POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

DIQ Project Report

DQ Issues: Duplication and Variable Types
ML Task: Classification

Authors: **Matteo Spreafico**
Federico Toschi

Project ID: **20**

Academic Year: 2023-2024

Contents

Contents **i**

1 Project Goal and Setup Choices **1**
1.1 Project Goal . . . 1
1.2 Duplication Issue Setup Choices . . . 2
1.2.1 Data Collection . . . 2
1.2.2 Experiments . . . 2
1.3 Variable Types Issue Setup Choices . . . 3
1.3.1 Data Collection . . . 3
1.3.2 Experiments . . . 3

2 Pipeline Implementation **5**
2.1 Step #0: Colab Environment Setup . . . 5
2.2 Step #1: Data Collection . . . 5
2.3 Step #2: Data Pollution . . . 6
2.3.1 Duplication Issue Pollution Functions . . . 6
2.3.2 Duplication Issue Experiments . . . 7
2.3.3 Variable Types Issue Pollution Functions . . . 8
2.3.4 Variable Types Issue Experiments . . . 9
2.4 Step #3: Data Analysis and Evaluation . . . 10
2.5 Step #4: Data Preparation . . . 11
2.6 Step #5: Data Analysis and Evaluation . . . 12

3 Results **13**
3.1 Duplication Experiments Discussion . . . 13
3.1.1 After Pollution, Before Data Preparation . . . 13
3.1.2 After Data Preparation . . . 15
3.2 Variable Types Experiments Discussion . . . 19

List of Figures **23**

List of Tables **25**

1 | Project Goal and Setup Choices

In this chapter, we will discuss the goal and the setup choices of our project.

1.1. Project Goal

Data Quality (DQ) is becoming more and more important for successful Machine Learning (ML) analysis pipelines. However, requirements for having good DQ are changing: we must no longer just ensure a good level of DQ for the traditional dimensions (i.e. Completeness, Accuracy, Consistency), as the success of a ML analysis can depend on a multitude of new data issues.

The project pipeline is made of the following steps:

1. Data Collection;
2. Data Pollution;
3. Data Analysis and Evaluation;
4. Data Preparation;
5. Data Analysis and Evaluation.

The Data Collection and Data Analysis and Evaluation steps were given with the project assignment, whereas the Data Pollution and the Data Preparation steps have been implemented by us.

We have been assigned to inject two DQ issues into the numeric dataset synthesized at the first step of the pipeline. Such DQ issues are **non-exact duplicates** and **variable types**. Concerning the first DQ issue, we have implemented both the Data Pollution and the Data Preparation steps of the pipeline, whereas concerning the second one, we have implemented only the Data Pollution step, as no preparation technique has been discussed in class with reference to such a scenario.

In the following pages, we will assess the impact of both the duplication (traditional) and the variable types (new) DQ issue on a **classification** task. Plus, we will assess the impact of deduplication techniques on the polluted dataset, to check whether the ML task performs any better (or worse) after Data Preparation.

1.2. Duplication Issue Setup Choices

We will now discuss the **10** experiments we have performed for the duplication issue.

1.2.1. Data Collection

For the following experiments, we have decided to keep the default Data Collection parameters, since we have obtained significant results with them.

1.2.2. Experiments

We have imagined 4 different, realistic scenarios of non-exact duplicates injection:

1. **Rounding-off**. Some sensors may have different sensitivities, thus they may sense the same real-world event with different levels of precision. If we add those values to the same dataset, we will a have non-exact duplicate. We have replicated this scenario by duplicating some tuples and rounding-off some figures in the duplicates.
2. **Gaussian noise**. Some sensors may collect data polluted with noise. If two sensors sense the same real-world event, they will likely provide slightly different values. We have replicated this scenario by duplicating some tuples and adding a Gaussian noise to each value.
3. **Scaling**. Some sensors may collect data that are stored with different units of measurement. If two sensors with different units of measurement sense the same real-world event, or if a sensor repeats a measurement after its settings have been changed, we will have a non-exact duplicate. We have replicated this scenario by duplicating some tuples and scaling their values with respect to some powers of 10.
4. **Feature Swapping**. Some human users, especially when datasets are difficult to interpret, may input data incorrectly. In particular, they may input some value in an incorrect feature column. If data happen to be inputted twice, we have a non-exact duplicate. We have replicated this scenario by duplicating some tuples and swapping some features.

For each scenario, we have performed two experiments. They differ due to the percentage of pollution we injected into the dataset, which translates into the number of tuples that have been duplicated and polluted.

Lastly, we have performed another two experiments in which we have injected some non-exact duplicates synthesized by means of the **Tanimoto similarity measure**. We have chosen to do so in order to obtain some non-exact duplicates that may be *easily* detected by standard record linkage methods, and we have chosen the Tanimoto similarity measure because of its good trade-off between time of execution and complexity (compared with cosine similarity and Pearson correlation based methods).

Unlike the strictly angular consideration in cosine similarity or the direct proportional assessment in Pearson correlation, the Tanimoto coefficient incorporates a comparative analysis of both vector magnitudes and orientations, computed using the following formula:

$$T(A, B) = \frac{A \cdot B}{||A||^2 + ||B||^2 - A \cdot B}$$

This method provided more analytically valuable results among our experiments with linearly-correlated data.

1.3. Variable Types Issue Setup Choices

We will now discuss the **10** experiments we have performed for the variable types issue.

1.3.1. Data Collection

For the following experiments, we have decided to keep the default Data Collection parameters, since we have obtained significant results with them.

1.3.2. Experiments

The goal of the variable types experiments is to understand the impact of adding various types of features (i.e. boolean, string,. . .) into a numeric dataset when performing a classification task. In order to do so, we have implemented two ways of adding new data:

1. **Correlated** types generation. New features are synthesized by applying some rules to the original, numeric features.
2. **Uncorrelated** types generation. New features are randomly synthesized.

We have performed **7** correlated and **3** uncorrelated types generation experiments. Each experiment in the same category differs in the number and the variety of the new synthesized features.

2 | Pipeline Implementation

In this chapter, we will discuss how we have implemented the pipeline of this project. Each section in this report refers to the corresponding section in the Python notebook.

2.1. Step #0: Colab Environment Setup

In this section, we have written some code to setup the Google Colab environment we have used to do our project. Namely, we have:

- mounted our Google Drive account to access the scripts given by the professor;
- imported many useful libraries (in particular, we have imported the professor's scripts for Data Collection, Data Analysis and Evaluation, and results plotting);
- fixed randomness by using a fixed seed (42) for both the Python's and the NumPy's random module;
- set some project-specific constants: list of classification algorithms, number of experiments, seed of the Data Collection function.

2.2. Step #1: Data Collection

In this section, we have created three datasets. The first two datasets are identical, but one has been employed for the duplication issue, whereas the second one has been employed for the variable types issue. We decided to do so because this way it was simpler to play with the Data Collection parameters while deciding which experiments were more interesting to perform with reference to the variable types issue, but in the end we have decided to keep the default parameters. The third and last dataset, instead, is made up of only 5 samples, and is used as a toy example to effectively show what each pollution function does. From now on, such dataset will be referred to as "example dataset".

Lastly, notice that we have checked (before creating the datasets) that the number of redundant variables must not be smaller than 0. This is because, if such an assertion was violated, then we would have a dataset with more informative features than the ones it actually has, which is clearly impossible.

2.3. Step #2: Data Pollution

In this section, for each DQ issue, we will first discuss the pollution functions we have implemented, and then how we have used them in our experiments. Notice that, here, we will just discuss the high-level details of interest. If you wish to have a further in-depth knowledge about how these functions work, they are thoroughly commented in our notebook.

2.3.1. Duplication Issue Pollution Functions

We have implemented the following pollution functions:

- `pollute_round_off`

  It returns the original dataset with a percentage of duplicates polluted according to the rounding-off rule (i.e. a random number of features is rounded-off to a certain level of precision, in such a way that no exact duplicate is ever produced).

- `pollute_gaussian_noise`

  It returns the original dataset with a percentage of duplicates polluted with a Gaussian noise of zero mean and standard deviation equal to the 10% of the dataset's standard deviation.

- `pollute_scaling`

  It returns the original dataset with a percentage of duplicates polluted according to the scaling rule (i.e. each feature is multiplied for a random power of 10, such that the exponent is between -12 and 12, both ends included). Notice that, even though each attribute in the polluted tuple is scaled with the same power of 10, different polluted tuples are scaled with a randomly chosen power of 10.

- `pollute_swapping`

  It returns the original dataset with a percentage of duplicates polluted according to the swapping rule (i.e. a random number of features are swapped).

- `pollute_similarity`

  It returns the original dataset with a percentage of duplicates, synthesized starting from the original dataset with a similarity measure (and a given similarity percentage). Several helper functions have been implemented to support the similar data generation. As previously discussed, we have implemented and considered using three similarity measure based generation mechanisms (i.e. cosine similarity based, Pearson correlation based, Tanimoto similarity based), but eventually decided to use only the Tanimoto similarity based method.

- `pollute_duplication`

  It pollutes the given dataset with non-exact duplicates, injecting the given percentage of duplicates with the given method. Then, shuffles the result and returns it. This function, along with the appropriate parameters, will be used by the ten experiments to pollute the dataset.

Notice that each pollution function comes with one or more usage examples that employ the example dataset to show how they work in practice.

2.3.2. Duplication Issue Experiments

Now, we will present the parameters with which the 10 duplication experiments have been performed:

1. The dataset will be slightly polluted (5%) with the rounding-off pollution function;
2. The dataset will be heavily polluted (50%) with the rounding-off pollution function;
3. The dataset will be slightly polluted (5%) with the Gaussian noise pollution function;
4. The dataset will be heavily polluted (50%) with the Gaussian noise pollution function;
5. The dataset will be slightly polluted (5%) with the scaling pollution function;
6. The dataset will be heavily polluted (50%) with the scaling pollution function;
7. The dataset will be slightly polluted (5%) with the swapping pollution function;
8. The dataset will be heavily polluted (50%) with the swapping pollution function;
9. The dataset will be slightly polluted (5%) with the Tanimoto similarity pollution function, where the non-exact duplicates are 75% similar to the original tuples;
10. The dataset will be heavily polluted (50%) with the Tanimoto similarity pollution function, where the non-exact duplicates are 75% similar to the original tuples.

With these experiments, we want to assess how each kind of duplicate and the amount of duplicates impact the classification performance metrics.

2.3.3. Variable Types Issue Pollution Functions

As previously said, we implemented two categories of data pollution functions for the Variable Types issue: **Correlated** pollution, and **Non-Correlated** pollution. In both cases, the pollution functions were written using a reference to the dataset's original **informative** features. This will be referred to as *initial features*, and can be found in the code as the constant `INIT_FEATURES`.

For Correlated pollution, we have implemented the following functions:

- `pollute_boolean`

  It returns the original dataset with a new feature of the Boolean type, generated following one of two rules implemented. The rules for this pollution apply non-linearities to randomly chosen features, only between the initial features, and return a boolean value based on a check on the sign/magnitude of the value (i.e. check if the value is greater than 0, or if the value is greater than the median of the initial features).

- `pollute_string`

  It returns the original dataset with a new feature of the String type, generated following one of three rules implemented. The rules for this pollution generate a string based on the values of randomly chosen features among the initial ones. Two rules generate a string via conversions from integer numbers to ASCII characters. The last rule categorizes each tuple by applying a complex, non-linear transformation to three predetermined features. It then compares these transformed values to calculated percentiles, assigning each data point a category ranging from "Very Low" to "Very High" based on where it falls within the dataset's distribution.

- `pollute_numeric`

  It returns the original dataset with a new feature of the Numeric type, generated following one of two rules implemented. The rules for this pollution generate a float based on a complex, non-linear transformation that makes use of logarithms, exponentials, or trigonometric functions. These transformations are applied to some features randomly chosen among the initial ones.

- `pollute_date`

It returns the original dataset with a new feature of the Date type, generated following one rule implemented. This rule randomly chooses some features among the initial ones, applies a non-linear transformation, truncates the result to avoid overly large data points, and converts the final result to a date (i.e. calculates a trigonometric transformation, and uses the decimals of the result as a timestamp).

- `pollute_correlated_data_types`

  It pollutes the given dataset with correlated features of the data types specified. Then, shuffles the result and returns it. This function, along with the appropriate parameters, will be used in the ten experiments to pollute the dataset.

For Non-correlated pollution, the same set of functions was implemented, but this time following a fully random approach in the generation of the values.

In addition, we made sure to limit the possible results of the random **numeric** function using the minimum and maximum values among the initial features. This was further parameterized with the addition of a *scale* parameter: this allows for the generation of numbers outside the boundary defined by the minimum and maximum values, up to a certain multiplicative factor.

Furthermore, another key aspect in the engineering of these pollution functions was to ensure that the transformations were not "too simple" for the ML algorithms that were going to be evaluated on our data. This meant using non-linear transformations, and sometimes adding some small random Gaussian noise.

Notice that each pollution function comes with one or more usage examples that employ the example dataset to show how they work in practice.

2.3.4. Variable Types Issue Experiments

Now, we will present the parameters with which the 10 variable types experiments have been performed:

1. The dataset will be slightly polluted with **2x Boolean Correlated** features.
2. The dataset will be slightly polluted with **2x Numeric Correlated** features.
3. The dataset will be slightly polluted with **2x String Correlated** features.
4. The dataset will be highly polluted with **5x String Correlated** features.
5. The dataset will be slightly polluted with **2x Numeric, 2x String Correlated** features.

6. The dataset will be highly polluted with **2x Boolean, 2x String, 2x Numeric, 2x Date Correlated** features.
7. The dataset will be very highly polluted with **50x Date, 50x Numeric Correlated** features.
8. The dataset will be slightly polluted with **1x Date Non-Correlated** feature.
9. The dataset will be slightly polluted with **1x String, 1x Date Non-Correlated** features.
10. The dataset will be slightly polluted with **2x String, 1x Numeric Non-Correlated** features.

With these experiments, we want to assess how various data types impact the performance of various ML algorithms for classification. Specifically, we want to answer the following questions:

- How impactful is feature variety compared to quantity?
- Which types of features impact performance the most?
- Can adding correlated features help models in learning a more effective representation?
- Which models tend to fit noise, and which models can distinguish relevant features from redundant/irrelevant ones?

2.4. Step #3: Data Analysis and Evaluation

In this section, we have used the given scripts to assess and plot the performance of the classification task with every polluted dataset. For each experiment, three plots have been generated, each comparing the given classification algorithms according to the following metrics:

1. **Performance**, i.e. F1 weighted score.
2. **Distance**, i.e. training set's F1 weighted score minus test set's F1 weighted score.
3. **Speed** of training.

The given classification algorithms are: **Decision Tree, Logistic Regression, K-Nearest Neighbors (KNN), Random Forest, AdaBoost, Multilayer Perceptron (MLP)**.

Notice that, before using the Data Analysis scripts on the variable types experiments, the polluted datasets have been converted into the DataFrame format, to allow the use of another (given) function that encodes categorical variables. This step is necessary, as the classification algorithms do not accept categoric values.

For the reasons we mentioned in the first chapter, concerning the variable types experiments, the pipeline stops here.

2.5. Step #4: Data Preparation

In this section, we have performed some standard Data Cleaning techniques in order to see if the prepared datasets have better metrics with respect to the polluted ones for the same classification tasks. In this case, the cleaning task is a deduplication one. Since the injected duplicates are not exact, we have performed a Record Linkage task by using the `record_linkage` Python module.

Notice that we have put ourselves in a realistic scenario, i.e. in the perspective of a user who does not know the pollution functions (that should mimic real-world pollution) that have been applied to it. Hence, we have not implemented any *ad hoc* deduplication function. Additionally, since the dataset is not interpretable and its distribution has a small variance, potential data exploration tasks would hardly be useful in this context, especially if the number of data points were to increase.

Since the dataset is numeric, we had to use a suitable numeric, non-exact comparison function. We have chosen the one provided by the `record_linkage` module, which offers five comparison methods: step, linear, exponential, Gaussian, and squared. We have picked all of them but the step method, since it had quickly proven to be ineffective, and applied them to each candidate link, for each pair of features with the same index. Notice that the choice of using the full indexer was made possible by the rather small (1000) number of data points we had to deal with. Plus, we had to set a condition that could discriminate between non-exact duplicates and different data points: after many attempts, we decided to impose that the sum of the similarity coefficients computed for each column by the comparison function had to be greater than a threshold, that we chose with fine tuning. If the threshold proved to be too strict (i.e. if no duplicate was found), we let the record linkage function try again with a smaller threshold, but without exceeding a certain lower bound (which was decided with fine tuning as well).

For each experiment and for each Record Linkage method, we have finally performed Data Deduplication and obtained the prepared datasets.

2.6. Step #5: Data Analysis and Evaluation

In this final section, we have used once again the given scripts to assess and plot the performance of the classification task with every prepared dataset, averaging the data obtained with the different Record Linkage methods (that turned out to be quite similar).

3 | Results

In this chapter, we will discuss the results of our experiments.

3.1. Duplication Experiments Discussion

We will now discuss the results of the duplication experiments.

3.1.1. After Pollution, Before Data Preparation

The following plots show the results of the 10 duplication experiments before the Data Preparation step.

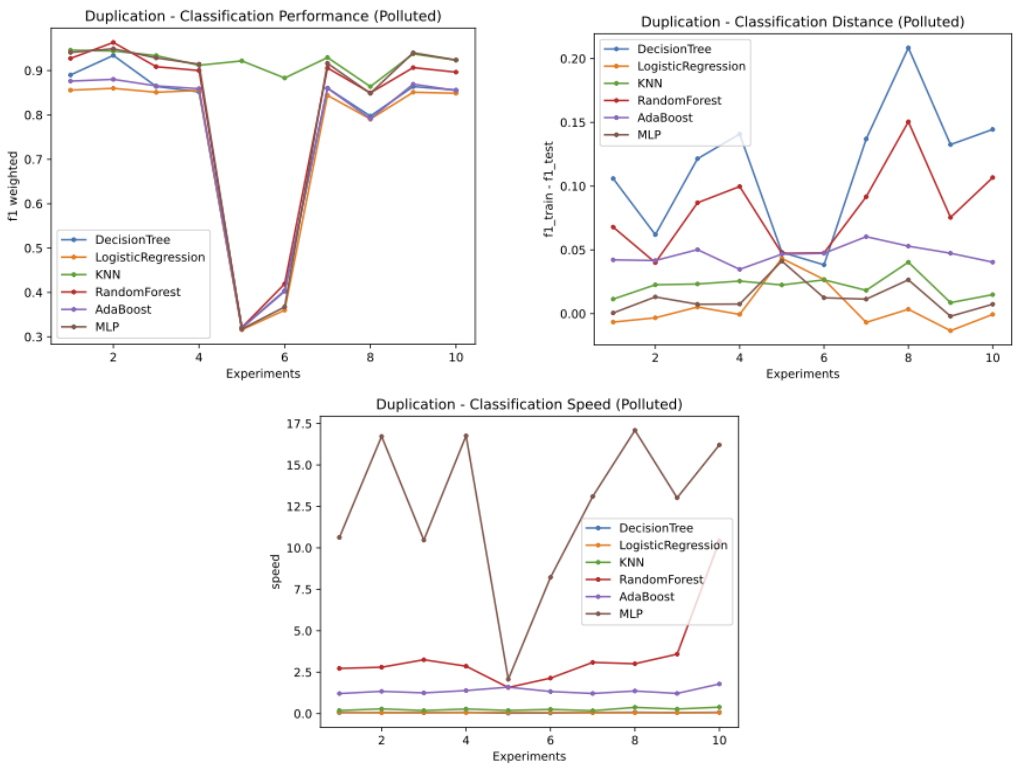

Figure 3.1: Classification Metrics of the Polluted Dataset (Duplication)

By looking at the plots in figure 3.1, we can state the following:

1. The rounding-off pollution function does not have a significant impact on the classification performance, in both cases (slight and heavy pollution), although there is a small sign of overfitting for the Decision Tree and Random Forest algorithms in the first experiment.

2. The Gaussian noise pollution function does not have a significant impact on the classification performance, although the distance plot highlights again that the Decision Tree and Random Forest algorithms are more likely to fit noise in the data.

3. The scaling pollution function, however, has a huge impact on the classification performance of every algorithm except KNN. For both experiments, we can see from the plots that there is no sign of overfitting, thus the performance drop could be associated with each algorithm's weaknesses. For example, decision trees are sensitive to the specific thresholds to split the data: extreme scaling can lead to poor performance because the splits may not generalize well. Similarly, an algorithm like Logistic Regression assumes linear separability. Power scaling can distort this linearity, making it hard for Logistic Regression to fit an appropriate decision boundary. Lastly, neural networks can be sensitive to the scale of the input data: extreme values can cause issues with gradient-based optimization, which can potentially lead to poor performance if the network is not carefully regularized.

   On the other hand, an algorithm like KNN, due to its majority voting system can be more resilient to scaled duplicates. This is because KNN naturally adapts to local decision boundaries: even if some points are completely out of scale, "correct" data points will still fall under the correct cluster of points.

   Although it may seem counter-intuitive that KNN's performance drops in the second scaling experiment, this could be caused by some statistical fluctuations. If we assume that the duplicated points are uniformly distributed in the range of scaling powers (from 10^{-12} to 10^{12}), we can expect around the same amount of points of the same class to be in the same "scale". The issue is that the clusters that are formed are significantly smaller compared to the original ones, and thus sometimes there may be more neighbors of one class closer to some point of the opposite class.

4. The swapping pollution function has, interestingly, a minor impact on the classification performance of every algorithm. Because of how we implemented this pollution function, the data points introduced are duplicates, but, if one is not aware of how these data points were created, they could be seen also as new, fake data points. For

this reason, we were expecting a bigger drop in performance in both cases; however, due to the small variance in the distribution of our dataset and to its lack of interpretability, swapping features did not create data points that are clearly wrong, but just noisy points.

A confirmation of this can be found by looking at the magnitude of the drop between experiments 7 and 8. We expected a drop in performance due to the introduction of noise; however, it was significantly bigger compared to the Gaussian pollution experiments (3-4) due to the "standard deviation" of the noise no longer being the 10% of the standard deviation of the dataset, but being related to the variance of each feature.

5. The similarity pollution function did not show a significant impact on the classification performance of all the algorithms, although DecisionTree and RandomForest show signs of slight overfitting. In addition, introducing artificial points similar to true points with the same label did not act as pollution, but supported the decisions of the classification algorithms. Furthermore, MLP showed a bigger gain in performance compared to other algorithms, which can be attributed to its ability to understand non-linear patterns in data.

Finally, the speed plot did not highlight any interesting results, apart from confirming that neural networks such as MLP are most often significantly slower compared to the other algorithms.

3.1.2. After Data Preparation

Before starting our discussion, it is important to highlight that significant improvements with respect to the classification metrics should not be always expected. This is because we have implemented very diverse pollution functions, while the deduplication procedure was rather standard, for the reasons we have already presented. We have produced a table that shows how many non-exact duplicates each Record Linkage method has detected.

For each classification metric, we have also produced a table that shows, for each ML algorithm, the results before and after the Data Preparation step. These tables, along with the plots, will help us support our conclusions. For each ML algorithm, the "before" values are shown on the left, whereas the "after" values are shown on the right. With reference to the "after" columns, the table cell is colored in orange if the "after" metric is worse than the "before" metric, in green if it is better.

The following images show both the plots and the aforementioned tables for the 10 duplication experiments after Data Preparation.

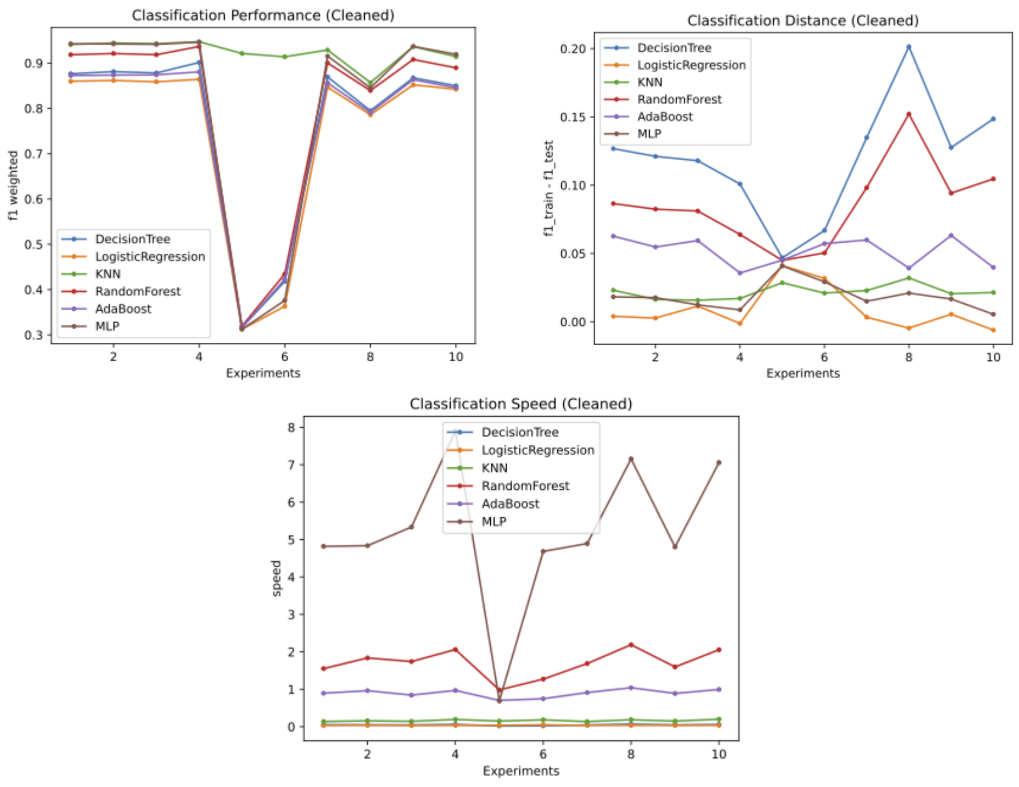

Figure 3.2: Classification Metrics of the Cleaned Dataset (Duplication)

Exp	DT		LR		KNN		RF		AdaBoost		MLP	
1	0.8857	0.8764	0.8558	0.8600	0.9460	0.9412	0.9269	0.9185	0.8762	0.8726	0.9424	0.9426
2	0.9375	0.8813	0.8600	0.8616	0.9441	0.9444	0.9639	0.9212	0.8802	0.8736	0.9480	0.9424
3	0.8675	0.8782	0.8511	0.8588	0.9337	0.9429	0.9012	0.9185	0.8654	0.8742	0.9302	0.9415
4	0.8502	0.9014	0.8553	0.8641	0.9119	0.9472	0.9025	0.9366	0.8591	0.8807	0.9158	0.9463
5	0.3194	0.3166	0.3158	0.3125	0.9218	0.9213	0.3212	0.3193	0.3203	0.3169	0.3178	0.3125
6	0.4090	0.4180	0.3601	0.3631	0.8832	0.9140	0.4122	0.4346	0.4023	0.4227	0.3660	0.3761
7	0.8655	0.8694	0.8440	0.8468	0.9293	0.9290	0.9059	0.9003	0.8603	0.8565	0.9146	0.9154
8	0.7975	0.7950	0.7908	0.7860	0.8639	0.8570	0.8511	0.8394	0.7913	0.7912	0.8455	0.8455
9	0.8607	0.8674	0.8511	0.8521	0.9376	0.9365	0.9071	0.9080	0.8694	0.8635	0.9409	0.9370
10	0.8550	0.8501	0.8489	0.8423	0.9233	0.9145	0.8984	0.8895	0.8550	0.8456	0.9219	0.9194

Table 3.1: Average Performance Before and After Data Preparation

	DT		LR		KNN		RF		AdaBoost		MLP	
1	0.1083	0.1268	-0.0066	0.0040	0.0114	0.0231	0.0671	0.0865	0.0422	0.0627	-0.0015	0.0183
2	0.0661	0.1211	-0.0033	0.0027	0.0226	0.0163	0.0408	0.0825	0.0417	0.0547	0.0142	0.0176
3	0.1238	0.1180	0.0052	0.0115	0.0233	0.0158	0.0874	0.0811	0.0502	0.0594	0.0035	0.0124
4	0.1417	0.1009	-0.0006	-0.0012	0.0256	0.0171	0.0967	0.0639	0.0347	0.0357	0.0119	0.0087
5	0.0471	0.0468	0.0434	0.0412	0.0225	0.0286	0.0456	0.0450	0.0467	0.0451	0.0413	0.0409
6	0.0386	0.0668	0.0268	0.0317	0.0265	0.0210	0.0313	0.0504	0.0473	0.0572	0.0220	0.0292
7	0.1350	0.1349	-0.0068	0.0033	0.0183	0.0229	0.0909	0.0982	0.0604	0.0599	0.0121	0.0150
8	0.2059	0.2014	0.0035	-0.0046	0.0404	0.0321	0.1520	0.1521	0.0529	0.0393	0.0290	0.0210
9	0.1282	0.1276	-0.0134	0.0055	0.0086	0.0206	0.0770	0.0942	0.0474	0.0632	0.0013	0.0167
10	0.1445	0.1485	-0.0006	-0.0061	0.0149	0.0214	0.1072	0.1046	0.0404	0.0398	0.0079	0.0055

Table 3.2: Average Distance Before and After Data Preparation

Exp	DT		LR		KNN		RF		AdaBoost		MLP	
1	0.0735	0.0526	0.1237	0.0348	0.2486	0.1386	1.5697	1.5509	0.9825	0.8947	4.3616	4.8185
2	0.1456	0.0511	0.0718	0.0388	0.6599	0.1578	1.8727	1.8380	0.9302	0.9629	6.1993	4.8368
3	0.0599	0.0477	0.0765	0.0343	0.3824	0.1438	1.9010	1.7407	0.8114	0.8442	4.4211	5.3343
4	0.0808	0.0616	0.1144	0.0360	0.3049	0.1946	1.9684	2.0619	0.9337	0.9685	6.4932	7.9005
5	0.0272	0.0196	0.0633	0.0378	0.2762	0.1501	0.9748	0.9832	0.6165	0.7025	0.5771	0.6893
6	0.0400	0.0249	0.0978	0.0497	1.0418	0.1834	1.1557	1.2713	0.6996	0.7471	4.2513	4.6836
7	0.0632	0.0469	0.0852	0.0359	0.1247	0.1363	1.7206	1.6872	0.8027	0.9122	5.9405	4.8936
8	0.1048	0.0684	0.1022	0.0374	0.1776	0.1863	2.0248	2.1902	1.0412	1.2588	7.8303	7.1545
9	0.0555	0.0486	0.0619	0.0363	0.1339	0.1496	1.5903	1.5971	0.7998	0.8892	5.9092	4.8035
10	0.1057	0.0583	0.0506	0.0364	0.1884	0.2007	1.9517	2.0565	0.9353	0.9945	6.2898	7.0566

Table 3.3: Average Speed Before and After Data Preparation

Exp. No	Linear	Exponential	Gauss	Square
1	42	41	50	50
2	444	438	500	500
3	3	1	26	1
4	2	1	2	9
5	24	24	27	27
6	202	202	219	219
7	2	2	13	26
8	8	8	15	15
9	2	2	12	27
10	3	3	15	1

Table 3.4: Data points removed by each Record Linkage algorithm

Before going into any analysis, here's a quick reminder about the pollution magnitude of the experiments:

- Odd-numbered experiments inject 50 duplicates (5% of the dataset).
- Even-numbered experiments inject 500 duplicates (50% of the dataset).

By looking at both the plots and the tables, we can state the following:

1. The rounding-off pollution seems to be easily detectable, as every record linkage function achieves a good result, assuming the removed data points are actually duplicates. However, due to the nature of this pollution function, it's likely that these are removing data points that supported the algorithms in making correct decisions, as rounding off would introduce a very similar data point with the same label as the original. For this reason, we mostly see performance drops after cleaning, with the biggest ones in the second experiment. Similarly with the distance metric, with less data the algorithms have less confidence, and thus show slightly more overfitting, again especially in the second experiment.

2. The Gaussian noise pollution, on the other hand, does not seem to be detectable with our deduplication setup. After running some experiments, we noticed that, on average, the similarity between a clean data point and its polluted version, with

our Gaussian pollution setup, will be between 0.85-0.89. The issue with this is that it's far too low compared to our threshold for duplicate detection, but given the nature of our dataset and its little variance, reducing the threshold to catch these duplicate points would mean inevitably removing true data points. Regarding the classification performance of the various algorithms, we see a trend with each of them obtaining a higher score. We assume this is due to the implementation of our Gaussian pollution function, because, having a standard deviation of the noise proportional to the one of our dataset, it's likely the duplicate points are not distorted enough to change the underlying patterns. Thus, after removing even a few duplicate points, that are probably very alike, the models are able to work even better with the variability introduced by this noise.

3. The scaling pollution is, approximately, detected by half. Considering the implementation of our function, and the range of powers with which a data point can be scaled, we can assume that about half of the data points have been downscaled with negative powers of 10. Due to this effect, and due to the underlying logic of the `record_linkage` library's `compare.numeric` function (i.e. for two tuples, it takes their absolute difference in value), it's likely that the detected duplicates are exactly those duplicates in the negative range of powers of 10.

   Regarding the classification performance, we see two trends: for the first experiment, each algorithm's performance worsened lightly, while for the second, all of them improved, sometimes by a significant margin. These effects can be attributed to the fact that, in the first case we have very few duplicates left which act as noise, while in the second case, it's more likely for similar points to be together in the same/nearby range of power of 10, and thus some of them will be interpreted as useful information. Still, the performance remains very poor due to the high number of completely out-of-scale data points.

4. The swapping pollution is, by its nature, undetectable with our current deduplication setup. This shows in the table 3.4, as in both experiments the removed data points are very few, and most likely true data points, that happened to be close to each other. This is supported by our deduplication setup, as, if no duplicate is found, the threshold gets slightly lower at each iteration. If one was aware of this type of pollution, the correct deduplication procedure would include comparisons across features. As a byproduct of this, we see no major improvements/drops in performance, but just variations which could be attributed to some statistical fluctuations. Regarding the distance metrics, we see that the Decision Tree and Random Forest algorithms heavily overfit. This can be attributed to the implementation of

our swapping pollution function that, as previously stated, potentially introduces noise of the magnitude of the feature that is being swapped.

5. The similarity pollution, in the same way as the Gaussian pollution, is difficult to detect with our deduplication setup. Again, this shows in table 3.4, where we see that very few points are removed in both experiments. This is caused, as in the Gaussian case, by the threshold used in our implementation, and cannot be adjusted to a lower value to avoid detecting true points. Although the pollution detection is similar to the Gaussian one, when looking at the performance metrics, this is more similar to the case of the swapping function, where we don't see any major improvements/drops. This can be attributed to the similarity percentage used in our setup, and the intrinsic nature of the Tanimoto similarity. Because it introduces a non-linear relationship between two duplicate points, these may be acting sometimes as useful information, sometimes as noise. When looking at the distance measure, however, we see that the same algorithms that showed to be fitting noise in the last two experiments, do the same here, but to a smaller magnitude, suggesting our interpretation of the new data points being ambivalent may be correct.

Finally, the speed plot did not highlight any new, interesting result.

3.2. Variable Types Experiments Discussion

We will now discuss the results of the 10 variable types experiments. First, we will show the related plots.

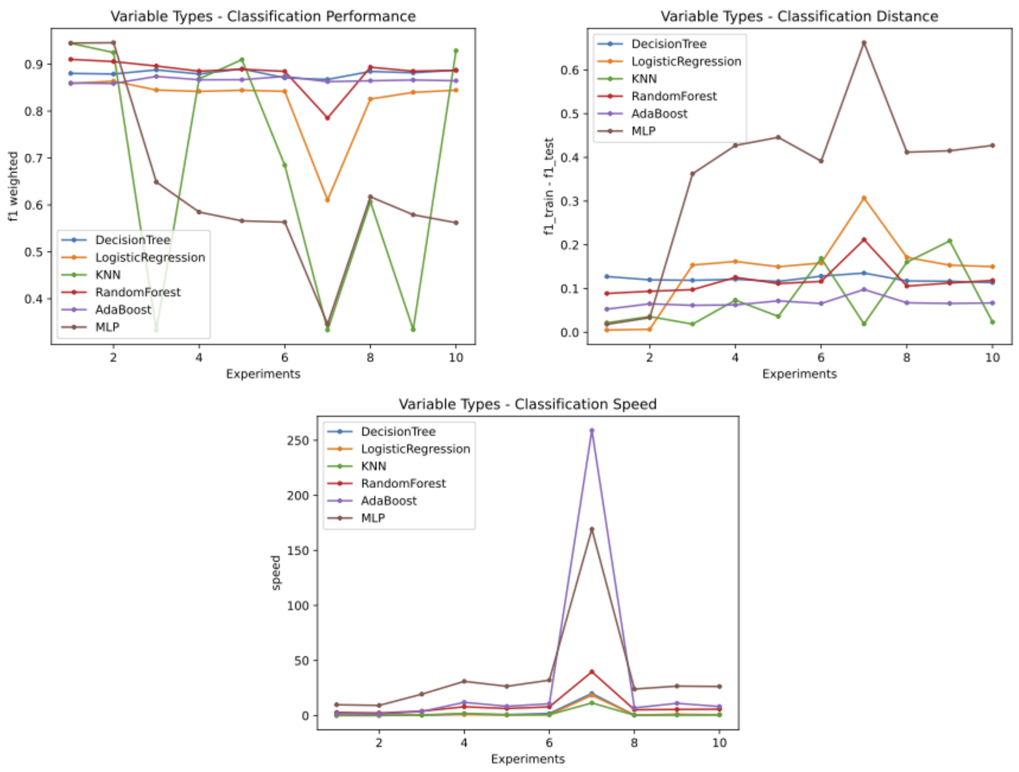

Figure 3.3: Classification Metrics of the Polluted Dataset (Variable Types)

By looking at the plots in figure 3.3, we can answer the questions we were asking ourselves in section 2.3.4:

How impactful is feature variety compared to quantity?

Looking at the performance plot, we can distinguish three different behaviors:

1. The resilient algorithms: Decision Tree, Logistic Regression, Random Forest, and AdaBoost. These algorithms maintain a good level of performance throughout the experiments, except for experiment 7, where the excessive quantity of redundant features interferes with the learning process of part of them: Decision Tree and AdaBoost show a consistent level of performance, while the others have a noticeable drop in performance, although still sufficient overall. Moreover, in this group, Logistic Regression is impacted the most, showing also some signs of overfitting in the distance graph.

2. KNN. This algorithm seems to be affected more by variety than quantity. Experiments with a small amount of additional features of the type String and/or Date, like 3 and 9, have the biggest impact. On the other hand, experiments like 4 show that having more features of the same type contributes to im-

proving performance. Interestingly, experiment 6 does not correspond to a significant drop in performance: this could be caused by the fact that some types of features seem to be helping the model, while others seem to be more confusing. Specifically, experiments with Boolean and Numeric features correspond to peaks in the performance of KNN, while Dates and Strings to drops. In addition, when taken to an extreme, like in experiment 7, having too many features seems to be affecting anyways the model a lot, although it does not result in its worse performance. Lastly, this model seems to be equally affected by both correlated and non-correlated features of the type String and Date: we can see in the plot that experiments 3 and 9 correspond to the lowest drops of the performance metric.

3. MLP. This algorithm seems to be affected by both conditions, although experiments with more quantity correspond to lower performances. Additionally, by looking at the distance graph, we can clearly see the overfitting trend of this model, which is at its worst during experiment 7.

Which types of features impact performance the most?

As seen in the previous analysis, the feature types impacting the most are both Dates and Strings. This is probably caused by the process of categorization before feeding the data to the algorithms: both of these are transformed into one-hot vectors, and for features with a high level of entropy (generated by part of the rules of the string pollution, and, in general, by the date pollution function), this means having as many distinct strings/dates as there are entries in the dataset. On the contrary, Numeric **correlated** features seem to help models the most in achieving better performance, especially those like MLP that are more able to understand the underlying non-linear relationships with which these were created.

Can adding correlated features help models in learning a more effective representation?

Before trying to answer, we must say there has been no significant experimental evidence of such a thing.

Our hypothesis, as explained above, is that features like the Numeric ones, although redundant, can help learn a more complex representation of data, especially for neural networks like MLP.

Other models that naturally select relevant features, ignoring redundant/useless ones, do not show any sign of better performance with any type of feature.

Which models tend to fit noise, and which models can distinguish relevant features from redundant/irrelevant ones?

From the previous analyses, it is clear that models like Decision Tree, Logistic Regression, AdaBoost, and Random Forest can distinguish the relevant features, ignoring other correlated/non-correlated ones, in any combination and quantity, although when taken to an extreme, as previously said, two of them are slightly impacted.

Contrarily, KNN and MLP seem to be affected by both correlated and non-correlated data types, in both positive (for numeric/boolean types) and negative (for strings and dates) ways. We can also further distinguish these two by looking at the distance plot: indeed, while MLP shows a clear trend of overfitting from experiment 3 onwards, KNN never goes above a distance of 0.2.

List of Figures

3.1 Classification Metrics of the Polluted Dataset (Duplication) 13
3.2 Classification Metrics of the Cleaned Dataset (Duplication) 16
3.3 Classification Metrics of the Polluted Dataset (Variable Types) 20

List of Tables

3.1 Average Performance Before and After Data Preparation 16
3.2 Average Distance Before and After Data Preparation 16
3.3 Average Speed Before and After Data Preparation 17
3.4 Data points removed by each Record Linkage algorithm 17