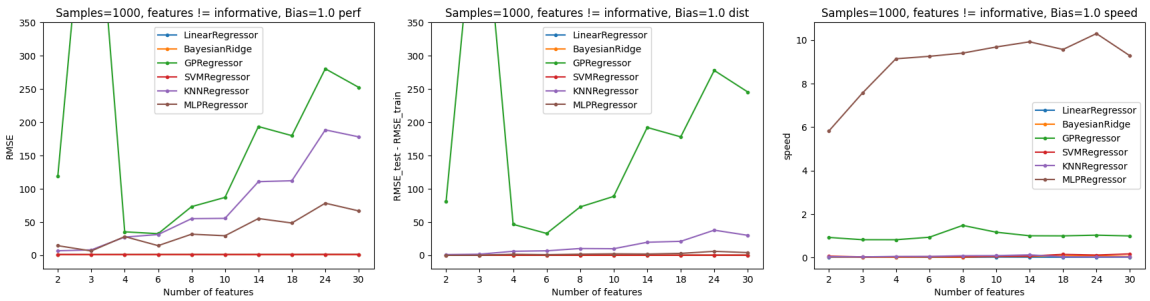
The results are analogous to those of Experiment 2.

Experiment 5



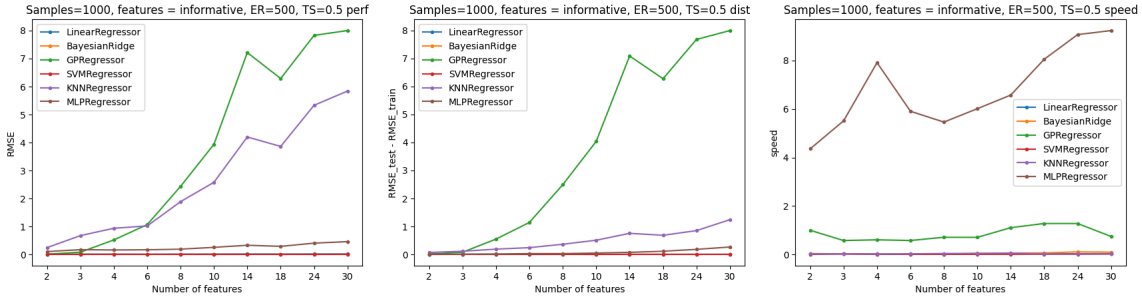
The results are analogous to those of Experiment 1: the bias doesn't seem to influence the results.

Experiment 6



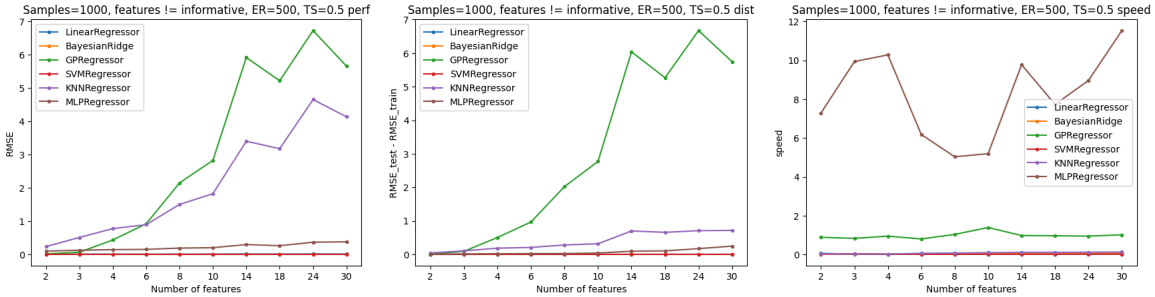
If we ignore the outlier results of the second dataset, the results are analogous to those of experiment 2.

Experiment 7



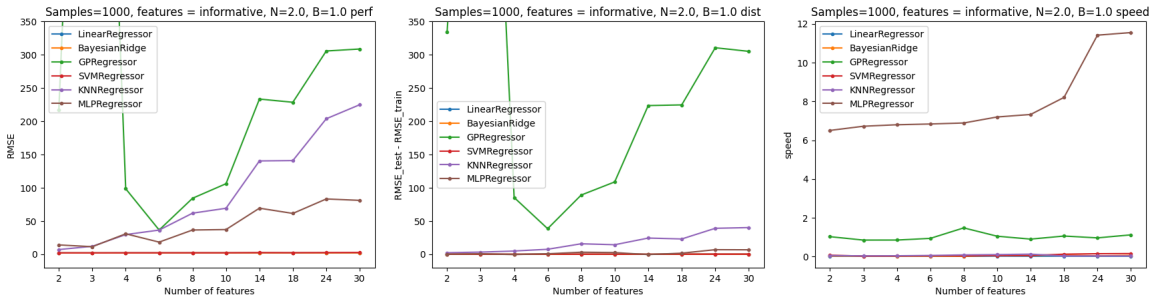
Introducing the effective rank lowers the RMSE just like it did in the first set of experiments, but the MLP regressor is reduced even more compared to the others.

Experiment 8



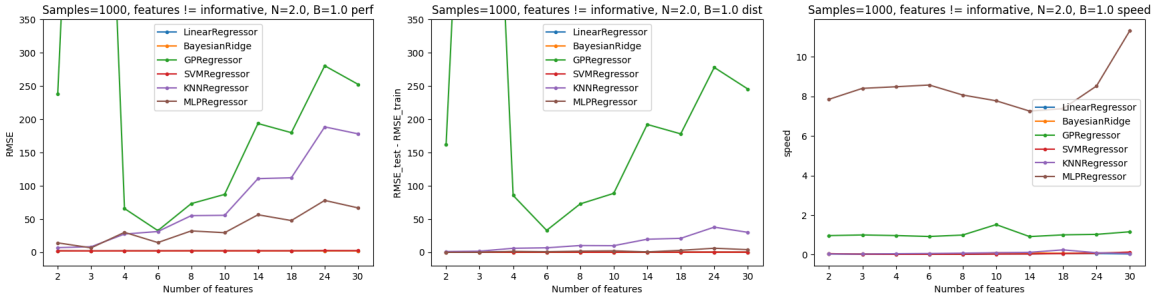
The results are analogous to those of Experiment 7.

Experiment 9



In this case, the RMSE of the GP regressor for the first 3 datasets is way higher than the rest, even if we don't consider the outlier. This might let us think that it performs worse with a small number of features, and it's true based on the first set of experiments but not to this extent.

Experiment 10



The results are analogous to those of Experiment 9, with slightly lower values as observed in Experiment 2.

In general, increasing the number of features increases the RMSE a lot. Using a bigger number of samples would reduce the error, as we've seen from the first set of experiments. We can notice that in this second set of experiments, even if the number of informative features lowers, there isn't a clear difference like in the first set of experiments. This is probably because errors in this case are much higher, reaching 300 RMSE, compared to 100 maximum in the other case.

Completeness

Setup Choices

In addressing completeness issues, our study incorporates considerations of both Missing Completely at Random (MCAR) and Missing Not at Random (MNAR) values. To simulate MCAR instances, we devised two distinct functions: one tailored to simulate missing values across the entire dataset and another focusing on simulating them within a single feature. These functions accept a probability parameter, determining the likelihood of values being missing. Adopting a parallel strategy, we simulated MNAR values, introducing a threshold as a parameter alongside a probability to simulate missing values above that threshold with the specified probability.

To facilitate our experiments, we crafted four arrays, each containing ten probabilities:

- ‘probs1’: ranging from 0.0 to 0.5 with a step of 0.05
- ‘probs2’: ranging from 0.5 to 1.0 with a step of 0.05
- ‘probs3’: ranging from 0.0 to 1.0 with a step of 0.1
- ‘probs4’: ranging from 0.0 to 0.3 with a step of approximately 0.0333

Additionally, we generated two arrays, each containing ten thresholds:

- ‘percentile_thresholds’: obtained by selecting values from 0 to 90 percentile of the entire dataset
- ‘random_thresholds’: obtained by randomly selecting values from the dataset

We conducted ten different experiments:

1. Simulation of MNAR values on the entire dataset, using ‘probs1’ as the probability array and ‘percentile thresholds’ as the threshold array. We employed a "mixed" approach by combining thresholds and probabilities. In this method, the first simulated dataset takes as parameters the value at the 0th percentile of the dataset as the threshold and 0.0 as the probability. Subsequently, the second simulated dataset takes the value at the 10th percentile of the dataset as the threshold and 0.05 as the probability, and so forth.
2. Simulation of MCAR values on the whole dataset, using ‘probs1’ as the probability array.
3. Simulation of MCAR values on the whole dataset, using ‘probs2’ as the probability array.
4. Simulation of MNAR values exclusively over the second feature, with a fixed threshold set to the median value. The simulation iterates over the probability array ‘probs1’.
5. For this specific experiment, a new dataset was generated with four features, of which only three are informative. Subsequently, MCAR values were simulated over the non-informative feature, iterating over the probabilities array ‘probs2’.
6. Revisiting the original dataset, MNAR values were simulated over the second

feature, using a fixed probability of 35% and a varying threshold from the 'percentile_thresholds'.

7. MNAR simulation over the third feature, employing a fixed probability of 85% and a varying threshold from the 'percentile_thresholds'.
8. MNAR simulation over the first feature, using a fixed probability of 35% and varying thresholds from the 'random_thresholds'.
9. MCAR simulation over the second feature, with a varying probability from 'probs1'.
10. MCAR simulation over the second feature, with a varying probability from 'probs3'.

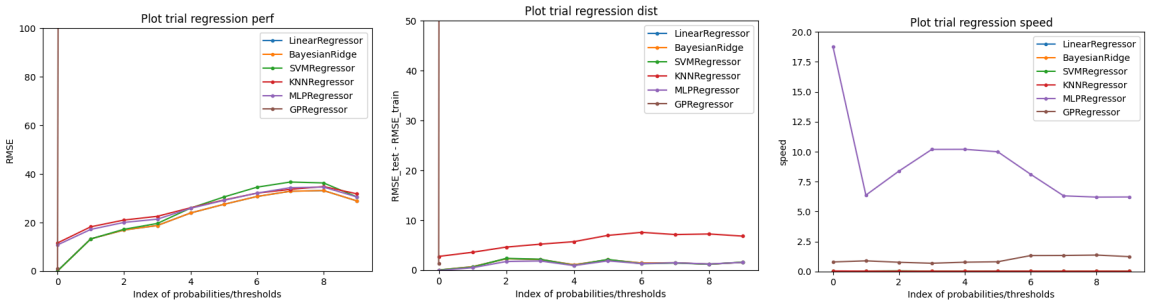
Pipeline Implementation

Our work involved applying diverse Data Imputation techniques to each experiment. Initially, we transformed all datasets from 'np.ndarray' to 'dataframes' using the Pandas library. Subsequently, we employed the following techniques across the ten experiments

1. Single Imputation by replacing missing values with a standard value (chosen as 0.00)
2. Single Imputation by propagating the last valid observation to the missing values using the function 'fillna(method="ffill")' from the Pandas library.
3. Single Imputation by replacing missing values with the mean value.
4. Machine Learning (ML) Imputation using KNNImputer with 5 neighbours from the Sklearn library.
5. Univariate Imputation using SimpleImputer from Sklearn library with strategy set to 'median', replacing missing values with the median value.
6. Machine Learning (ML) Imputation using functions from the Sklearn library and a function from this link.
7. Single Imputation by replacing missing values with the standard deviation value.
8. Single Imputation by propagating the last valid observation starting form the end to the missing values using the function 'fillna(method="bfill")' from the Pandas library.
9. Machine Learning (ML) Imputation using KNNImputer with 5 neighbours from the Sklearn library.
10. Machine Learning (ML) Imputation using functions from the Sklearn library and a function from this link.

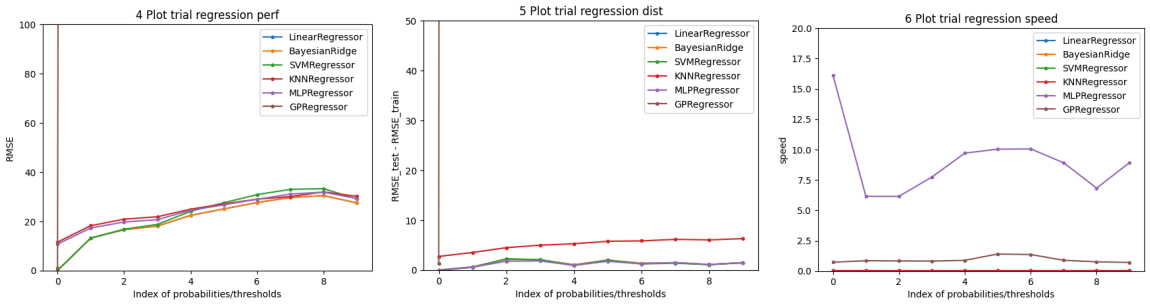
Results

Experiment 1



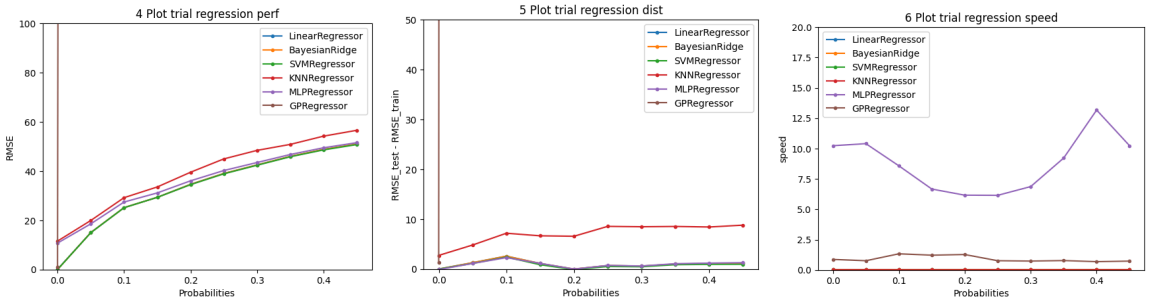
In the above figures we can observe the results of the analysis performed on the dataset of the first experiment. In terms of speed the algorithms are pretty fast, except for MLPRegressor, but this is not a surprising result.

To facilitate result interpretation, a y-axis plot limit was imposed due to the GPRegressor yielding extremely high values (exceeding 10000), which could make graph inspection very difficult. Regarding Root Mean Squared Error (RMSE), the observed trends are expected, with errors tending to increase alongside higher probabilities of missing values, despite higher thresholds.

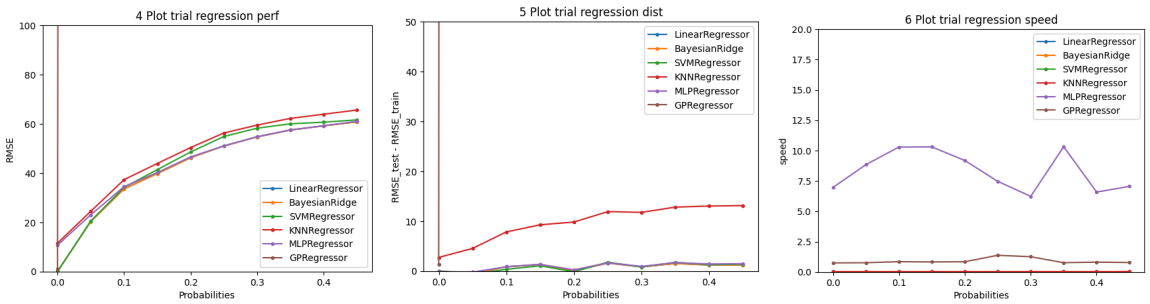


Post Data Imputation, the analysis of the dataset shows comparable results. Replacing missing values with 0 aligns with the default behavior of many machine learning algorithms when encountering missing data.

Experiment 2

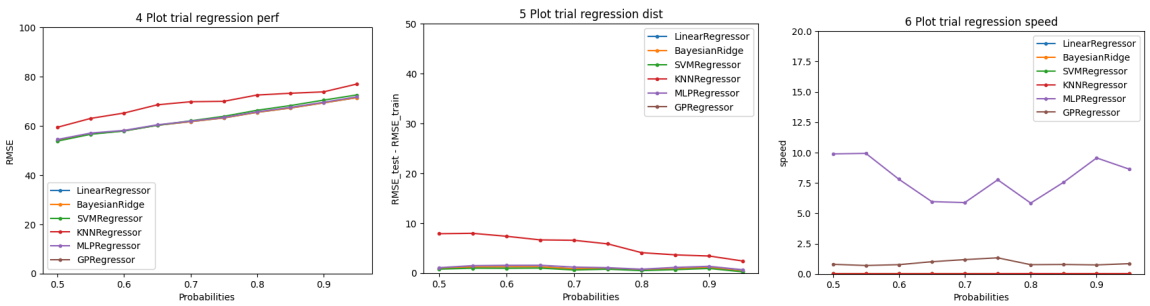


Results align with expectations, demonstrating an increase in RMSE with higher missing probabilities.

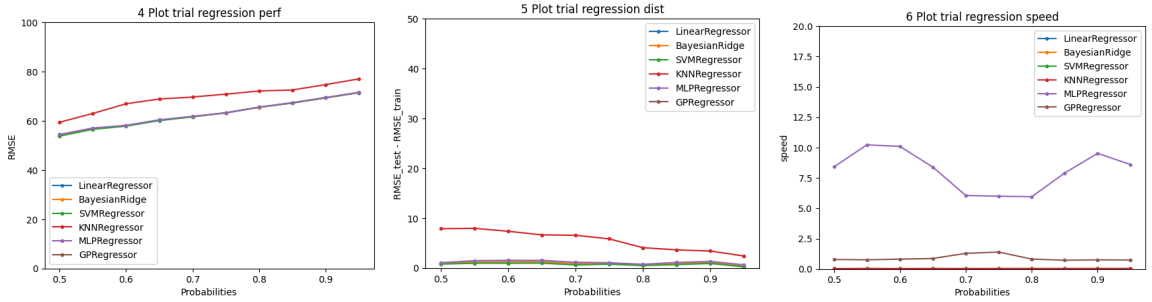


Following Data Imputation, we note a slight elevation in RMSE, particularly with more substantial missing probabilities. This can be attributed to the 'ffill' method, which propagates values from potentially diverse features and/or samples, altering the dataset significantly and complicating the learning process of the Regressors.

Experiment 3

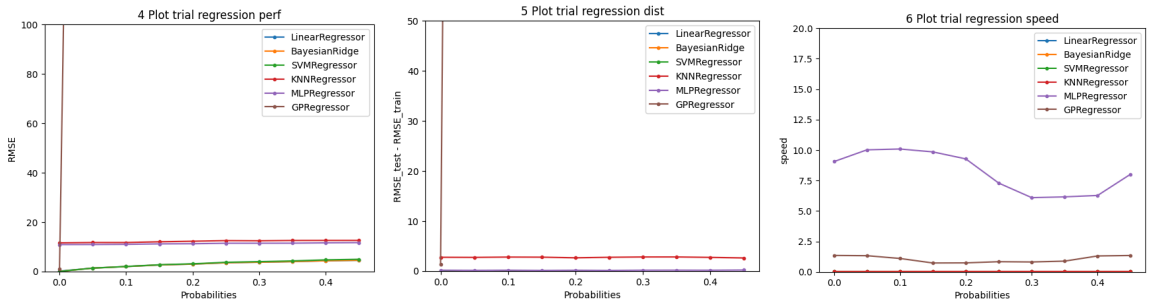


In this simulation, higher missing probabilities are considered, leading to an obvious increase in RMSE.

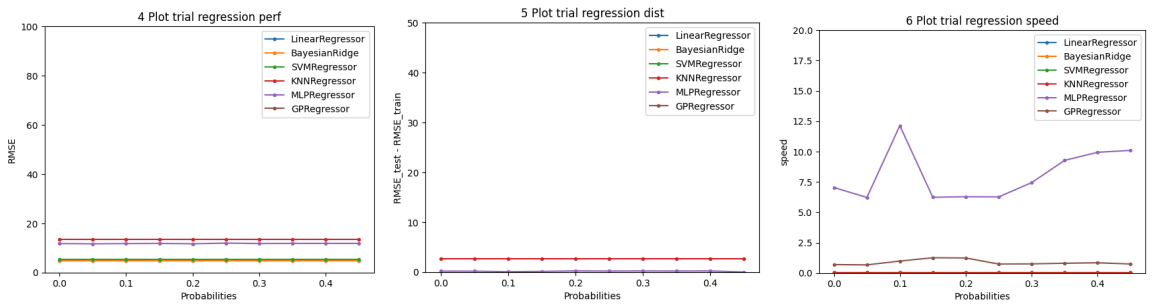


Post Data Imputation, results remain similar, as replacing all missing values with the mean of the entire dataset does not provide very meaningful information for the Regressors.

Experiment 4

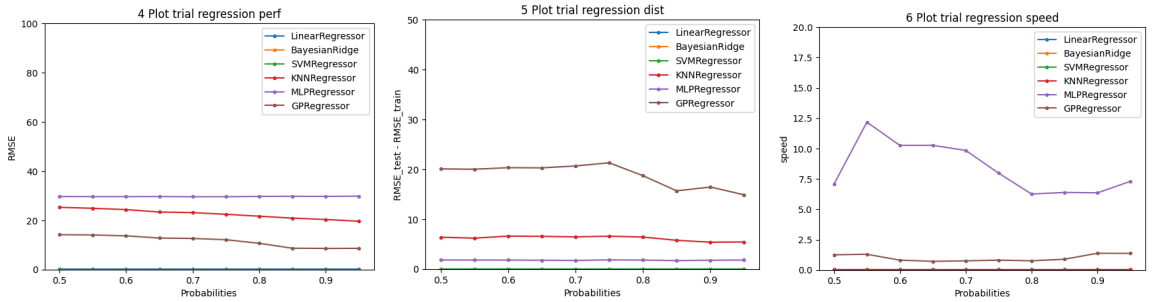


Lower RMSE is observed in this case, likely attributed to missing values existing solely within one feature. Algorithms demonstrate efficient learning, although the error slightly increases with higher missing probabilities. KNNRegressor and MLPRegressor exhibit comparatively worse performance, with the former being more affected by the absence of values due to its reliance on neighboring values.
The low performance of MLPRegressor is probably due to the small dimension of the dataset.

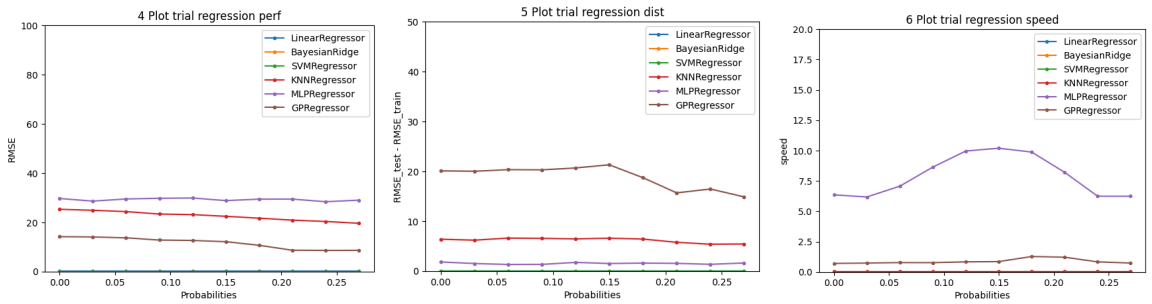


After Data Imputation, RMSE shows a slight increase for smaller probabilities and a slight decrease for larger ones. Notably, overall RMSE improves for MLPRegressor.

Experiment 5

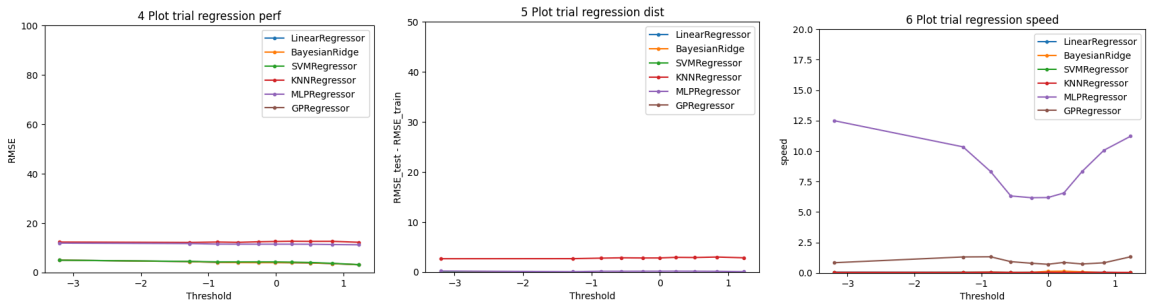


Here, RMSE tends toward 0 for certain algorithms, with improvement observed as missing probabilities increase. Since missing values are simulated only over the non-informative feature, LinearRegressor, SVMRegressor, and BayesianRidge exhibit no significant difference in RMSE, while other algorithms demonstrate reduced errors due to the absence of values in the non-informative feature, facilitating more efficient learning for algorithms like KNNRegressor.

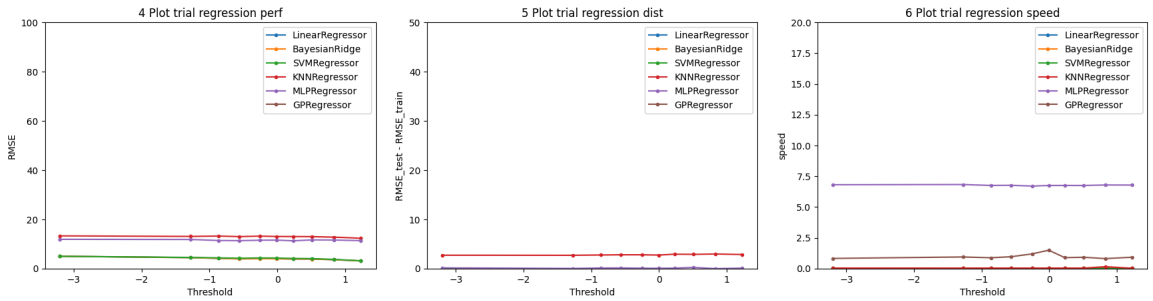


Post Data Imputation, no substantial differences are observed in the results.

Experiment 6

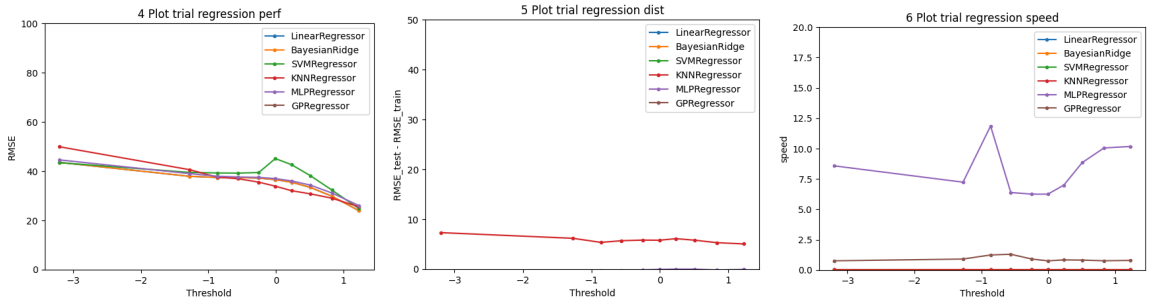


In this case, we considered a fixed probability and a varying threshold. It is not surprising to observe that with a higher threshold, the RMSE slightly decreases until almost reaching 0.

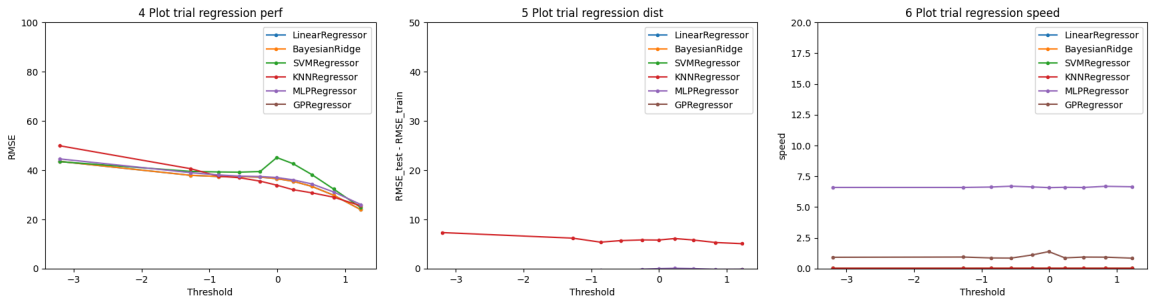


Post Data Imputation, no substantial differences are observed in the results.

Experiment 7

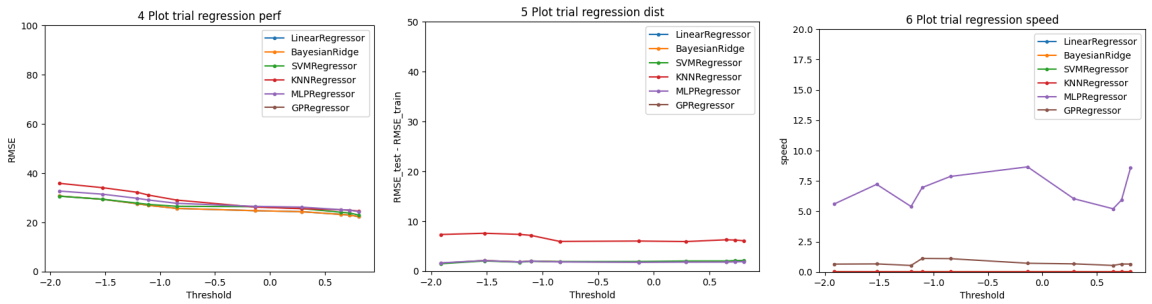


In this instance, we considered a varying threshold with a fixed probability. Given that the missing probability is substantially higher than before (85%), it is not surprising to observe a higher RMSE compared to the previous experiment.

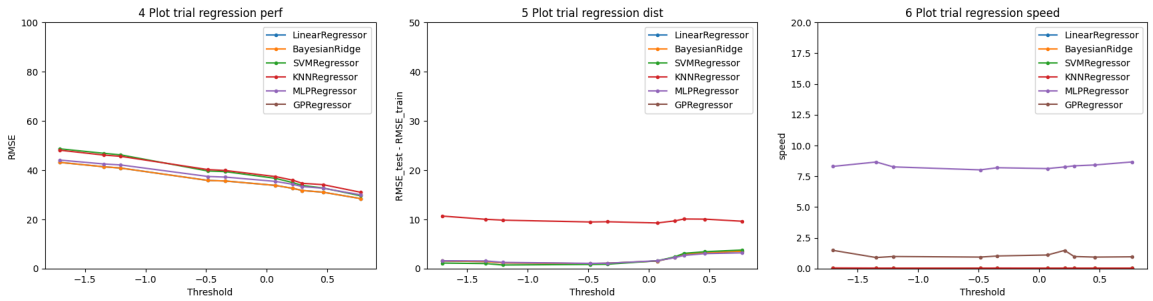


Post Data Imputation, no substantial differences are observed in the results.

Experiment 8

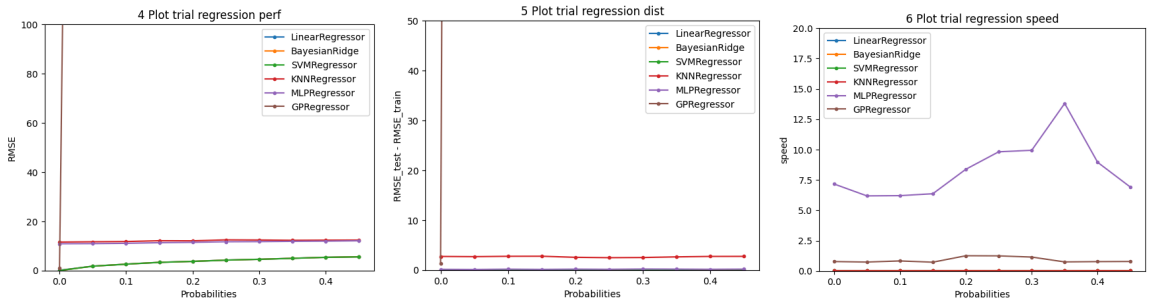


We sorted the ‘random_thresholds’ array to make the plot more readable. The results are analogous to the previous two experiments.

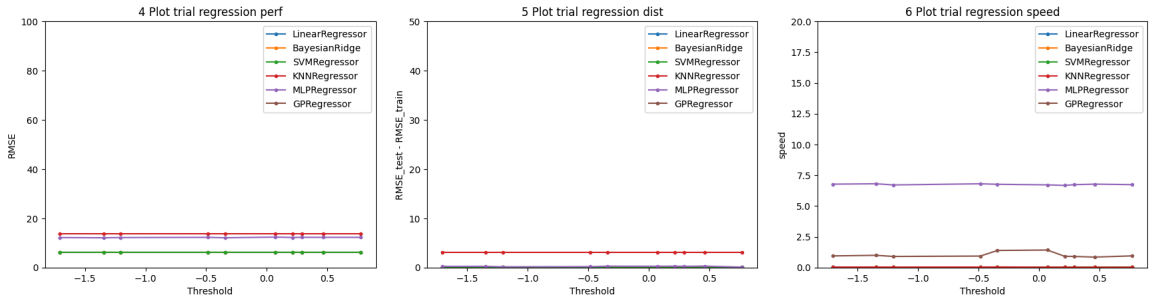


In this case, the Data Imputation increased the RMSE. The explanation to this behavior is probably similar to the one given to Experiment 2.

Experiment 9

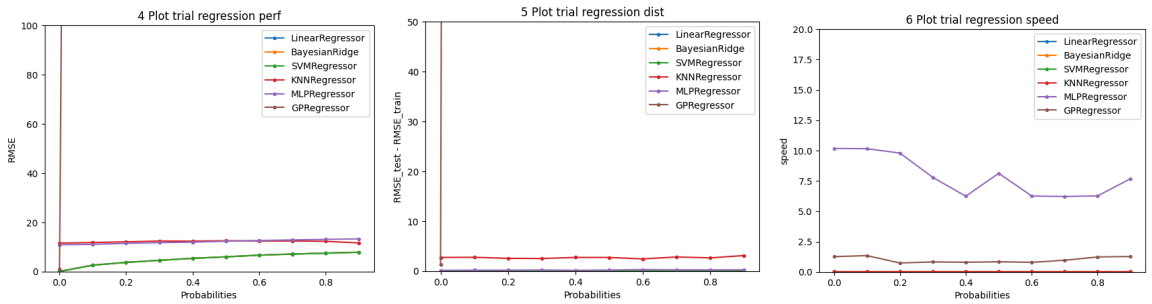


The results are analogous to those of Experiment 4.

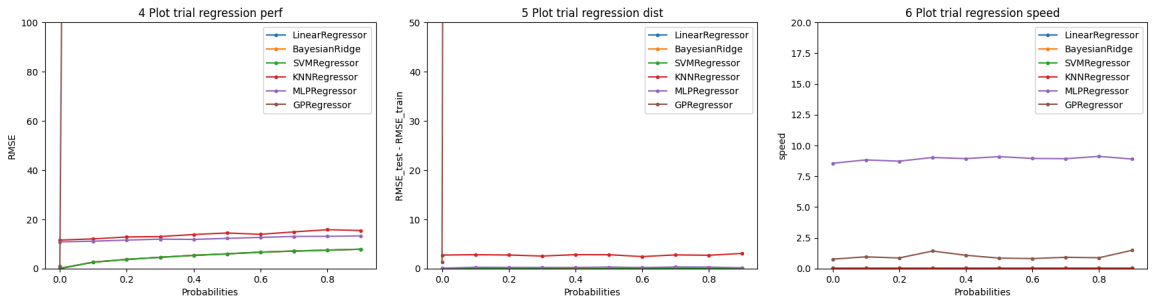


In this case, the Data Imputation increased the RMSE. The behavior is similar to the one of Experiment 4.

Experiment 10



The results are analogous to those of Experiment 4.



Here, the KNN regressor seems to perform worse, but everything else is exactly the same.