POLITECNICO
MILANO 1863

Data and Information Quality Report

Francesco Zanella

Person Code: 10835648

Project ID: 41

ASSIGNED DQ ISSUE: Accuracy

Ml task: Regression

Contents

List of Figures

List of code

Chapter 1

Introduction

The following report delves into the impact of enhancing dataset accuracy during the preprocessing phase on the overall performance of a regression model. The primary focus lies in the process of identifying and removing *outliers*, an approach employed to augment the model's accuracy.

Regarding the dataset for analysis, the decision has been made to utilize a synthetically generated dataset that closely mimics a real-world dataset.

1.1 Motivations

The decision to use a synthetically generated dataset closely resembling a real-world dataset is motivated by the desire to create a controlled environment for analysis. This approach allows for the systematic manipulation of data characteristics, ensuring a comprehensive exploration of the model's performance under various conditions.

In evaluating and improving the accuracy of a dataset, various techniques beyond outliers removal, including syntactic and semantic methods, can be considered. However, due to the synthetic nature of the dataset, practical constraints limited the feasibility of employing these additional techniques. While outlier removal addresses one aspect of data refinement, the decision to focus on this particular method within the context of a synthetic dataset is driven by practical considerations, acknowledging the challenges posed by the inherent characteristics of the data under investigation.

Chapter 2

Setup choices

2.1 Data collection

The first phase of the project involves data collection, and as mentioned earlier, the dataset for analysis has been synthetically generated for the reasons stated above. Specifically, I choose to use the **make_regression** function from scikit-learn.

```
make_regression(
    n_samples=1000,
    n_features=3,
    n_informative=3,
    n_targets=1,
    bias=10.0,
    effective_rank=None,
    tail_strength=0.5,
    noise=10.0,
    seed=2023
)
```

An explanation of the most significant parameters and the rationale behind their selection:

- **n_samples**: A dataset with 1000 samples might be deemed sufficiently representative to address the complexity of the problem. This size can capture variations in the data and provide a reliable basis for analysis.
- **n_features**: I kept 3 features to enhance visualization, manage model complexity, and reduce computational requirements, ensuring a focused analysis on key variables

for interpretation and understanding of relationships.

- **bias**: Introducing a bias term at 10 can simulate scenarios where there might be measurement errors or missing information regarding relevant variables not explicitly incorporated in the model. This contributes to creating a more realistically complex dataset by accounting for unmodeled factors or uncertainties in the relationship between features and the target variable.

- **noise**: Setting noise to 10 introduces realistic variability based on empirical observations, enhancing the dataset's realism. This choice reflects uncertainties and measurement errors, contributing to a more authentic representation and improving the practicality of the regression analysis.

2.2 Experiments

I opted to conduct 10 distinct experiments on the synthetically generated dataset, subsequently contaminated using the **data_pollution** function. In each experiment, I've introduced outliers at varying distances within the dataset. Furthermore, within each experiment, I varied the percentage of introduced outliers from 5 % to 50 % to assess their impact systematically.

Code 2.1: Experiment initialization

```
for i in range(n_experiments):
        print("Experiment {}".format(i))



        # A: DATA COLLECTION
        X, y = make_dataset_for_regression(n_samples=1000,
           n_features=n_features, n_informative=3, n_targets=1,
           bias=10.0, effective_rank=None, tail_strength=0.5, noise
           =10.0, seed=2023)
```

```
        # B: DATA POLLUTION
        # for each experiment the outlier distance grows
        # from the X we build ten different versions of the dataset
            with a different percentage of outliers
        X_polluted_list = []
        X_outliers_list = []
        for j in range(5,51,5):
            a,b = data_pollution(X=X.copy(),n_features=n_features,
                outlier_percentage=j/100,distribution="random",
                low_bound = -10 - i * 10, high_bound = 10 + i*10)
            X_polluted_list.append(a)
            X_outliers_list.append(b)
```

Chapter 3

Pipeline Implementation

To implement the pipeline for each of the 10 experiments, i followed these steps:

- **Data collection**: Using the make_regression function as explained in Chapter 2. I have created a dataset for each experiment on which outliers are introduced with different distances. Inside each experiment outliers distance is the same but the percentage of outliers vary from 5 % to 50 %.
- **Data pollution**: Through a dataset pollution function for each experiment outliers are randomly introduced at varying distances and with different percentages.
- **Data Analysis on Polluted Datasets**: For each experiment and for each percentage of introduced outliers (5 % to 50 %), I performed regression algorithms, subsequently comparing the results with those obtained after the cleaning phase.
- **Data Preparation**: In this phase, outliers were detected and removed using the DBSCAN technique.
- **Data Analysis on Cleaned Dataset**: After having applied Data Preparation to the datasets I reapply regression to compare the results with those obtained using the polluted datasets.

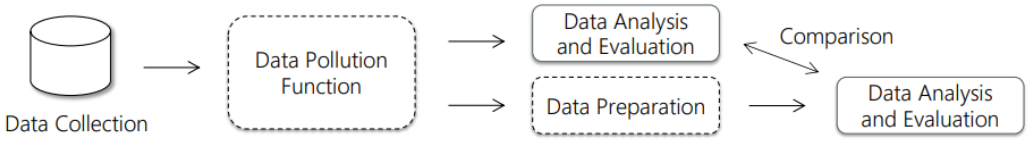


Figure 3.1: Pipeline implementation

3.1 Data Pollution

The data_pollution function is used to contaminate the dataset by adding outliers to it. In particular, inside the main I've called this function in the following way:

Code 3.1: main.py

```
# B: DATA POLLUTION
    # from the X we build ten different versions of the dataset
       with a different percentage of outliers
    X_polluted_list = []
    X_outliers_list = []
    # for each experiment I introduce different percentages of
       outliers
    for j in range(5,51,5):
        a,b = data_pollution(X=X.copy(),n_features=n_features,
           outlier_percentage=j/100,distribution="random",mean =
           -10 - i * 10, std = 10 + i*10)
        X_polluted_list.append(a)
        X_outliers_list.append(b)
```

For each experiment the outlier distance grows 10 times * i (experiment number).

Inside the **data_pollution** function based on the passed parameter **outlier_percentage**, some points are randomly chosen to be transformed into outliers by applying a shift. This shift can be random, normal, or multinomial, depending on the requirement. In my case and in the experiments presented in the report, a random shift was used.

Code 3.2: data_pollution.py

```
    size = int(n_samples * outlier_percentage)

    # random pick #size indices of samples to make them outliers
    outlier_indices = np.random.choice(n_samples, size=size,
       replace=False)

    # how much i want to shift the outliers and which distribution
       to use ?
    if distribution == 'normal':
```

```
        outliers_shift = np.random.normal(loc=mean, scale=std, size
            =(size,n_features))
    elif distribution == 'random':
        outliers_shift = np.random.uniform(mean, std, size=(size,
            n_features))
    elif distribution == 'multinomial':
        outliers_shift = np.random.multinomial(size=(size,
            n_features))
```

Then the shift is actually applied to the selected indexes and the new dataset polluted is returned:

Code 3.3: data_pollution.py

```
# create the polluted dataset
    for j,i in enumerate(outlier_indices):
        df_polluted.iloc[i] += outliers_shift[j]

    return df_polluted.values,outlier_indices)
```

3.2 Visualization of the outliers introduced

To gain a visual understanding of how data pollution works, I also implemented a function named **print_dataset_polluted** to visualize in 3D, for each experiment, the dataset with the addition of outliers randomly generated at different percentages. Outliers are depicted in red, while the remaining points are shown in blue. The more the experiment is high the more the distance between outliers and not outliers grows as we can see in the example down here.

Figure 3.2: Experiment 0 polluted datasets

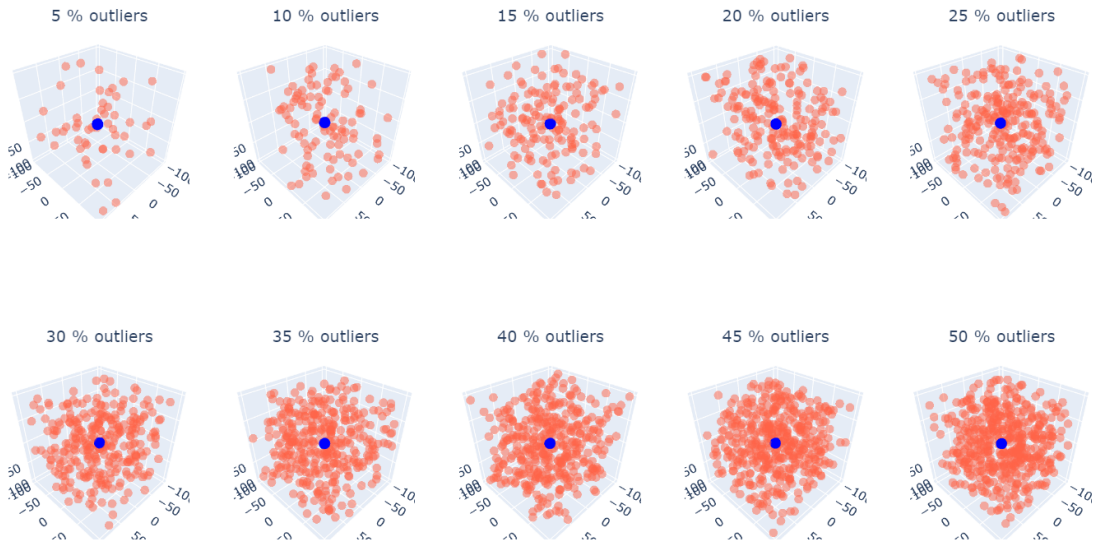


Figure 3.3: Experiment 9 polluted datasets

3.3 Data Analysis on Polluted Datasets

After introducing the outliers, we assess the performance of various regression algorithms on the "dirty" datasets using two different metrics: performance and distance between train and test. (speed is not considered since is not relevant for the analysis we are conducting). We evaluate these metrics across five different regression algorithms ("LinearRegressor," "BayesianRidge," "SVMRegressor," "KNNRegressor," and "MLPRegressor").

Code 3.4: main.py

```
# D: DATA ANALYSIS ON POLLUTED DATASETS
        results_for_each_algorithm_polluted= []
        for algorithm in REGRESSION_ALGORITHMS:
            results_for_each_percentage_polluted = []
            for d_polluted in X_polluted_list:
                result = regression(d_polluted, y, algorithm, SEED)
                results_for_each_percentage_polluted.append(result)

            results_for_each_algorithm_polluted.append(
               results_for_each_percentage_polluted)
```

Then we use some utils functions, **plot** and **print_table** to create a plot for each metric and a table with the numeric results for further analysis and comparison with the cleaned version.

Plot for experiment 0 for performance metric:

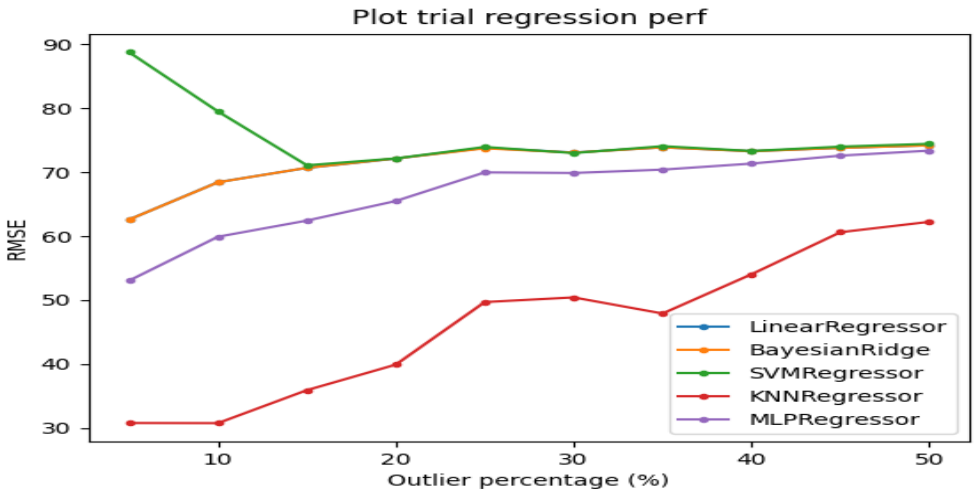


Figure 3.4: Experiment 0 performance evaluation

Table with numeric results for experiment 0 for performance metric:

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
0	63.163180	63.159939	89.422120	34.318283	53.280868
1	66.437975	66.445957	72.437892	34.496370	59.520468
2	70.558766	70.539484	71.080793	42.102650	64.008785
3	69.559184	69.577994	69.587994	38.754099	64.227037
4	71.776237	71.762242	71.906671	43.446698	67.820386
5	73.072861	73.032984	73.158191	51.298336	69.422826
6	72.139714	72.144959	72.379701	50.792194	69.523785
7	74.421829	74.389718	74.518605	52.065995	71.987511
8	73.647222	73.660022	73.925356	58.141087	72.154340
9	74.400791	74.408236	74.624049	60.930044	73.351257

3.4 Data Preparation

In this phase, I need to remove the previously introduced outliers. Several removal methods have been tried, but the most effective one appears to be DBSCAN, a clustering-based resolution method.

The reasons why DBSCAN performed better can be listed as follows:

- **Density-based Approach**: DBSCAN excels in identifying outliers based on their density within the dataset. It is particularly effective when outliers exhibit lower density compared to the main clusters, allowing it to separate them more efficiently. Since in our datasets the not-outliers have an higher density with respect to outliers, dbscan is the perfect choice.
- **Adaptability to Cluster Shapes**: since my data pollution function introduce outliers with no assumptions on the shape, dbscan is the best approach because is able to to adapt to clusters of various shapes ensures that outliers forming irregularly shaped clusters are more accurately identified and isolated.
- **Easily tunable**: DBSCAN's parameters, such as epsilon (neighborhood distance) and minimum points, can be adjusted to accommodate different characteristics of the data. This adaptability proves beneficial when dealing with varying densities and sizes of outliers as in our case when dealing with different percentages of outliers introduced (5 % to 50 %). In my case the perfect fit are: - epsilon = 0.9 - min_samples = 5

Below is the implementation of the outliers removal function.

Code 3.5: data_preparation.py

```
def data_preparation(X,y,n_features):

    """
    @param X: the ndarray to clean
    """
    df = pd.DataFrame(X, columns=[f'feature_{i}' for i in range(
        n_features)])
```

```
    df['target'] = y

    eps = 0.9
    min_samples = 5
    dbscan = DBSCAN(eps=eps, min_samples=min_samples)
    df['cluster'] = dbscan.fit_predict(df[['feature_0', 'feature_1'
        , 'feature_2']])

    clean_df = df[df['cluster'] != -1]

    outlier_indices = df[df['cluster'] == -1].index

    # feature_0,feature_1, feature_2, target, cluster
    return clean_df[['feature_0', 'feature_1', 'feature_2']].values
        , clean_df['target'].values,outlier_indices
```

3.5 Data Analysis on Cleaned Dataset

After having removed the outliers, we assess the performance of various regression algorithms on the "cleaned" datasets using two different metrics: performance and distance between train and test. (speed is not considered since is not relevant for the analysis we are con- ducting). We evaluate these metrics across five different regression algorithms ("Linear- Regressor," "BayesianRidge," "SVMRegressor," "KNNRegressor," and "MLPRegressor").

Code 3.6: main.py

```
results_for_each_algorithm_cleaned= []
        for algorithm in REGRESSION_ALGORITHMS:
            results_for_each_percentage_cleaned = []
            for h in range(len(X_cleaned_list)):
                result = regression(X_cleaned_list[h],
                   y_cleaned_list[h], algorithm, SEED)
                results_for_each_percentage_cleaned.append(result)

            results_for_each_algorithm_cleaned.append(
               results_for_each_percentage_cleaned)
```

As we did with the polluted datasets we use some utils functions, **plot** and **print_table** to create a plot for each metric and a table with the numeric results to compare the results the analysis before having cleaned the dataset.

Plot for experiment 0 for performance metric:

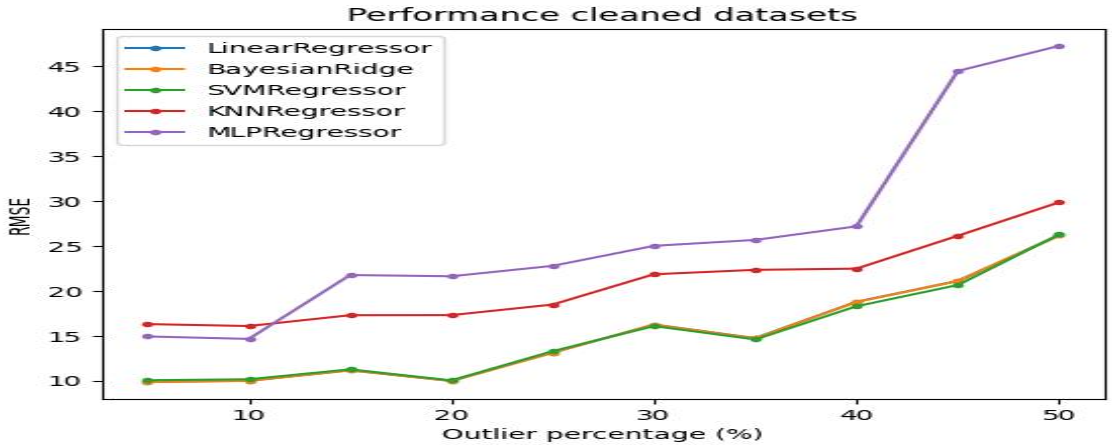


Figure 3.5: Experiment 0 performance evaluation

Table with numeric results for experiment 0 for performance metric:

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
0	10.000809	10.004900	10.061993	15.591680	14.282306
1	13.779366	13.785035	13.755910	18.363689	17.365236
2	11.156695	11.149361	11.198446	16.654966	20.006016
3	14.420850	14.416217	14.412040	20.076572	23.117938
4	10.101238	10.100419	10.229415	16.608829	19.610424
5	13.288874	13.282606	13.421034	20.085632	22.696792
6	16.936100	16.929205	16.784082	23.383860	25.214464
7	12.283842	12.269709	12.288667	19.080897	25.419249
8	17.696928	17.685258	17.461082	21.627933	42.025983
9	19.039555	19.021964	18.721794	23.225001	41.567862

Chapter 4

Results

In this final chapter, I will discuss the results obtained by comparing the outcomes of each experiment and providing reasons for the observed trends.

For each experiment, two metrics will be presented:

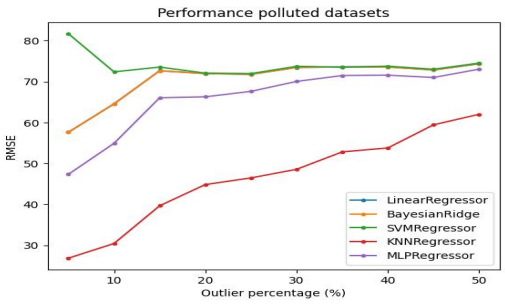
- Perfomance (RMSE)
- Distance train-test

After the pollution phase, using the dataset with outliers, and following the cleaning phase with the cleaned datasets. For each experiment, the effectiveness of outlier removal will then be demonstrated by showing the percentage of outliers correctly identified and removed.

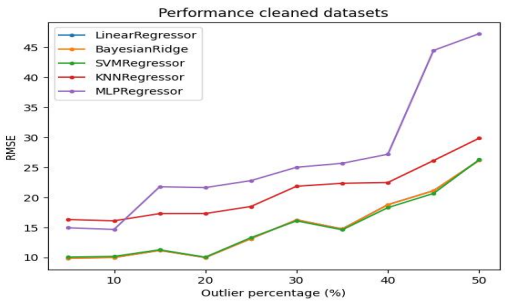
4.1 Experiment N°0

The outliers for this experiment were generated with a uniform shift between *-10 - i10* and *10 + i10*, where $i = 0$. For $i = 0$, they were generated with a uniform shift between -10 and 10.

4.1.1 RMSE - polluted vs cleaned datasets



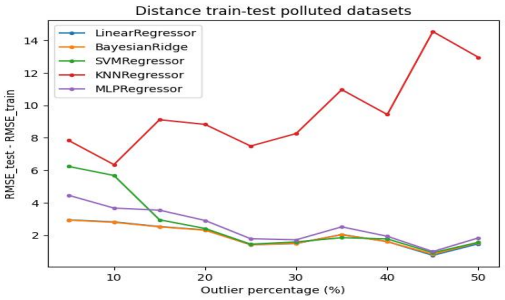
(a) Rmse polluted datasets.



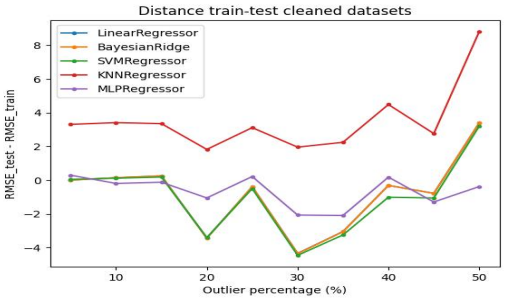
(b) Rmse cleaned datasets.

Figure 4.1: RMSE comparison.

4.1.2 train/test distance - polluted vs cleaned datasets



(a) train/test distance polluted datasets.



(b) train/test distance cleaned datasets.

Figure 4.2: train/test distance comparison.

4.1.3 Outliers removal precision

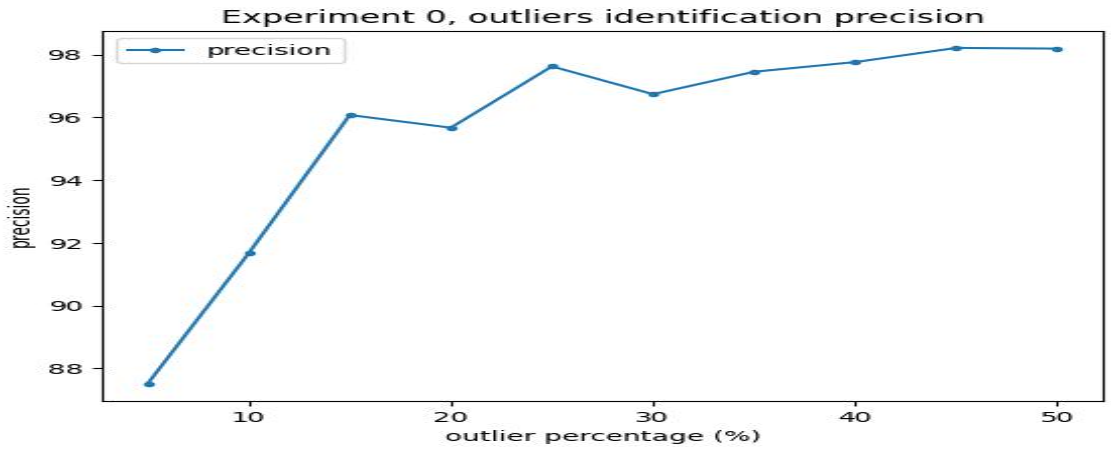
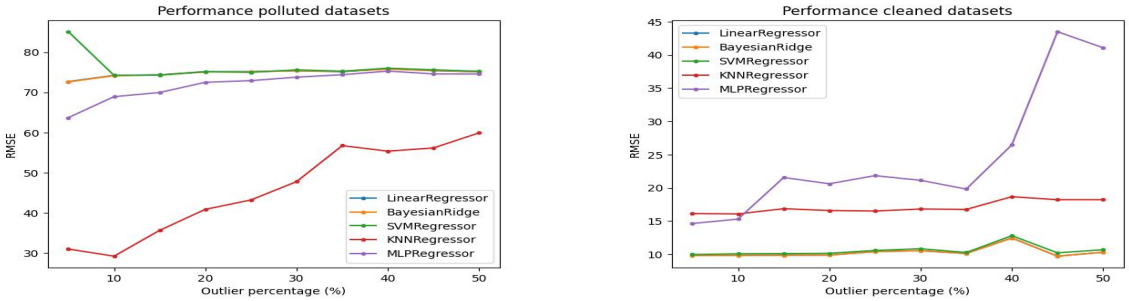


Figure 4.3: Outliers removal precision

4.2 Experiment N°1

The outliers for this experiment were generated with a uniform shift between *-10 - i10* and *10 + i10*, where $i = 1$. For $i = 1$, they were generated with a uniform shift between -20 and 20.

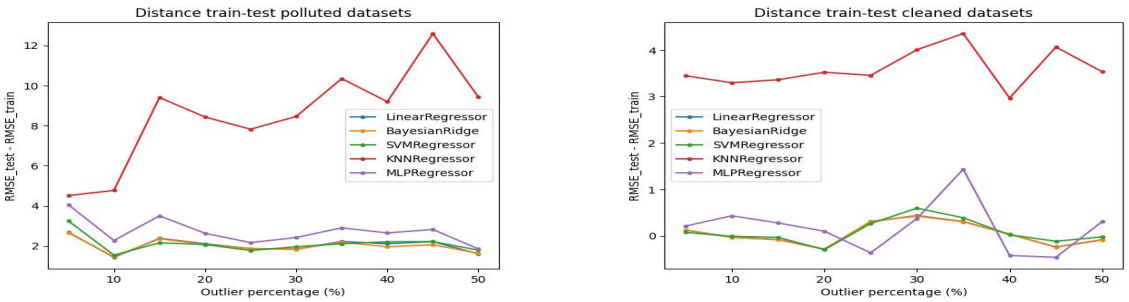
4.2.1 RMSE - polluted vs cleaned datasets



(a) Rmse polluted datasets. *(b) Rmse cleaned datasets.*

Figure 4.4: RMSE comparison.

4.2.2 train/test distance - polluted vs cleaned datasets



(a) train/test distance polluted datasets. *(b) train/test distance cleaned datasets.*

Figure 4.5: train/test distance comparison.

4.2.3 Outliers removal precision

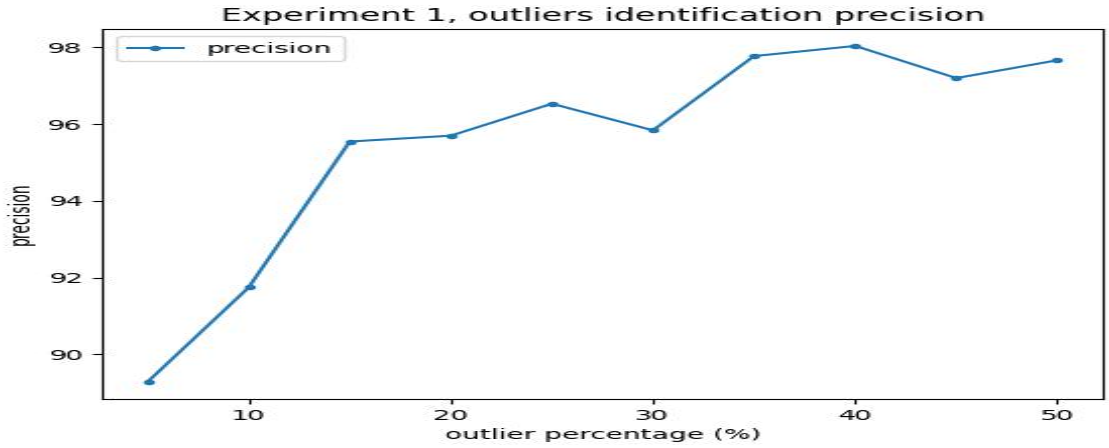
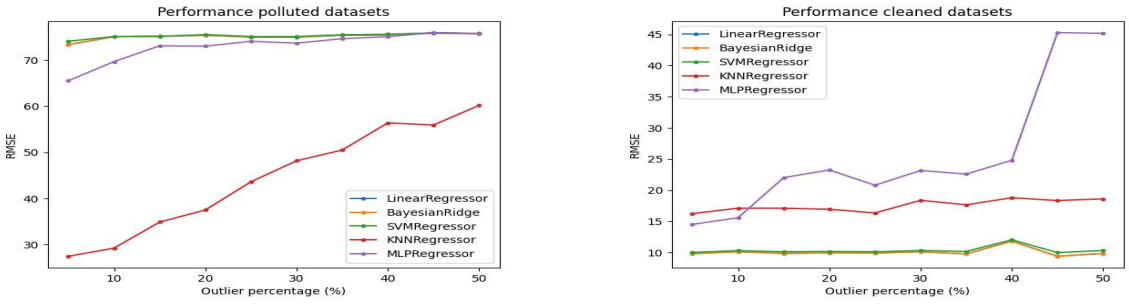


Figure 4.6: Outliers removal precision

4.3 Experiment N°2

The outliers for this experiment were generated with a uniform shift between *-10 - i10* and *10 + i10*, where $i = 2$. For $i = 2$, they were generated with a uniform shift between -30 and 30.

4.3.1 RMSE - polluted vs cleaned datasets

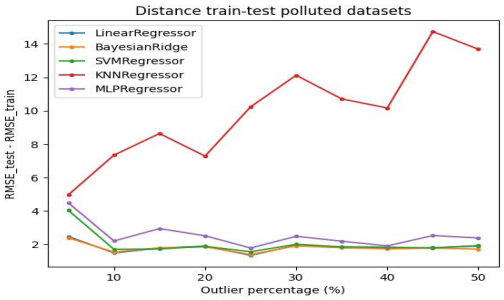


(a) Rmse polluted datasets.

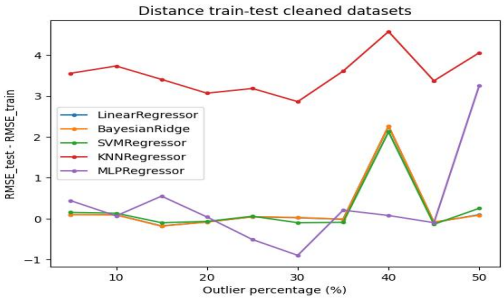
(b) Rmse cleaned datasets.

Figure 4.7: RMSE comparison.

4.3.2 train/test distance - polluted vs cleaned datasets



(a) train/test distance polluted datasets.



(b) train/test distance cleaned datasets.

Figure 4.8: train/test distance comparison.

4.3.3 Outliers removal precision

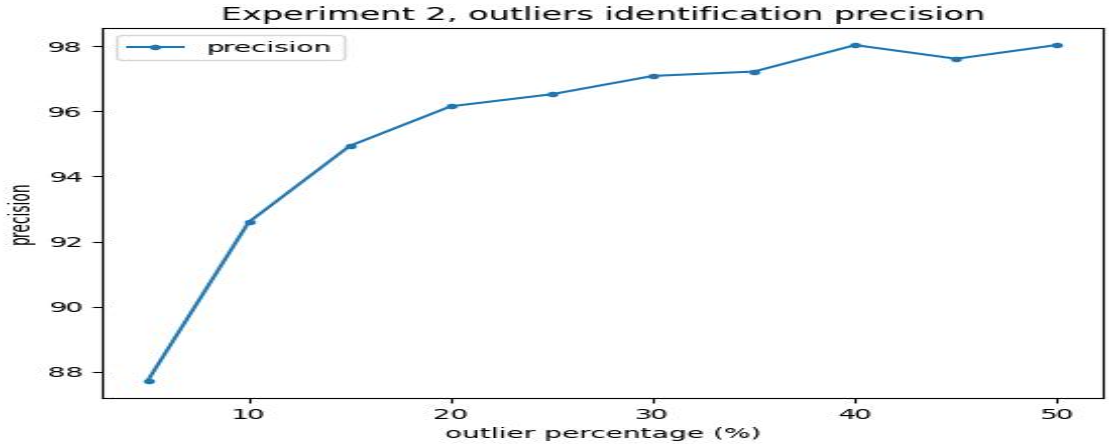
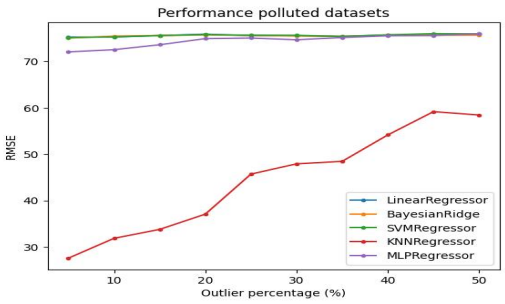


Figure 4.9: Outliers removal precision

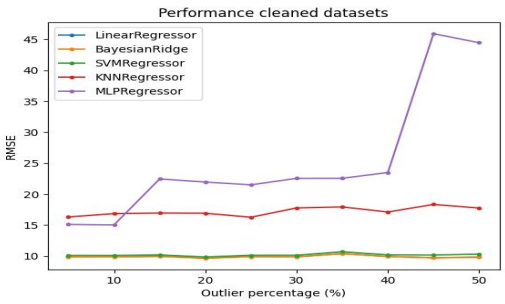
4.4 Experiment N°3

The outliers for this experiment were generated with a uniform shift between *-10 - i10* and *10 + i10*, where $i = 3$. For $i = 3$, they were generated with a uniform shift between -40 and 40.

4.4.1 RMSE - polluted vs cleaned datasets



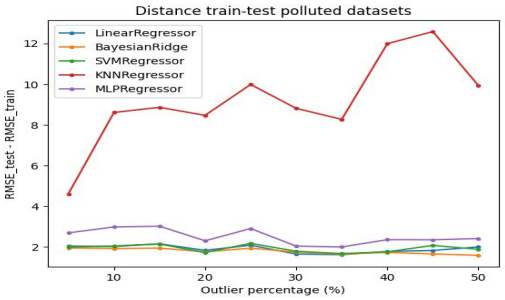
(a) Rmse polluted datasets.



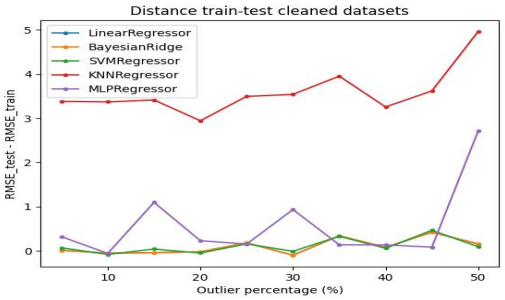
(b) Rmse cleaned datasets.

Figure 4.10: RMSE comparison.

4.4.2 train/test distance - polluted vs cleaned datasets



(a) train/test distance polluted datasets.



(b) train/test distance cleaned datasets.

Figure 4.11: train/test distance comparison.

4.4.3 Outliers removal precision

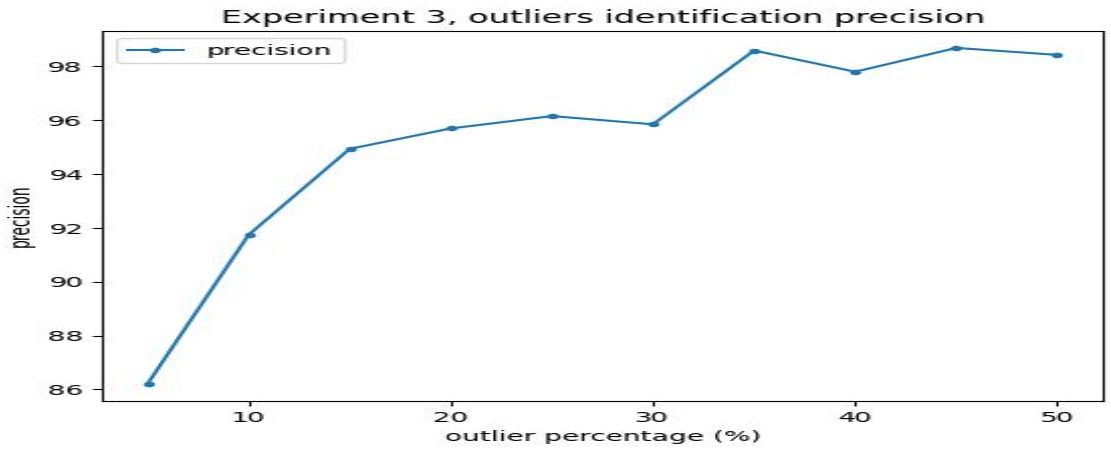
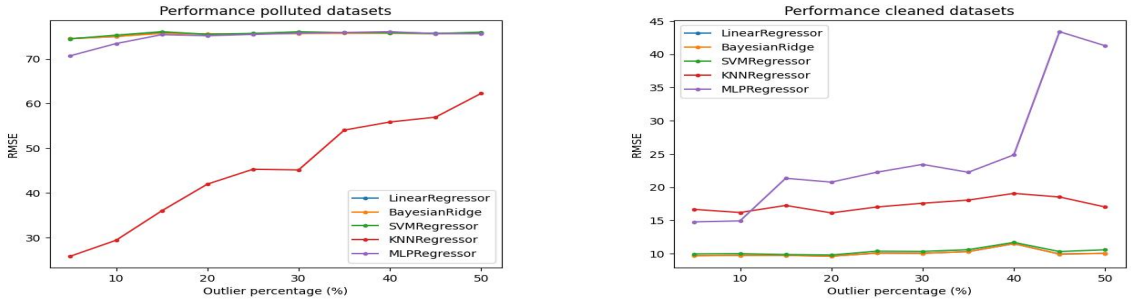


Figure 4.12: Outliers removal precision

4.5 Experiment N°4

The outliers for this experiment were generated with a uniform shift between *-10 - i10* and *10 + i10*, where $i = 4$. For $i = 4$, they were generated with a uniform shift between -50 and 50.

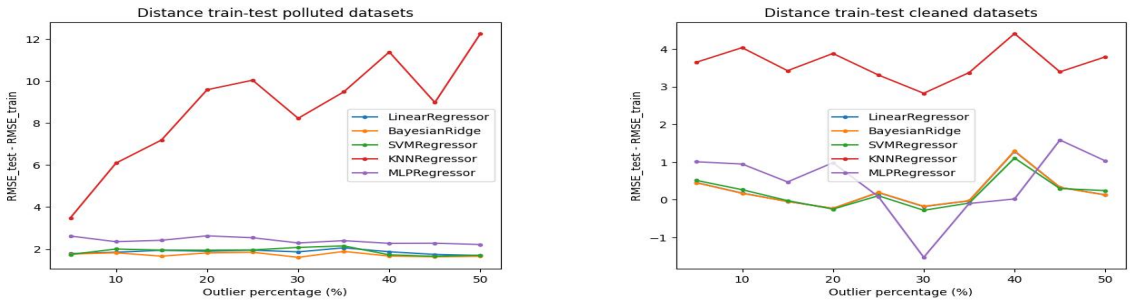
4.5.1 RMSE - polluted vs cleaned datasets



(a) Rmse polluted datasets. *(b) Rmse cleaned datasets.*

Figure 4.13: RMSE comparison.

4.5.2 train/test distance - polluted vs cleaned datasets



(a) train/test distance polluted datasets. *(b) train/test distance cleaned datasets.*

Figure 4.14: train/test distance comparison.

4.5.3 Outliers removal precision

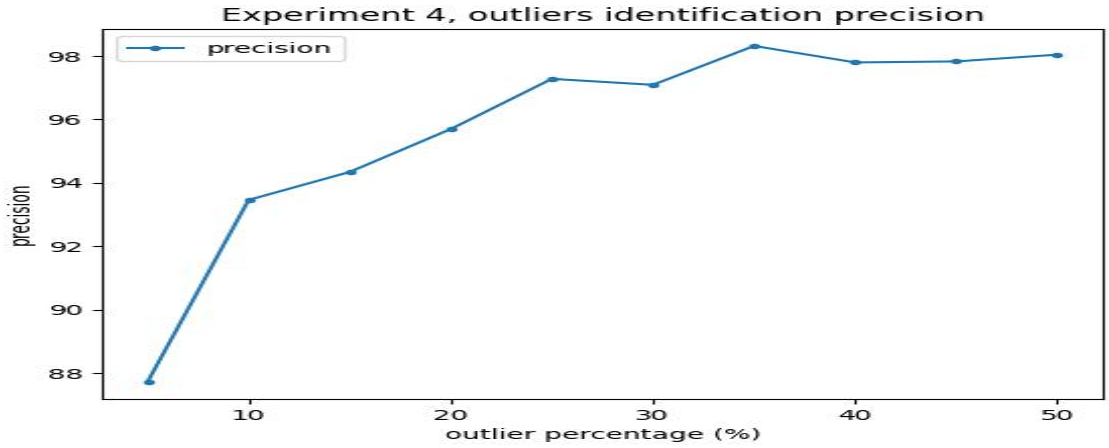
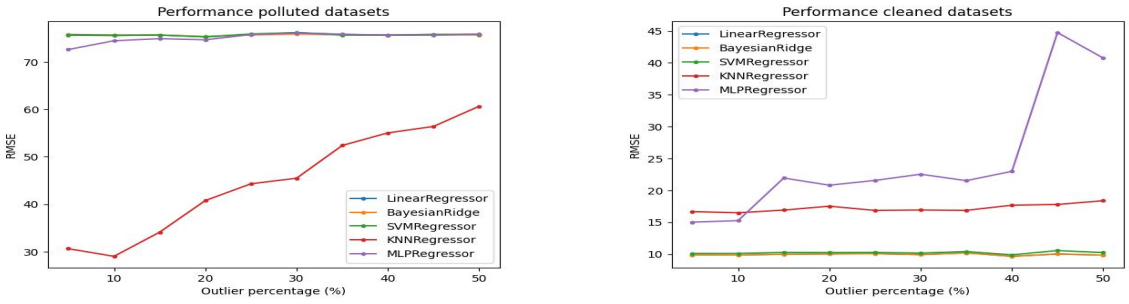


Figure 4.15: Outliers removal precision

4.6 Experiment N°5

The outliers for this experiment were generated with a uniform shift between *-10 - i10* and *10 + i10*, where $i = 5$. For $i = 5$, they were generated with a uniform shift between -60 and 60.

4.6.1 RMSE - polluted vs cleaned datasets

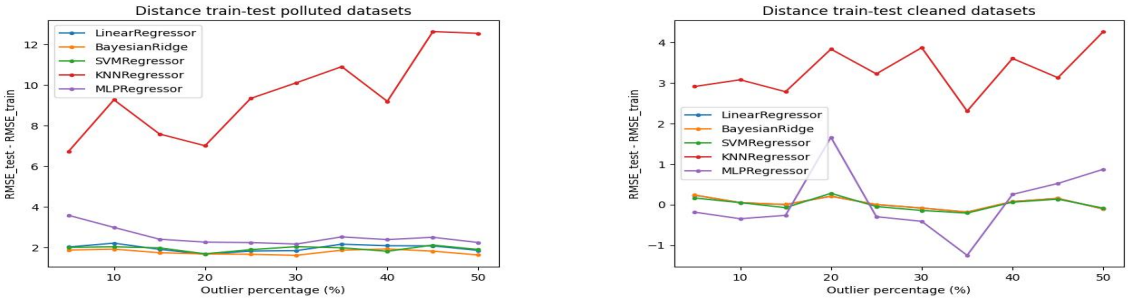


(a) Rmse polluted datasets.

(b) Rmse cleaned datasets.

Figure 4.16: RMSE comparison.

4.6.2 train/test distance - polluted vs cleaned datasets



(a) train/test distance polluted datasets. *(b) train/test distance cleaned datasets.*

Figure 4.17: train/test distance comparison.

4.6.3 Outliers removal precision

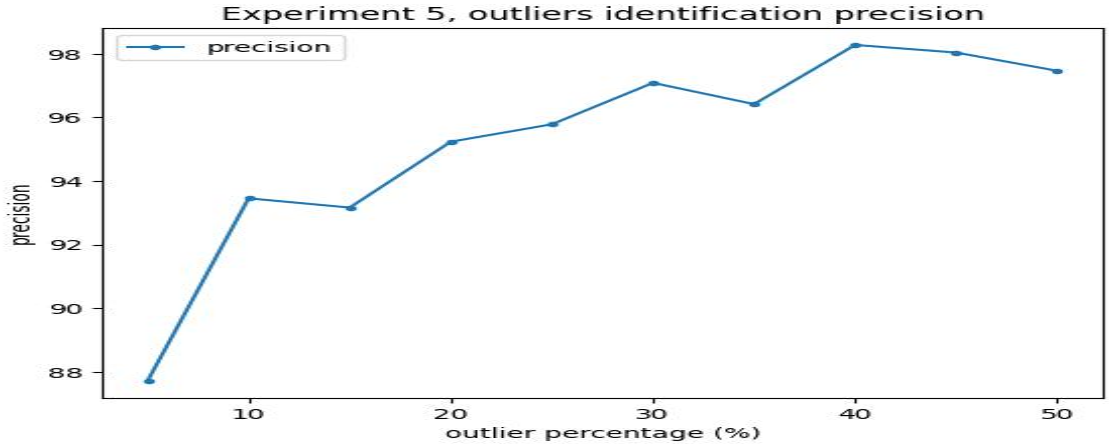
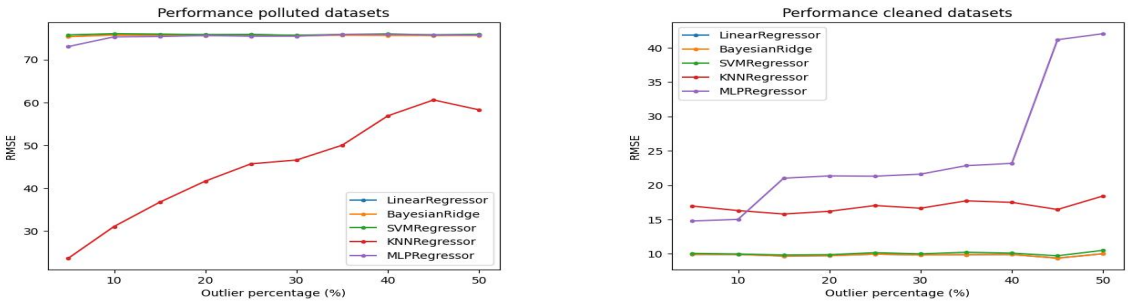


Figure 4.18: Outliers removal precision

4.7 Experiment N°6

The outliers for this experiment were generated with a uniform shift between *-10 - i10* and *10 + i10*, where $i = 6$. For $i = 6$, they were generated with a uniform shift between -70 and 70.

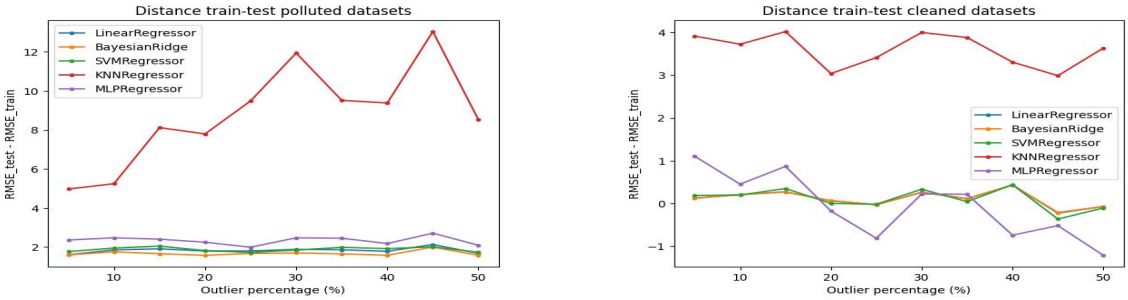
4.7.1 RMSE - polluted vs cleaned datasets



(a) Rmse polluted datasets. *(b) Rmse cleaned datasets.*

Figure 4.19: RMSE comparison.

4.7.2 train/test distance - polluted vs cleaned datasets



(a) train/test distance polluted datasets. *(b) train/test distance cleaned datasets.*

Figure 4.20: train/test distance comparison.

4.7.3 Outliers removal precision

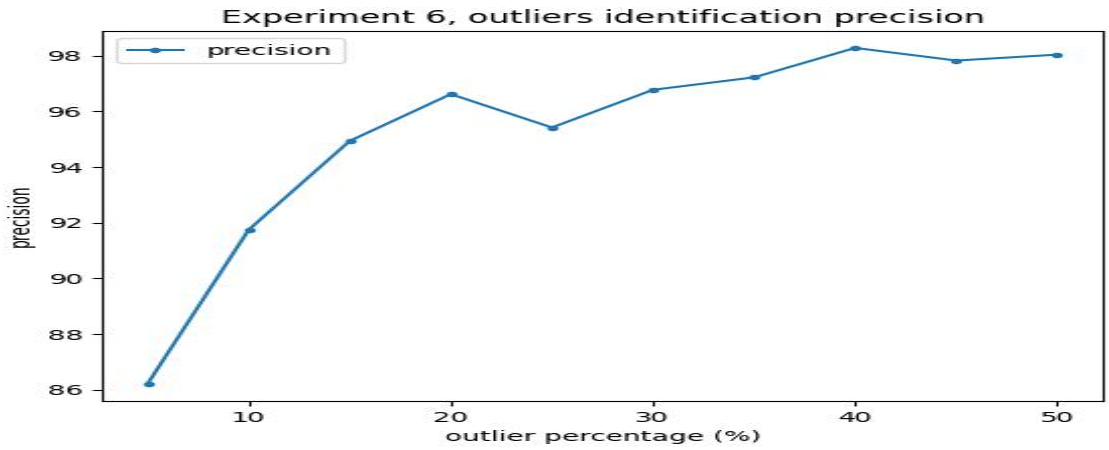
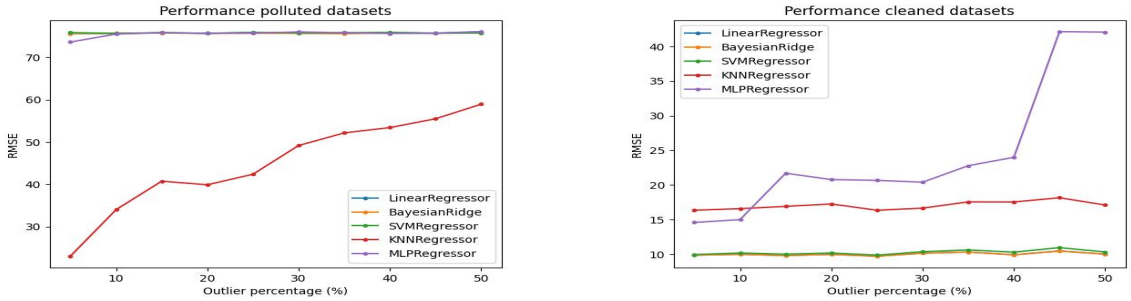


Figure 4.21: Outliers removal precision

4.8 Experiment N°7

The outliers for this experiment were generated with a uniform shift between *-10 - i10* and *10 + i10*, where $i = 7$. For $i = 7$, they were generated with a uniform shift between -80 and 80.

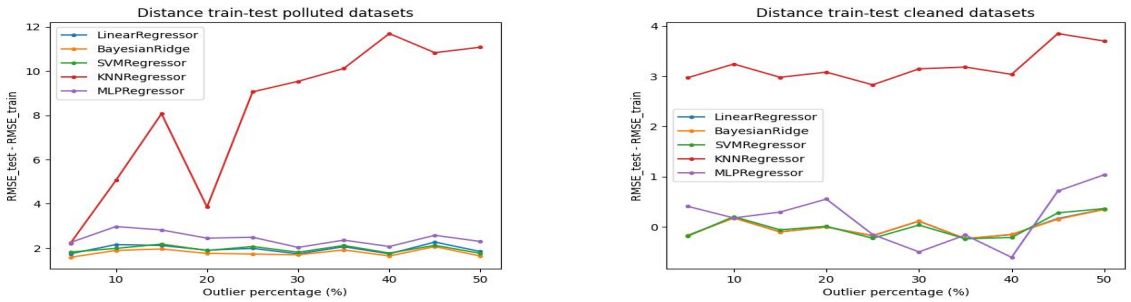
4.8.1 RMSE - polluted vs cleaned datasets



(a) Rmse polluted datasets. *(b) Rmse cleaned datasets.*

Figure 4.22: RMSE comparison.

4.8.2 train/test distance - polluted vs cleaned datasets



(a) train/test distance polluted datasets. *(b) train/test distance cleaned datasets.*

Figure 4.23: train/test distance comparison.

4.8.3 Outliers removal precision

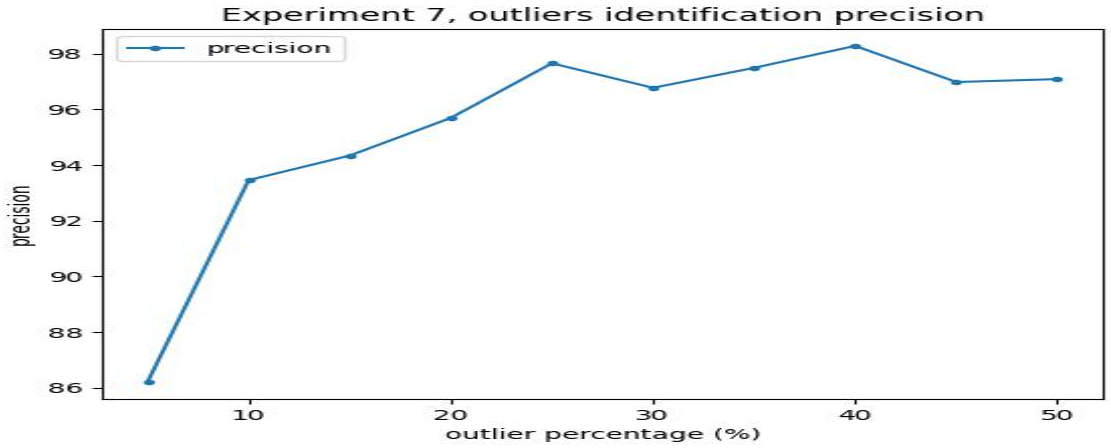
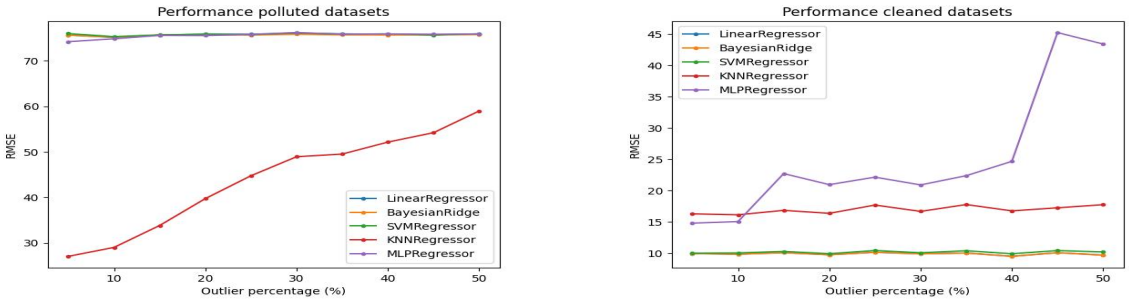


Figure 4.24: Outliers removal precision

4.9 Experiment N°8

The outliers for this experiment were generated with a uniform shift between *-10 - i10* and *10 + i10*, where $i = 8$. For $i = 8$, they were generated with a uniform shift between -90 and 90.

4.9.1 RMSE - polluted vs cleaned datasets

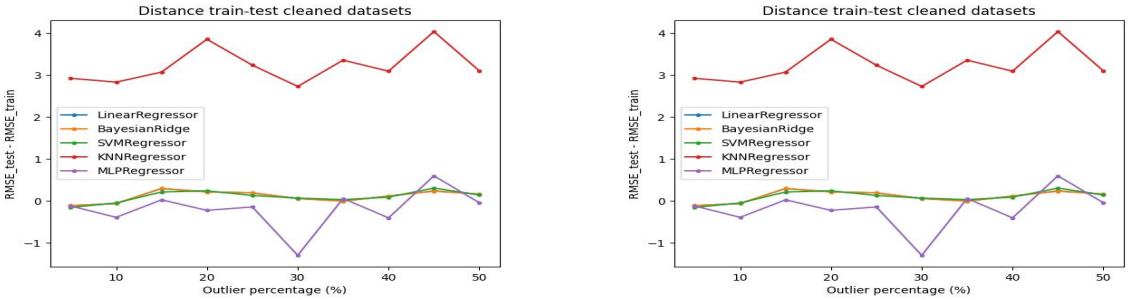


(a) Rmse polluted datasets.

(b) Rmse cleaned datasets.

Figure 4.25: RMSE comparison.

4.9.2 train/test distance - polluted vs cleaned datasets



(a) train/test distance polluted datasets. *(b) train/test distance cleaned datasets.*

Figure 4.26: train/test distance comparison.

4.9.3 Outliers removal precision

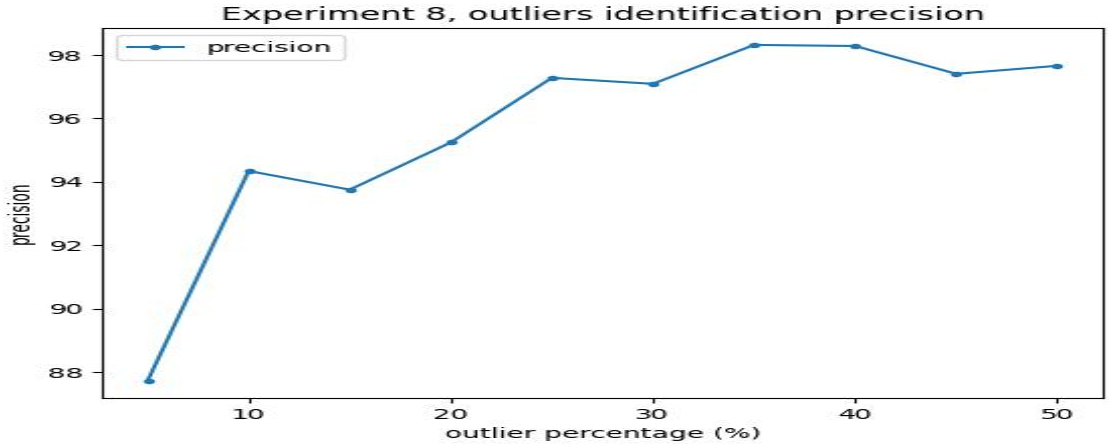
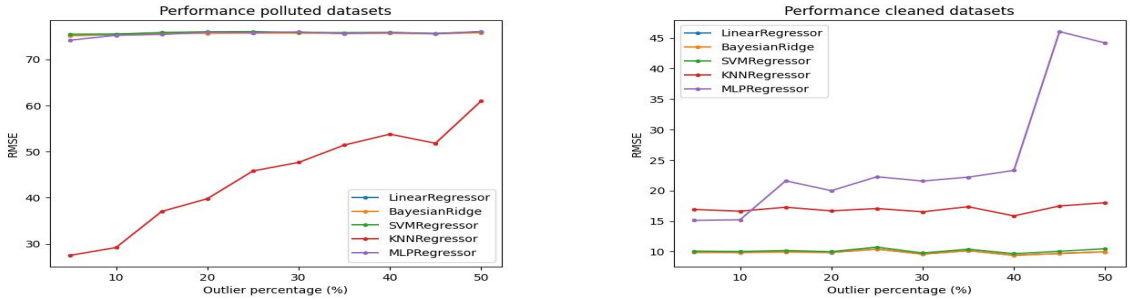


Figure 4.27: Outliers removal precision

4.10 Experiment N°9

The outliers for this experiment were generated with a uniform shift between *-10 - i10* and *10 + i10*, where $i = 9$. For $i = 9$, they were generated with a uniform shift between -100 and 100.

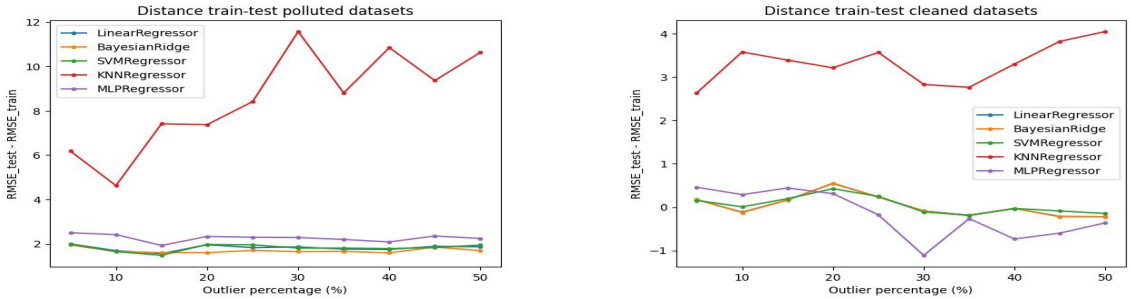
4.10.1 RMSE - polluted vs cleaned datasets



(a) Rmse polluted datasets. *(b) Rmse cleaned datasets.*

Figure 4.28: RMSE comparison.

4.10.2 train/test distance - polluted vs cleaned datasets



(a) train/test distance polluted datasets. *(b) train/test distance cleaned datasets.*

Figure 4.29: train/test distance comparison.

4.10.3 Outliers removal precision

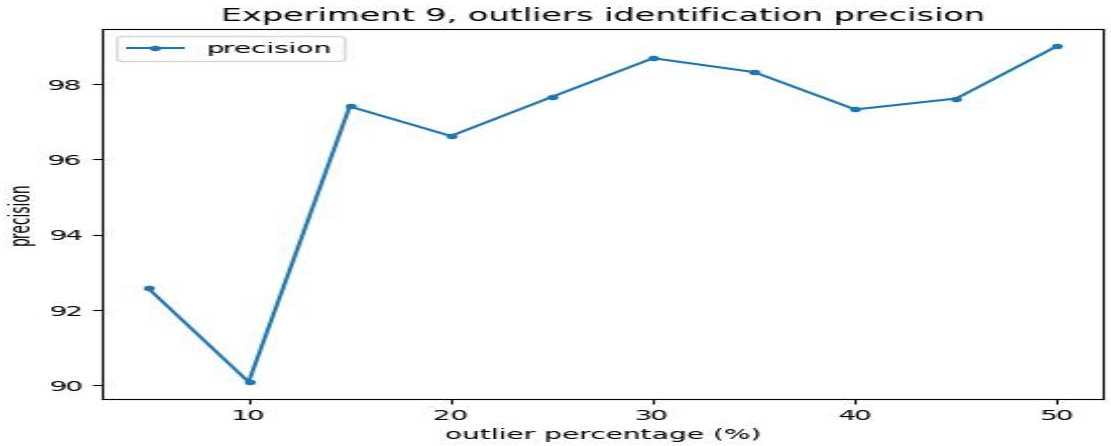


Figure 4.30: Outliers removal precision

Chapter 5

Result Analysis

5.1 Conclusion and final Analysis

Analyzing the RMSE and train-test distance graphs reveals that the trend, regardless of the conducted experiment and therefore the distance of the outliers, is increasing. This indicates that as the percentage of introduced outliers increase, the difficulty for the model to predict accurately also increases. This is due to the fact that outliers have the potential to disrupt the identification of patterns in the data.

Regarding the performance difference among various regression algorithms on datasets before and after the removal of outliers, a clear distinction in performance is evident. On the polluted datasets, the average RMSE is around 70, whereas after the cleaning process, it reduces significantly to approximately 15.

Finally, regarding the removal of outliers in each experiment, upon examining the outlier removal graphs, it's noticeable that as the experiment progresses and consequently, as the distance of the outliers increases, the precision of DBSCAN improves. It succeeds in correctly identifying approximately 98% of the outliers when the outliers distance is maximum.

Chapter 6

Tabular Data

In this last chapter, all the numerical results for each experiment for each dataset, with different percentages of outliers, are shown in tables.

On the x-axis, different types of regression algorithms are applied, and on the y-axis, the percentage of outliers introduced ranges from 5% to 50%.

Table 6.1: Experiment: 0, distance train-test on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	0.320194	0.319240	0.344815	3.348590	0.362002
10	-1.870116	-1.876443	-1.875456	3.115206	-0.650492
15	1.402633	1.406944	1.291711	3.423490	0.232895
20	-1.079900	-1.092029	-0.977919	3.450677	0.489070
25	0.318286	0.317935	0.252757	3.962182	0.478380
30	2.850167	2.858986	2.611055	5.842836	1.267001
35	3.686770	3.698735	3.491091	5.490293	1.218646
40	1.877184	1.873492	1.729146	5.542162	1.187109
45	1.333772	1.337403	1.088220	4.308310	-0.301974
50	-3.177092	-3.188239	-3.595986	2.892941	-0.489062

Table 6.2: Experiment: 0, distance train-test on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	3.397198	3.388335	1.603006	5.065547	4.610330
10	0.698682	0.705924	1.317667	4.475236	1.607845
15	1.930728	1.930973	1.878661	7.989773	2.475714
20	1.516052	1.544405	1.349890	6.996160	2.122141
25	1.926014	1.927747	1.915789	7.396818	1.657100
30	1.216590	1.263912	1.219750	8.623251	1.317708
35	2.082214	2.058516	2.233970	11.437670	2.804902
40	1.724988	1.740530	1.972887	9.024929	2.385510
45	1.908930	1.899383	1.891703	11.083931	2.248667
50	1.336293	1.369030	1.343816	11.626772	1.948955

Table 6.3: Experiment: 0, mean_perf on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	9.711035	9.710279	9.867115	16.007188	14.575422
10	11.838713	11.835237	11.914077	17.616766	16.275515
15	11.510177	11.507479	11.549122	17.025090	21.001647
20	12.634132	12.634158	12.613459	18.045538	23.787797
25	11.480977	11.477761	11.715554	17.776851	22.197920
30	16.505921	16.500447	16.436980	22.423197	24.826851
35	16.557137	16.558453	16.456296	21.789259	25.000855
40	16.244356	16.243862	16.306334	22.362831	27.987152
45	19.021097	19.025316	18.910938	22.821804	45.596915
50	24.608682	24.622095	24.456073	28.554748	46.739137

Table 6.4: Experiment: 0, mean_perf on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	62.560438	62.551521	86.586591	24.957175	52.591999
10	65.431350	65.444040	74.276422	28.468940	55.348534
15	68.103037	68.084263	68.902911	37.382502	61.192914
20	69.156078	69.148194	69.209316	40.598664	63.511697
25	71.415478	71.468433	71.305610	43.215773	66.439051
30	72.488806	72.495250	72.535561	49.902889	69.242200
35	74.236250	74.192240	74.299370	50.686999	71.655380
40	72.995219	72.984811	73.222281	54.305200	71.251674
45	72.134822	72.191496	72.464957	56.714614	70.323288
50	73.780179	73.810298	73.823334	57.905309	72.509382

Table 6.5: Experiment: 1, distance train-test on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	0.060540	0.058581	0.102288	3.829680	0.242356
10	-0.019795	-0.020694	-0.003834	2.811042	0.259417
15	-0.006582	-0.009356	0.046601	2.795295	-0.292072
20	0.700152	0.700862	0.623060	4.022584	0.585205
25	0.126693	0.129183	0.091991	3.364637	0.396692
30	-0.342125	-0.346311	-0.498477	4.094199	0.426259
35	0.025513	0.028357	0.005404	3.131645	-0.815865
40	-0.155614	-0.161497	-0.172904	4.424101	1.026665
45	-0.848331	-0.848453	-0.893127	3.729910	1.405349
50	-0.259282	-0.266733	-0.107990	3.659500	0.494835

Table 6.6: Experiment: 1, distance train-test on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	1.667216	1.680044	2.456241	5.034101	2.184511
10	1.679904	1.679623	1.675647	6.847001	2.236999
15	1.751611	1.754688	1.841634	6.421234	2.105639
20	1.928365	1.859288	1.863676	10.920745	2.406056
25	1.504063	1.531011	1.603072	9.070298	2.189013
30	2.037904	1.900089	2.234072	11.602619	2.867211
35	1.723443	1.753301	1.874346	9.610549	2.359306
40	2.029119	1.985473	2.076603	12.326844	2.584260
45	1.604642	1.683199	1.691934	10.967198	2.170174
50	2.078661	2.056653	2.026063	15.644649	2.657432

Table 6.7: Experiment: 1, mean_perf on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	9.947824	9.945985	10.086807	16.463386	15.018251
10	10.152105	10.145897	10.343219	16.564064	15.286236
15	10.186175	10.187321	10.357888	16.943471	21.663047
20	10.705572	10.699687	10.846762	17.133491	21.482226
25	9.995210	9.992522	10.181741	17.008078	20.710746
30	15.499167	15.493493	15.386719	20.830867	23.652215
35	10.033841	10.043966	10.123959	17.086295	22.285512
40	11.777129	11.780367	11.819039	18.326120	23.556389
45	10.939372	10.940872	11.115312	17.869888	43.159810
50	9.417978	9.427840	9.654177	16.823477	38.739588

Table 6.8: Experiment: 1, mean_perf on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	70.432615	70.467079	73.338577	26.556214	60.565564
10	73.598607	73.574058	73.770245	31.845298	67.460187
15	73.522782	73.552818	73.689580	36.299508	69.494725
20	75.347851	75.354215	75.429893	42.401733	71.415837
25	74.124170	74.151919	74.455140	42.408449	71.864401
30	75.525550	75.620268	75.514791	51.857391	74.455755
35	75.102007	75.086596	75.160352	52.743516	74.164946
40	74.718485	74.765098	74.693373	56.860457	74.096800
45	75.103904	75.084865	75.128089	57.471795	74.556625
50	75.282002	75.390286	75.328952	59.953461	74.950178

Table 6.9: Experiment: 2, distance train-test on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	0.059890	0.061896	0.021028	2.981177	0.080719
10	0.035883	0.036152	0.033821	2.906464	-0.116816
15	0.196702	0.196532	0.209030	3.370034	-0.300735
20	0.087374	0.086738	0.100166	3.577270	0.428288
25	0.036700	0.043275	-0.049388	3.170299	0.188637
30	0.112345	0.119437	0.022037	3.534350	-0.038103
35	0.307863	0.306981	0.284207	3.310341	0.388721
40	0.303002	0.306585	0.241238	3.792516	-0.044494
45	0.156532	0.154773	0.180910	3.993899	1.369499
50	0.060118	0.070402	-0.073378	3.104220	-1.294451

Table 6.10: Experiment: 2, distance train-test on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	2.320330	2.264414	2.749635	6.217094	3.364890
10	2.334410	2.297354	2.265179	4.688236	3.734833
15	1.658892	1.678413	1.933645	6.823232	2.164949
20	2.168072	2.113118	2.177483	7.869058	2.776758
25	2.277922	2.159352	2.410249	11.220425	2.823409
30	1.646834	1.687902	1.843064	6.957157	2.167245
35	2.205596	2.005710	2.033636	12.360736	3.015362
40	2.142095	1.943060	2.179701	9.741101	2.645826
45	1.853661	1.748156	1.838723	11.658028	2.521809
50	2.002064	1.909923	1.908603	12.651549	2.369377

Table 6.11: Experiment: 2, mean_perf on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	10.074199	10.073334	10.236893	16.760487	14.885894
10	9.880071	9.879158	10.029144	16.259347	15.257598
15	10.102731	10.099579	10.311584	16.488246	22.545032
20	9.881469	9.881637	10.086374	16.564347	21.414296
25	10.099427	10.098641	10.306364	17.183691	20.983926
30	9.987573	9.985360	10.256299	17.512591	21.917155
35	9.855439	9.849925	10.193896	17.370895	22.032387
40	10.039601	10.037079	10.377348	17.385753	25.632225
45	9.797170	9.790649	10.346035	18.789054	45.389188
50	10.119377	10.118096	10.607317	18.356856	43.368037

Table 6.12: Experiment: 2, mean_perf on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	74.843491	74.787841	75.365428	24.185539	66.849648
10	74.302754	74.319655	74.225233	30.949106	70.371010
15	74.954778	74.926311	75.267091	35.396241	72.383372
20	74.566875	74.625221	74.777026	39.136578	72.204512
25	75.126767	75.144332	75.226032	48.814538	74.621262
30	75.358521	75.342035	75.454172	47.312288	74.786113
35	75.711396	75.668481	75.937228	51.350191	75.160594
40	75.546321	75.603942	75.562945	53.470701	75.179967
45	75.851445	75.710545	75.844934	57.590157	75.586636
50	75.234430	75.281591	75.275036	59.551728	74.849971

Table 6.13: Experiment: 3, distance train-test on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	-0.055102	-0.052521	-0.081918	3.362808	0.411761
10	0.246602	0.244443	0.285963	3.163918	0.786276
15	0.054226	0.054400	0.049559	3.647612	0.798781
20	0.077041	0.082225	-0.009798	3.089705	-0.633348
25	0.200759	0.202132	0.195186	3.314963	-0.703798
30	0.057150	0.066461	-0.076725	2.935634	0.306240
35	0.147624	0.141572	0.259809	3.867472	0.998432
40	-0.006954	-0.003757	-0.099549	3.168537	-0.624241
45	0.659740	0.649813	0.712000	4.536037	1.205775
50	-0.159957	-0.151672	-0.217905	3.655485	0.279043

Table 6.14: Experiment: 3, distance train-test on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	1.619881	1.606615	1.589433	6.800519	2.323217
10	2.112937	2.032963	2.052864	7.524468	3.153606
15	1.762263	1.628998	1.865232	5.025290	2.378215
20	1.721896	1.728937	1.770242	6.950166	2.413308
25	1.674911	1.674860	1.784325	7.166710	2.285316
30	1.716693	1.713640	1.887269	7.300360	2.100597
35	1.977910	1.612945	1.935410	10.442185	2.508101
40	1.916063	1.785085	2.118507	13.040574	1.961880
45	2.330463	1.918750	2.282909	12.455982	2.979045
50	1.735867	1.577369	1.856618	10.681021	2.203891

Table 6.15: Experiment: 3, mean_perf on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	9.751393	9.751427	9.876496	16.637157	14.768432
10	9.743176	9.740395	9.965841	16.536426	15.208682
15	10.029646	10.031736	10.160154	16.825834	21.750107
20	9.897674	9.900697	10.060204	16.673567	20.958958
25	10.037216	10.038814	10.235110	16.728488	22.644237
30	10.224834	10.226273	10.439132	17.066190	22.230558
35	9.600423	9.596998	9.976640	17.450698	22.707633
40	10.036697	10.030313	10.438031	19.206036	24.370716
45	10.259791	10.249707	10.693263	17.451312	42.132596
50	9.472313	9.465514	10.014216	17.955113	44.643338

Table 6.16: Experiment: 3, mean_perf on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	75.264348	75.208097	75.040714	28.057705	70.527714
10	75.125939	75.035225	75.131593	30.202253	72.277580
15	75.708023	75.657580	75.850184	31.224015	73.744893
20	75.101192	75.125629	75.299275	41.246622	74.587893
25	75.515107	75.560539	75.626008	41.078015	74.843809
30	75.855755	75.736438	75.885431	48.139605	75.732962
35	75.930623	75.645447	76.007278	51.269827	75.659724
40	75.811880	75.732845	76.031780	53.982312	75.498283
45	75.579303	75.644568	75.728370	56.131323	75.558138
50	75.925692	75.687202	76.013443	60.027631	75.988780

Table 6.17: Experiment: 4, distance train-test on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	-0.121718	-0.121468	-0.087590	3.230328	0.229615
10	-0.010889	-0.014775	0.035665	2.900736	0.257500
15	0.077819	0.076054	0.069509	4.210404	0.730805
20	0.028237	0.026569	0.073110	3.939933	0.718557
25	0.256891	0.257868	0.231325	3.327856	-0.427001
30	0.085304	0.074559	0.208208	3.632270	0.703537
35	0.264169	0.261845	0.303974	3.858227	0.827334
40	0.288244	0.277646	0.436029	3.849375	1.833603
45	0.085957	0.082883	0.174437	3.751087	0.375994
50	-0.235441	-0.239267	-0.157939	3.249528	-0.968405

Table 6.18: Experiment: 4, distance train-test on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	1.649512	1.627987	1.506966	6.658961	3.583860
10	1.753098	1.730272	1.798136	5.704095	2.329481
15	1.894289	1.842653	1.686224	7.812740	2.620035
20	1.953309	1.780096	1.969126	6.005651	2.601119
25	2.055563	1.925397	1.969533	10.702503	2.691949
30	2.524804	2.323329	2.543281	11.083702	2.986825
35	1.882897	1.911660	1.990829	9.035385	2.305234
40	1.923892	1.572512	1.796142	9.858914	2.331601
45	1.823136	1.602914	1.975770	12.181575	2.234973
50	2.055277	1.962744	2.004542	10.496462	2.308352

Table 6.19: Experiment: 4, mean_perf on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	9.754501	9.751097	9.968372	15.908304	14.403348
10	9.776550	9.772659	9.996868	16.578787	15.098378
15	9.764382	9.762136	9.947943	17.270891	22.014708
20	9.799665	9.800287	10.034402	16.144457	21.521114
25	9.701404	9.699349	9.959860	16.953886	21.850454
30	10.093867	10.093132	10.345922	16.412358	20.622446
35	10.071361	10.065645	10.419360	18.271208	23.414101
40	9.720858	9.713717	10.128129	17.143913	24.707125
45	9.839265	9.840410	10.195688	17.460837	43.595429
50	9.946579	9.945767	10.380916	17.974587	42.698187

Table 6.20: Experiment: 4, mean_perf on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	74.534150	74.500687	75.054901	28.358769	69.645837
10	75.323398	75.321771	75.230913	34.619620	73.798153
15	75.591556	75.593926	75.458936	35.414967	74.791283
20	75.749901	75.673808	75.787899	39.866805	75.352700
25	75.779806	75.724814	75.848430	46.539890	75.976910
30	75.640623	75.668866	75.694607	54.928075	75.746830
35	75.237786	75.283908	75.285283	49.445081	75.129121
40	75.865655	75.644879	75.884734	55.542736	75.816399
45	75.806178	75.661669	75.844584	55.453605	75.834224
50	75.528193	75.570421	75.646632	57.514681	75.571878

Table 6.21: Experiment: 5, distance train-test on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	-0.001975	-0.001715	0.016771	3.100010	0.308918
10	0.193000	0.193175	0.186510	3.945942	0.462742
15	0.111644	0.115557	0.053502	3.496803	-0.621040
20	-0.118902	-0.113862	-0.186697	2.583092	-0.931929
25	-0.194512	-0.191010	-0.215105	2.567610	-0.365642
30	0.132297	0.134178	0.083522	3.683991	0.773709
35	0.211253	0.218464	0.099075	3.123719	-1.117215
40	0.441237	0.448858	0.349436	3.956775	0.946974
45	0.182814	0.187957	0.169165	3.376943	-0.462554
50	0.439515	0.441570	0.404491	4.221578	1.261260

Table 6.22: Experiment: 5, distance train-test on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	1.654239	1.651005	1.972785	3.848958	1.850763
10	1.911846	1.799284	1.875247	7.093212	2.643403
15	2.076527	1.909920	1.942261	7.018695	2.886724
20	1.933899	1.578372	1.828478	9.029566	2.511122
25	1.928544	1.700536	1.826390	7.278809	2.523826
30	1.812941	1.756977	1.912406	6.217103	2.130578
35	1.765613	1.572334	1.898250	8.748307	2.134870
40	1.772122	1.586494	1.745731	9.970525	1.921333
45	1.965762	1.731255	2.023778	12.570246	2.367629
50	2.011212	1.711355	2.120421	14.086262	2.525445

Table 6.23: Experiment: 5, mean_perf on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	9.799533	9.798947	9.936788	16.515609	15.209035
10	9.799222	9.794858	10.022614	16.635430	14.393359
15	9.771998	9.772750	9.920968	16.482144	21.373269
20	9.610953	9.614971	9.746303	15.685684	21.063504
25	9.941745	9.942285	10.077415	16.453699	21.291413
30	10.039117	10.039956	10.339915	17.284151	20.872063
35	10.155541	10.151877	10.443608	18.157994	23.440264
40	10.122170	10.109146	10.668604	17.499901	26.489181
45	10.001302	10.001088	10.469046	17.691092	43.026323
50	10.404081	10.400929	10.715926	18.706307	42.916604

Table 6.24: Experiment: 5, mean_perf on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	75.538721	75.460790	75.822315	24.143149	72.024856
10	76.027212	75.816914	76.008475	34.917794	74.832715
15	75.648479	75.661576	75.644943	34.507476	75.012326
20	76.018897	75.735532	75.920609	41.967093	75.999469
25	75.886896	75.690990	76.048929	45.576814	75.864104
30	75.718158	75.698793	75.974514	49.495674	75.776662
35	75.862513	75.644885	76.001565	52.026991	75.931371
40	75.757587	75.644827	75.838007	50.754187	75.719537
45	75.897141	75.780904	75.984014	59.409794	76.006123
50	75.774736	75.689619	75.871531	62.320786	75.856176

Table 6.25: Experiment: 6, distance train-test on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	0.546942	0.549994	0.513485	3.945039	0.717041
10	0.287978	0.285229	0.320172	2.632978	-0.115064
15	0.377800	0.375793	0.399563	3.561792	0.349118
20	-0.239520	-0.243255	-0.185988	3.816818	0.915891
25	0.187015	0.182813	0.263392	2.951557	-0.454213
30	-0.232863	-0.238160	-0.176759	3.221073	-0.381766
35	-0.018708	-0.019676	0.005382	2.758482	-0.722945
40	0.058234	0.061899	0.012697	3.128322	-0.588778
45	0.070067	0.084591	-0.171222	1.834228	-1.070334
50	0.254443	0.264702	0.070917	3.510942	-1.008938

Table 6.26: Experiment: 6, distance train-test on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	2.077903	1.957618	2.209002	6.315377	2.699485
10	1.628450	1.688341	1.690008	5.048644	2.371998
15	2.104150	1.791726	2.055603	8.071621	2.458721
20	1.784377	1.730417	1.842450	6.929765	2.102207
25	1.939811	1.663782	1.925967	8.617847	2.663145
30	2.149568	1.705874	2.197949	12.217399	2.467296
35	2.046445	1.765251	2.169684	10.464680	2.746847
40	1.965023	1.697236	1.932337	10.023281	2.353969
45	2.395743	2.170975	2.359256	12.637624	2.595760
50	1.911446	1.699379	1.888853	13.196973	2.339798

Table 6.27: Experiment: 6, mean_perf on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	9.902557	9.899338	10.110070	16.301621	14.853069
10	9.780009	9.777969	9.991821	17.016365	15.330695
15	9.947337	9.946084	10.104950	16.614099	21.595388
20	9.977718	9.982945	10.064411	16.659627	21.909026
25	10.084522	10.074876	10.404440	17.349089	22.058360
30	9.801977	9.799161	10.040810	17.229707	21.065455
35	9.944934	9.934794	10.317957	16.744980	20.359060
40	9.721335	9.728347	9.891487	17.208160	26.053317
45	9.844230	9.837541	10.365180	17.957597	45.971858
50	10.470595	10.471944	10.790321	18.327262	40.727813

Table 6.28: Experiment: 6, mean_perf on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	74.988023	75.047880	75.286324	22.217150	72.675375
10	75.871515	75.792551	75.974489	29.719248	74.973393
15	75.663346	75.641334	75.861933	35.544769	75.279252
20	75.982803	75.852780	75.904882	40.322504	75.815999
25	75.972683	75.674750	75.962697	44.476885	76.067798
30	75.707633	75.644869	75.756259	47.366877	75.508647
35	75.730177	75.664029	75.804286	51.188288	75.709285
40	75.726374	75.669240	75.888366	52.850228	75.933164
45	75.690164	75.609656	75.712154	54.006263	75.522081
50	75.970732	75.751076	75.907169	63.294584	76.080563

Table 6.29: Experiment: 7, distance train-test on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	0.218678	0.219532	0.209550	3.072856	-0.237110
10	-0.085131	-0.080315	-0.127123	3.386935	0.680809
15	0.320428	0.318576	0.324224	3.644759	0.717247
20	-0.100617	-0.098410	-0.150939	2.626505	-1.389506
25	0.333028	0.332488	0.350223	3.283227	-0.223113
30	0.191959	0.179506	0.348577	3.421442	0.937273
35	-0.777013	-0.786375	-0.692138	3.808466	1.221404
40	0.226152	0.215329	0.323637	4.705438	0.344545
45	-0.085980	-0.081426	-0.216681	2.568123	-0.522680
50	0.182207	0.163382	0.420759	3.755854	1.124731

Table 6.30: Experiment: 7, distance train-test on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	1.804906	1.735035	1.853552	5.573101	2.991953
10	2.188105	2.060877	2.229850	5.477392	2.895993
15	1.722777	1.715049	1.708073	8.643221	2.214750
20	1.717029	1.599783	1.755110	11.809465	2.308189
25	1.835625	1.665059	1.853120	8.996628	2.311252
30	2.005921	1.740953	1.989526	11.183616	2.527324
35	1.670574	1.609115	1.685244	11.515631	2.118587
40	1.986089	1.659099	2.087339	11.533459	2.508814
45	2.247508	1.975720	2.186600	11.998384	2.870217
50	1.807449	1.572388	1.727835	9.048931	1.976084

Table 6.31: Experiment: 7, mean_perf on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	9.641577	9.642351	9.785163	16.063695	14.436488
10	9.988773	9.993005	10.089502	16.389560	15.690153
15	9.892193	9.888456	10.083481	16.554602	21.981976
20	9.977377	9.979289	10.116346	16.006075	20.552944
25	9.902408	9.902950	10.065987	16.339814	21.176122
30	10.066068	10.066487	10.236584	17.008268	22.247177
35	11.829001	11.817165	12.012258	19.681696	24.323341
40	10.373343	10.365459	10.634615	18.369617	25.320919
45	10.050686	10.040203	10.398243	17.583729	44.213164
50	9.657439	9.648129	10.207029	19.195053	42.268112

Table 6.32: Experiment: 7, mean_perf on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	75.696038	75.618518	75.825998	24.262746	74.411160
10	75.656110	75.547933	75.739429	30.243951	74.998801
15	75.472821	75.396625	75.427053	37.722380	75.281696
20	75.674919	75.653748	75.773228	41.120445	75.845018
25	75.695237	75.665685	75.684439	47.392643	75.512031
30	75.962417	75.706126	75.955080	49.997325	76.099645
35	75.812962	75.684927	75.791355	53.267795	75.837919
40	75.902667	75.720803	75.892350	52.527771	75.875459
45	75.902264	75.760515	75.953023	57.635514	76.011447
50	75.944158	75.754768	75.949340	58.449278	75.841950

Table 6.33: Experiment: 8, distance train-test on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	0.136511	0.136436	0.148626	2.722186	0.078499
10	0.359336	0.356280	0.416577	3.424271	0.569016
15	-0.155973	-0.157250	-0.159851	3.394134	0.249904
20	0.218170	0.213818	0.269623	4.416805	0.295684
25	0.395977	0.402895	0.290650	3.028415	-0.021789
30	-0.013825	-0.010837	-0.080276	3.515107	-0.221823
35	0.028402	0.033717	-0.023190	2.971630	-0.706247
40	0.339931	0.338802	0.306801	3.086846	-0.454477
45	0.271376	0.277034	0.147834	3.881025	-0.195734
50	-0.252919	-0.259838	-0.071953	4.130206	1.764417

Table 6.34: Experiment: 8, distance train-test on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	1.873237	1.829375	1.935226	4.657095	2.730691
10	1.887722	1.768594	1.777080	6.847593	2.538097
15	1.738938	1.595502	1.853940	7.490498	1.988525
20	1.964597	1.794282	1.961774	8.216757	2.426859
25	2.345493	2.260779	2.508644	12.123259	3.183002
30	1.932395	1.683461	1.825636	9.818920	2.449388
35	1.757782	1.572409	1.797508	9.165446	2.064890
40	1.694612	1.573657	1.743898	10.699702	2.123561
45	1.660733	1.658598	1.809254	10.775345	1.946153
50	1.815599	1.705056	1.786558	11.174650	2.092348

Table 6.35: Experiment: 8, mean_perf on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	9.935136	9.934144	10.121021	15.858246	14.797765
10	9.920664	9.915145	10.193070	17.063906	15.798232
15	9.873550	9.877079	9.972281	16.532934	21.500714
20	9.671698	9.671574	9.926775	15.981445	20.733547
25	9.970696	9.965748	10.272979	16.872536	21.743135
30	10.201425	10.201136	10.426584	17.425057	21.319457
35	10.001487	10.001467	10.206813	17.950370	22.925708
40	10.146444	10.141082	10.548316	17.615252	23.205522
45	9.628367	9.623311	10.022599	17.762711	43.535638
50	9.581208	9.576528	10.118916	18.604441	44.101362

Table 6.36: Experiment: 8, mean_perf on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	75.053478	75.089282	75.099467	25.712623	74.127241
10	75.548399	75.614143	75.684274	32.444699	75.080502
15	75.972103	75.756745	76.001707	34.663939	75.715338
20	75.924075	75.766529	75.938440	41.505237	75.912750
25	75.690862	75.619890	75.609372	44.863838	75.691529
30	75.805066	75.654750	75.790593	52.539037	75.867671
35	75.819633	75.647231	75.892817	51.364719	75.796239
40	75.737398	75.644821	75.772832	55.756038	75.821066
45	75.574148	75.639605	75.621105	54.736846	75.571478
50	75.745928	75.718710	75.836744	60.139349	75.574820

Table 6.37: Experiment: 9, distance train-test on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	-0.051926	-0.050546	-0.050743	3.219929	0.285561
10	0.150645	0.146909	0.197625	3.575750	0.481924
15	-0.010590	-0.008401	-0.032402	3.251715	-0.082394
20	0.110667	0.113531	0.107915	2.906393	0.120970
25	0.189192	0.191566	0.124570	3.202883	-0.969586
30	-0.276780	-0.271922	-0.327355	2.805577	-0.091847
35	-0.007098	-0.013089	0.069020	3.598676	0.250499
40	0.162616	0.162060	0.118682	3.328919	-0.238607
45	-0.330868	-0.337156	-0.140485	4.487694	2.269730
50	0.280474	0.269585	0.385296	3.951342	0.768588

Table 6.38: Experiment: 9, distance train-test on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	1.900292	1.753944	1.812573	5.084941	2.505555
10	2.041072	1.934832	1.779331	4.728861	2.544641
15	1.735375	1.740177	1.713945	5.014840	2.284614
20	1.852310	1.602903	1.863871	7.859012	2.253095
25	2.022143	1.780699	1.913555	5.585482	2.504818
30	2.033488	1.595059	2.108611	9.739524	2.413327
35	1.938358	1.620656	1.730486	10.026886	2.279031
40	1.978126	1.725001	1.945289	12.419255	2.364266
45	2.112569	1.941713	2.418687	12.627183	2.588418
50	1.847795	1.591250	1.960427	12.259931	2.064775

Table 6.39: Experiment: 9, mean_perf on cleaned dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	9.938953	9.938167	10.096750	16.589482	15.012518
10	10.217606	10.208856	10.497730	16.686835	15.431510
15	9.912600	9.918582	9.998598	16.325043	20.176401
20	9.848500	9.853911	9.978478	16.399772	22.006393
25	10.293753	10.296109	10.436518	16.798212	21.434031
30	9.925534	9.921295	10.227360	17.283484	21.937798
35	10.146557	10.145394	10.347680	17.615207	21.222152
40	9.755779	9.756529	10.086937	17.329992	24.563140
45	10.119312	10.118445	10.653695	18.441951	41.015363
50	9.758919	9.761956	10.066060	18.407790	43.126892

Table 6.40: Experiment: 9, mean_perf on polluted dataset

	LinearRegressor	BayesianRidge	SVMRegressor	KNNRegressor	MLPRegressor
5	75.756350	75.738647	75.697388	26.683627	74.477997
10	76.188537	75.937505	75.873210	30.826362	75.846724
15	75.399873	75.381181	75.406458	36.318396	75.314228
20	75.954635	75.722203	76.118389	39.420208	75.826583
25	75.678499	75.699867	75.732524	43.109191	75.588164
30	75.927715	75.659929	75.886272	48.926023	75.810641
35	75.808197	75.644863	75.851482	52.355961	75.550870
40	75.845751	75.727777	76.003541	55.265569	75.903731
45	75.857557	75.763950	76.084587	58.522142	75.932156
50	75.835652	75.682428	75.989449	60.209295	75.970773