William Stucchi - 10661711

Giada Silvestrini - 10659711

Academic Year: 2023/2024

Data and Information Quality course project

Project ID: 13

Data quality issues: Accuracy and Dependency

Machine learning task: Classification

Index:

1. Project description

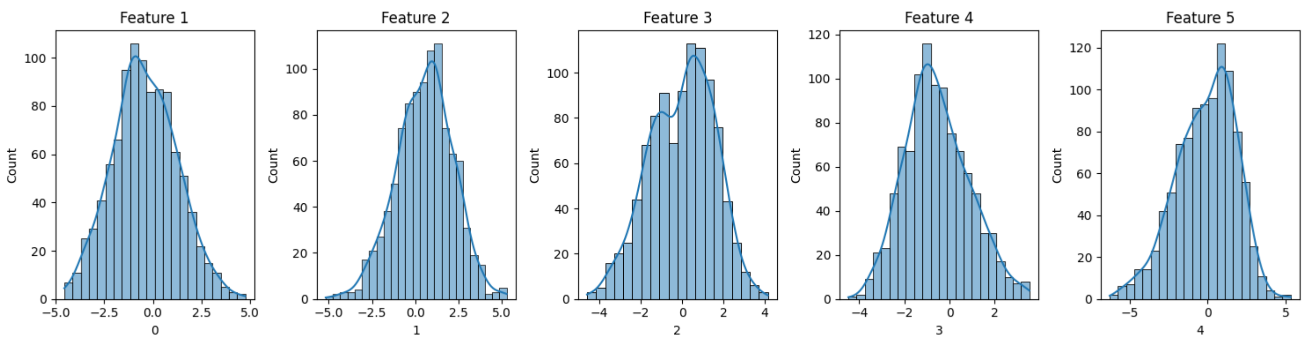
This project aimed to allow us to work on experiment with some data quality issues and to apply resolution techniques seen during the Data and Information Quality course. We were assigned to assess the performances of a classification task on a synthetically generated dataset, that we purposely infected to decrease the data quality, lowering the accuracy or injecting functional dependencies. Our objective was to analyse the performances, while training, of 20 different versions: we compared separately 10 polluted datasets with decreased accuracy, and another 10 datasets with injected functional dependencies.

2. Dataset generation and description

We used the provided scikit learn function *make_classification* to generate the dataset with the following parameters:

- n_samples = 1000
- n_features = 5
- n_informative = 5
- n_redundant = 0
- n_repeated = 0
- n_classes = 2
- n_clusters_per_class = 2
- weights = None
- flip_y = 0.01
- class_sep = 1.0
- hypercube = True
- seed = 2023

We obtained a dataset composed of 5 features (columns) and 1000 samples (rows). The following figures represent the value distributions of each column of the dataset.



The dataset is composed of all float64 values, with no other data type. Furthermore, we observed that the labels are of Boolean type, so the samples of the dataset can only belong to class 0 or 1.

From the data profiling analysis, we discovered that each column has no missing values, there are no infinites and there are no repeated values. Additionally, we noticed that each column has been generated sampling from a gaussian distribution with different mean and variance.

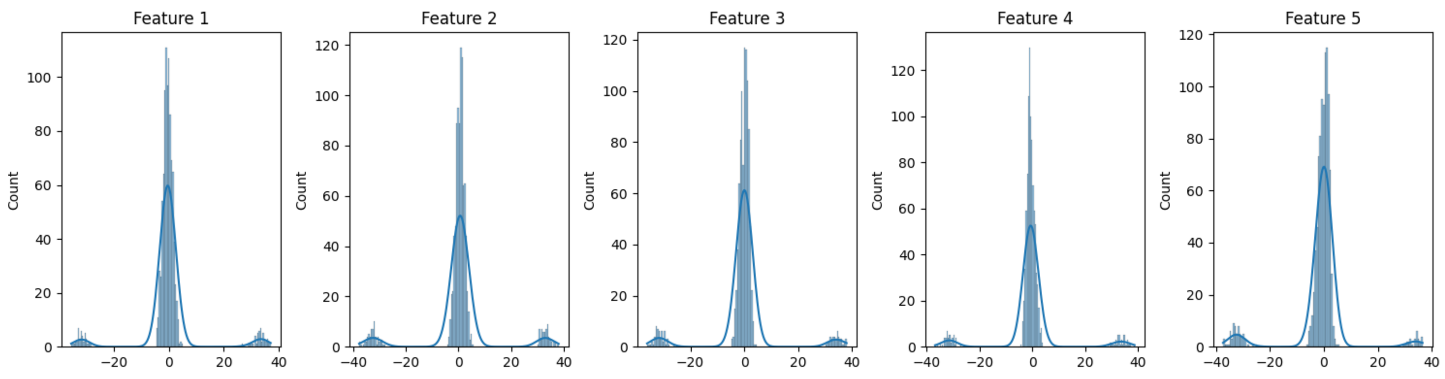
3. Accuracy dimension

3.1 Data pollution

We generated 10 different experiments, with decreasing levels of accuracy. To achieve such purpose, we injected an increasing number of outliers in the columns of the dataset: specifically, we created a mask on length 1000, with several zeros and ones according to the percentage of outliers required (parameter of the function). For the rows of the dataset corresponding to the cells of the mask with value 1, we modified the value of a randomly selected feature. We inserted there a value sampled from a normal distribution with a specific mean and variance, which are parameters of the pollution function. The mean value represents the distance between the outliers that we want to inject and the original set of points, while the variance represents how spread we want the outliers to be around the mean. The obtained datasets are a new version of the original one, generated with these parameters:

- Experiment 1: percentage=.05, mean=2, variance=0
- Experiment 2: percentage=.1, mean=2, variance=2
- Experiment 3: percentage=.15, mean=3, variance=0
- Experiment 4: percentage=.2, mean=3, variance=2
- Experiment 5: percentage=.25, mean=4, variance=0
- Experiment 6: percentage=.3, mean=4, variance=2
- Experiment 7: percentage=.35, mean=5, variance=0
- Experiment 8: percentage=.40, mean=5, variance=2
- Experiment 9: percentage=.45, mean=6, variance=0
- Experiment 10: percentage=.5, mean=6, variance=2

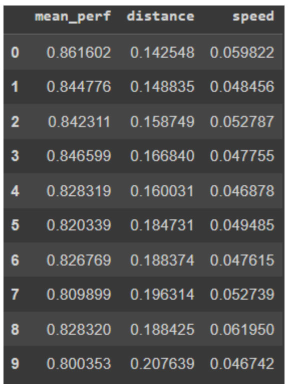
In the figure below it is shown an example of the features distributions in experiment 10. Notice how the pollution function modifies the distribution of the original dataset, adding outlier elements that do not respect the original structure of the dataset.



We decided to structure our 10 experiments with these parameters to cover the larger number of cases possible. We wanted to analyse the situation in which the outliers are close to the real data points, but also when they are far and, for this reason, they heavily modify the mean of the real features. The values of the variance have been chosen to consider two major cases: when the variance is set to 0 we are creating different clusters of outliers, while when the variance is set to 2 we are producing sparse outliers values. We decided to increase the number of outliers injected from 5% to 50% in steps of 5% to observe a linear degradation in performance of the classification algorithms in the different cases.

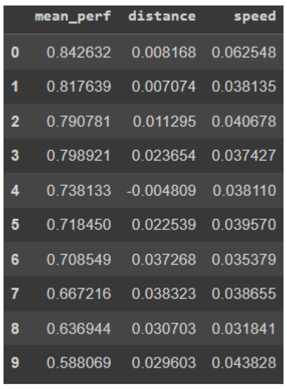
3.2 Classification analysis of the polluted experiments

We performed classification analysis on each experiment with 6 different algorithms, that are: DecisionTree, LogisticRegression, KNN, RandomForest, AdaBoost and finally MLP. These are the values for performance, distance between training and test errors and speed for each experiment and classification algorithm:



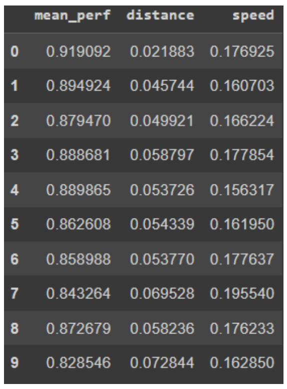
	mean_perf	distance	speed
0	0.861602	0.142548	0.059822
1	0.844776	0.148835	0.048456
2	0.842311	0.158749	0.052787
3	0.846599	0.166840	0.047755
4	0.828319	0.160031	0.046878
5	0.820339	0.184731	0.049485
6	0.826769	0.188374	0.047615
7	0.809899	0.196314	0.052739
8	0.828320	0.188425	0.061950
9	0.800353	0.207639	0.046742

Decision tree



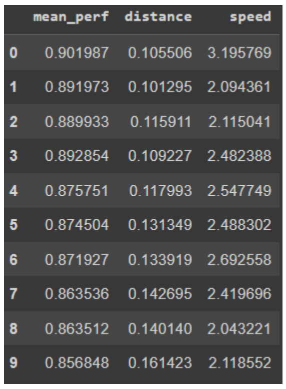
	mean_perf	distance	speed
0	0.842632	0.008168	0.062548
1	0.817639	0.007074	0.038135
2	0.790781	0.011295	0.040678
3	0.798921	0.023654	0.037427
4	0.738133	-0.004809	0.038110
5	0.718450	0.022539	0.039570
6	0.708549	0.037268	0.035379
7	0.667216	0.038323	0.038655
8	0.636944	0.030703	0.031841
9	0.588069	0.029603	0.043828

Logistic regression



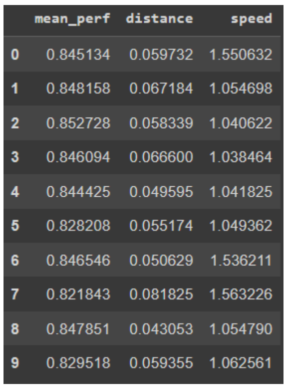
	mean_perf	distance	speed
0	0.919092	0.021883	0.176925
1	0.894924	0.045744	0.160703
2	0.879470	0.049921	0.166224
3	0.888681	0.058797	0.177854
4	0.889865	0.053726	0.156317
5	0.862608	0.054339	0.161950
6	0.858988	0.053770	0.177637
7	0.843264	0.069528	0.195540
8	0.872679	0.058236	0.176233
9	0.828546	0.072844	0.162850

KNN



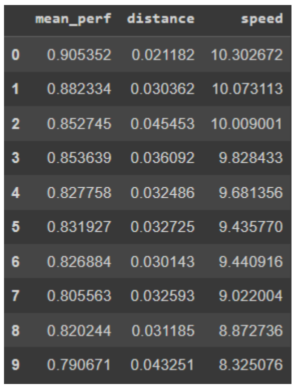
	mean_perf	distance	speed
0	0.901987	0.105506	3.195769
1	0.891973	0.101295	2.094361
2	0.889933	0.115911	2.115041
3	0.892854	0.109227	2.482388
4	0.875751	0.117993	2.547749
5	0.874504	0.131349	2.488302
6	0.871927	0.133919	2.692558
7	0.863536	0.142695	2.419696
8	0.863512	0.140140	2.043221
9	0.856848	0.161423	2.118552

Random forest



	mean_perf	distance	speed
0	0.845134	0.059732	1.550632
1	0.848158	0.067184	1.054698
2	0.852728	0.058339	1.040622
3	0.846094	0.066600	1.038464
4	0.844425	0.049595	1.041825
5	0.828208	0.055174	1.049362
6	0.846546	0.050629	1.536211
7	0.821843	0.081825	1.563226
8	0.847851	0.043053	1.054790
9	0.829518	0.059355	1.062561

AdaBoost



	mean_perf	distance	speed
0	0.905352	0.021182	10.302672
1	0.882334	0.030362	10.073113
2	0.852745	0.045453	10.009001
3	0.853639	0.036092	9.828433
4	0.827758	0.032486	9.681356
5	0.831927	0.032725	9.435770
6	0.826884	0.030143	9.440916
7	0.805563	0.032593	9.022004
8	0.820244	0.031185	8.872736
9	0.790671	0.043251	8.325076

MLP

From these values we can understand that the performances progressively worsen with the increasing of the percentage of outliers injected, just like the distance between train and test errors. The speed value strongly depends on the algorithm considered, even if we can see a trend for which a higher number of outliers implies a slightly faster execution. We think this happens because the algorithms converge to a worse value of the error and cannot further increase their performance, hence stopping prematurely.

3.3 Outlier detection and correction

For the matter of outlier detection, we implemented different methods from the literature. Later, we selected the one showing the most precise results considering the number of outliers expected from each experiment.

Firstly, we tried the Grubb’s test, the Mean Absolute Deviation, the Local Outlier Factor and KNN approaches, but we noticed that the choice of the parameters necessitates manual tuning for each column of each experiment due to the characteristics of each algorithm. For this reason, we discarded these options.

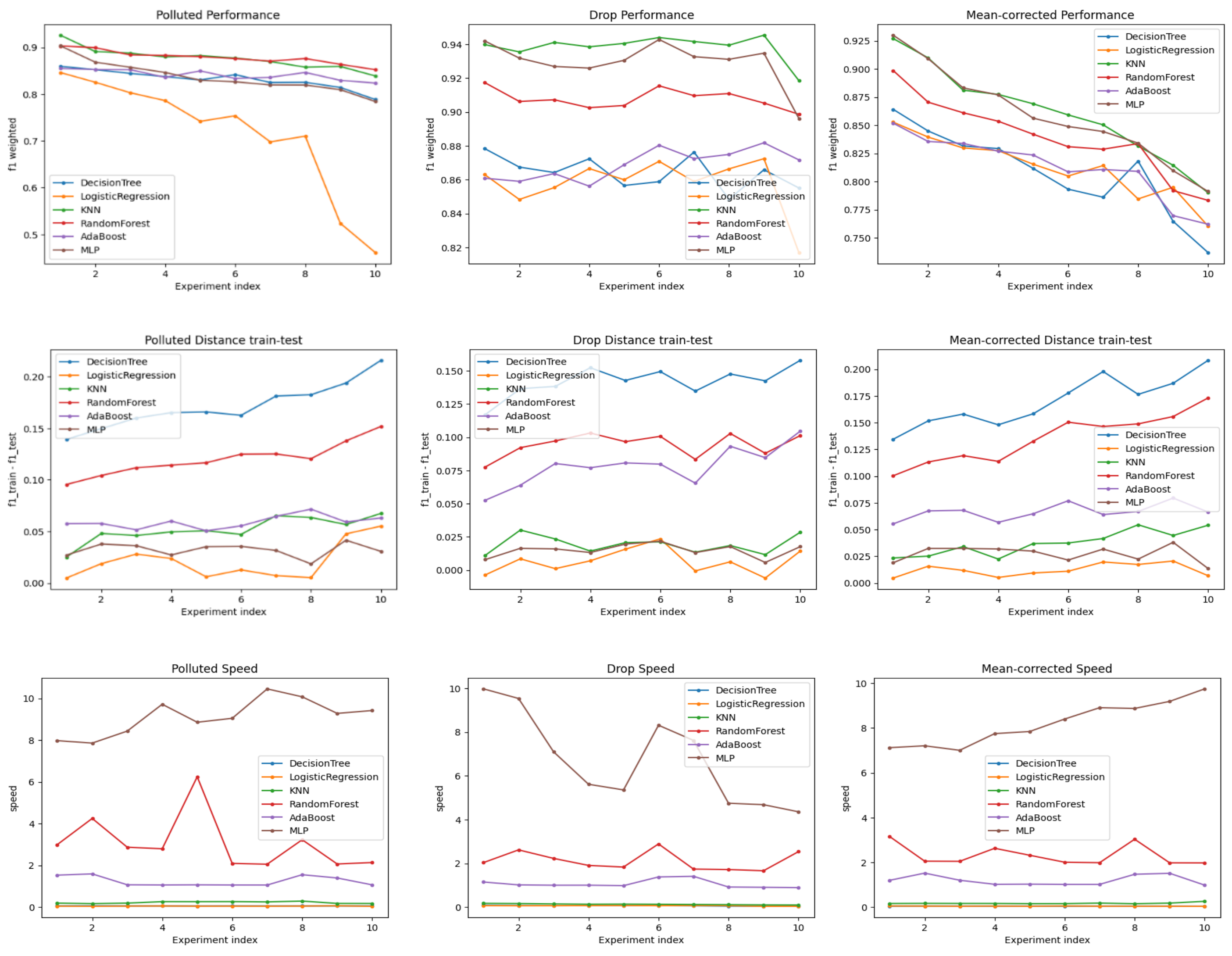
We finally chose the z-score statistical method, because it independently discovered a more reasonable number of outliers with respect to our expectations, without the need of fine tuning the parameters. It was necessary to identify a single value of the threshold to observe good results.

Once we selected all the outliers, we implemented two different approaches of dealing with them: dropping the entire row or substituting the wrong values with the mean of that feature (the mean has been computed not considering the outlier values), as they were the most reasonable options we had in this situation. Since the dataset is synthetically generated and the values have no real meaning, we could not consider applying other correction methods, thus we had no way to infer a possible relation between the rows, or columns, of the dataset.

After the handling of the detected outliers, we performed the classification steps exactly as previously done, but this time considering the corrected datasets. The first classification considered the dataset in which we dropped the outliers, while the second considered the dataset with substituted values.

3.4 Classification results comparison

As last step of the pipeline we proceeded by comparing the obtained results in the classification of the different sets of experiments (polluted experiments, experiments with dropped outliers and experiments with mean-corrected outliers).



For what concerns the performances, we can clearly see that, as expected, the generalization capabilities on the polluted dataset degrade when the number of outliers increases. This is a logical consequence because each algorithm will have to work on wrong and insignificant data. When one correction is applied, we would expect to observe better trends, but this happens only when the outlier rows are dropped from the dataset. We can in fact state that each algorithm improves its performances, some more evidently than others (e.g.: logistic regression). We can reason on this by saying that removing erroneous data makes the algorithm work on fewer but more meaningful samples. On the other hand, we observe a less satisfying result for what concerns the second correction method (mean-corrected). Even if some algorithms improved (e.g.: logistic regression), we expected to see a trend of improvement with respect to the polluted dataset and something comparable, if not even better, in relation with the previous correction approach. This may be caused by the fact that the datasets distributions are heavily modified due to the high presence of new samples on the mean.

Considering the train-test error distance, we observe that, as expected, it generally increases with the number of outliers present in the dataset, showing a degradation of the performances. Correcting by dropping outliers, we notice a nearly constant error, symptom of the fact that we detected in a right way and then eliminated all the outliers. By correcting with the mean, we observe trends similar to the polluted datasets, since we are just substituting the wrong values with approximated ones.

Given the fact that the speed is a property of the algorithm, we can state that reducing the size of the dataset may help to decrease the computation time, while correcting but not modifying the number of samples doesn't significantly improve the initial condition.

We can overall state that dropping the outliers can be considered the best approach between the two evaluated in this analysis.

4. Functional dependencies

4.1 Data pollution

Starting from the dataset that we generated like explained in section 2, we decided to pollute it by injecting different number of new features that are dependent on the already existing ones.

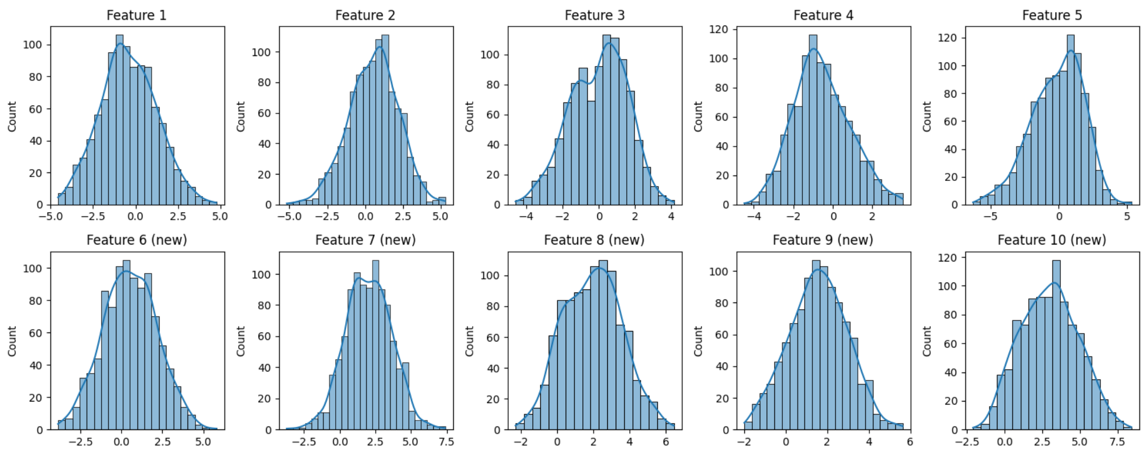
We aimed at creating different experiments, each one adding a new feature correlated to a specified existing one with a given Pearson correlation coefficient. For this purpose, we also introduced a random error to the new values. For instance, we may have wanted to add to the original dataset one feature correlated to feature 3 with a level of correlation 0.8 and a random error of 6. To inject more than one feature, the pollution function is iterated over the modified dataset.

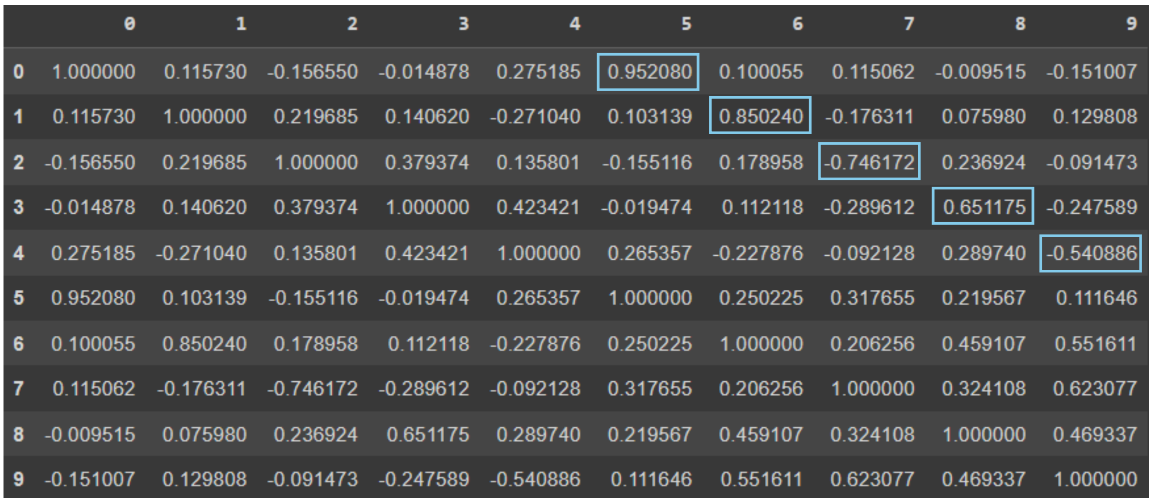
The pollution function takes as inputs a dataset, the feature for which we want to generate the dependent samples, the Pearson's correlation coefficient and the error we want to inject.

The experiments we built have the following parameters:

- Experiment 1: dataset=X, feature=0, coeff=-1, error=5.8
- Experiment 2: dataset=dataset_of__exp_1, feature=1, coeff=.9, error=5.8
- Experiment 3: dataset= dataset_of__exp_2, feature=2, coeff=-.8, error=5.8
- Experiment 4: dataset= dataset_of__exp_3, feature=3, coeff=-.7, error=5
- Experiment 5: dataset= dataset_of__exp_4, feature=4, coeff=-.6, error=6.9
- Experiment 6: dataset=X, feature=0, coeff=.95, error=5.8
- Experiment 7: dataset= dataset_of__exp_6, feature=1, coeff=.85, error=5.8
- Experiment 8: dataset= dataset_of__exp_7, feature=2, coeff=-.75, error=5.8
- Experiment 9: dataset= dataset_of__exp_8, feature=3, coeff=.65, error=5
- Experiment 10: dataset= dataset_of__exp_9, feature=4, coeff=-.55, error=6.9

Below are reported the features of experiment 9, which added 5 new features to the initial dataset X, each one correlated to one original feature (in the figure each column contains the original feature and the correlated one).





	0	1	2	3	4	5	6	7	8	9
0	1.000000	0.115730	-0.156550	-0.014878	0.275185	0.952080	0.100055	0.115062	-0.009515	-0.151007
1	0.115730	1.000000	0.219685	0.140620	-0.271040	0.103139	0.850240	-0.176311	0.075980	0.129808
2	-0.156550	0.219685	1.000000	0.379374	0.135801	-0.155116	0.178958	-0.746172	0.236924	-0.091473
3	-0.014878	0.140620	0.379374	1.000000	0.423421	-0.019474	0.112118	-0.289612	0.651175	-0.247589
4	0.275185	-0.271040	0.135801	0.423421	1.000000	0.265357	-0.227876	-0.092128	0.289740	-0.540886
5	0.952080	0.103139	-0.155116	-0.019474	0.265357	1.000000	0.250225	0.317655	0.219567	0.111646
6	0.100055	0.850240	0.178958	0.112118	-0.227876	0.250225	1.000000	0.206256	0.459107	0.551611
7	0.115062	-0.176311	-0.746172	-0.289612	-0.092128	0.317655	0.206256	1.000000	0.324108	0.623077
8	-0.009515	0.075980	0.236924	0.651175	0.289740	0.219567	0.459107	0.324108	1.000000	0.469337
9	-0.151007	0.129808	-0.091473	-0.247589	-0.540886	0.111646	0.551611	0.623077	0.469337	1.000000

Correlation table of experiment 10 (with Pearson's coefficient)

4.2 Functional dependencies detection

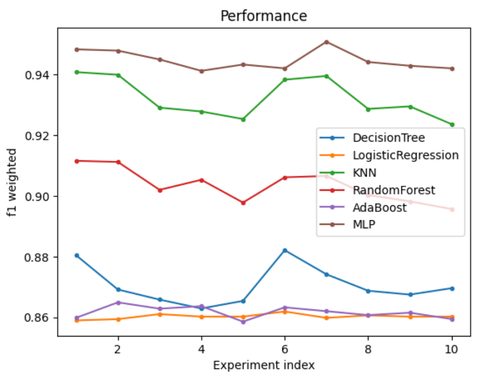
After having created the 10 experiments by polluting the original dataset as aforementioned, for each of them we performed an analysis with two different automatic tools to detect the functional dependencies. More specifically, we employed the *tane* and the *fd_mine* algorithms.

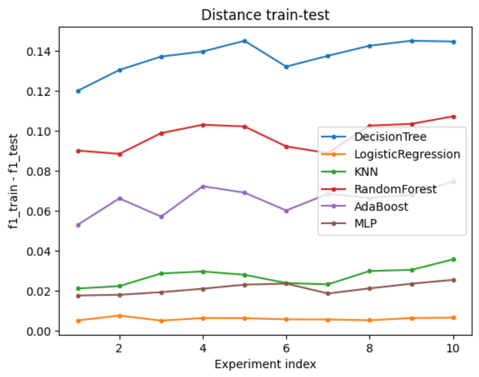
The *tane* algorithm is a top-down algorithm that explores the search space of attribute sets to find all non-trivial functional dependencies in a given relation. It also utilizes a pruning strategy to reduce the search space and improve efficiency. *Fd_mine* is an evolution of *tane* as it exploits the property of the functional dependencies in order to further prune the lattice.

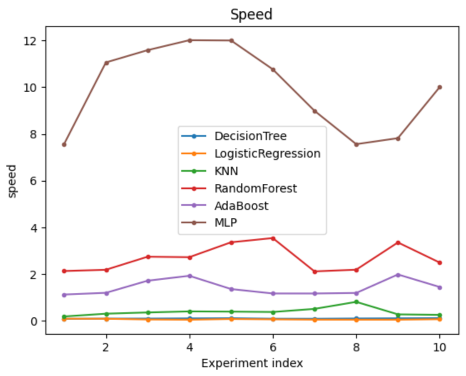
From both these analyses we observed that the introduced dependencies are indeed correctly detected, and other unexpected dependencies between the original features are highlighted, meaning that there might be some correlation even in the creation of the synthetic dataset.

4.3 Classification analysis of polluted experiments

We executed the classification with the same 6 algorithms as before (DecisionTree, LogisticRegression, KNN, RandomForest, AdaBoost and finally MLP) and the following graphs show their results:





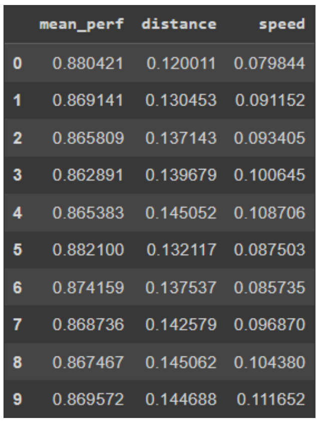


For what regards the performances, we can see that the general trend is slightly affected by the number of new injected features. In fact, we can clearly see that the experiments having more dependent features tend to have lower general performance across all the algorithms. This may be caused by the fact that the new features are not completely correlated, thus introducing a degree of error degrades the performances.

Similar considerations can be applied to the train-test error distance, even if with a smaller scale. The degradation is less noticeable.

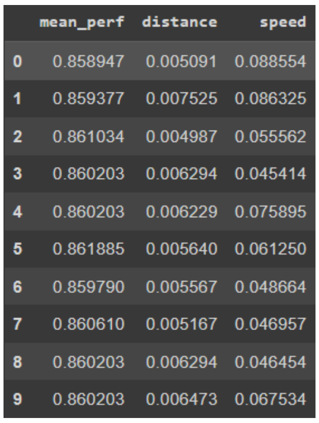
Since the behaviour of the speed curves for the different algorithms do not converge, we cannot infer general reasoning by looking at these graphs.

The numeric values represented in the graphs are reported in the tables below.



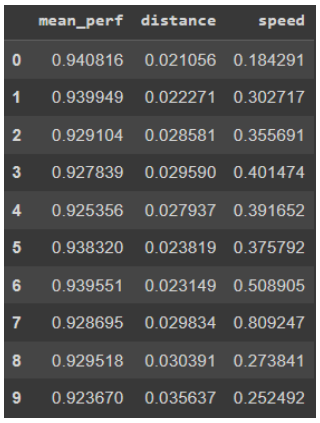
	mean_perf	distance	speed
0	0.880421	0.120011	0.079844
1	0.869141	0.130453	0.091152
2	0.865809	0.137143	0.093405
3	0.862891	0.139679	0.100645
4	0.865383	0.145052	0.108706
5	0.882100	0.132117	0.087503
6	0.874159	0.137537	0.085735
7	0.868736	0.142579	0.096870
8	0.867467	0.145062	0.104380
9	0.869572	0.144688	0.111652

Decision tree



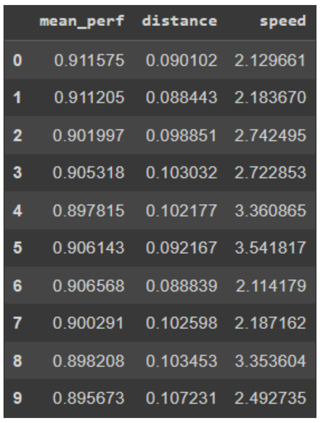
	mean_perf	distance	speed
0	0.858947	0.005091	0.088554
1	0.859377	0.007525	0.086325
2	0.861034	0.004987	0.055562
3	0.860203	0.006294	0.045414
4	0.860203	0.006229	0.075895
5	0.861885	0.005640	0.061250
6	0.859790	0.005567	0.048664
7	0.860610	0.005167	0.046957
8	0.860203	0.006294	0.046454
9	0.860203	0.006473	0.067534

Logistic regression



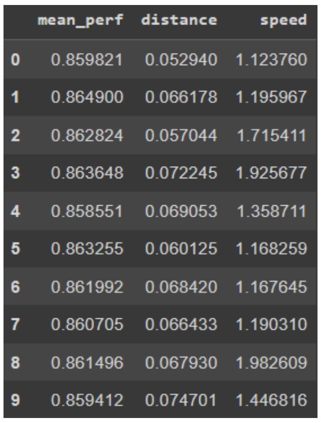
	mean_perf	distance	speed
0	0.940816	0.021056	0.184291
1	0.939949	0.022271	0.302717
2	0.929104	0.028581	0.355691
3	0.927839	0.029590	0.401474
4	0.925356	0.027937	0.391652
5	0.938320	0.023819	0.375792
6	0.939551	0.023149	0.508905
7	0.928695	0.029834	0.809247
8	0.929518	0.030391	0.273841
9	0.923670	0.035637	0.252492

KNN



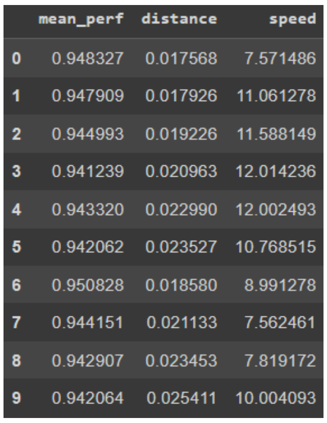
	mean_perf	distance	speed
0	0.911575	0.090102	2.129661
1	0.911205	0.088443	2.183670
2	0.901997	0.098851	2.742495
3	0.905318	0.103032	2.722853
4	0.897815	0.102177	3.360865
5	0.906143	0.092167	3.541817
6	0.906568	0.088839	2.114179
7	0.900291	0.102598	2.187162
8	0.898208	0.103453	3.353604
9	0.895673	0.107231	2.492735

Random forest



	mean_perf	distance	speed
0	0.859821	0.052940	1.123760
1	0.864900	0.066178	1.195967
2	0.862824	0.057044	1.715411
3	0.863648	0.072245	1.925677
4	0.858551	0.069053	1.358711
5	0.863255	0.060125	1.168259
6	0.861992	0.068420	1.167645
7	0.860705	0.066433	1.190310
8	0.861496	0.067930	1.982609
9	0.859412	0.074701	1.446816

AdaBoost



	mean_perf	distance	speed
0	0.948327	0.017568	7.571486
1	0.947909	0.017926	11.061278
2	0.944993	0.019226	11.588149
3	0.941239	0.020963	12.014236
4	0.943320	0.022990	12.002493
5	0.942062	0.023527	10.768515
6	0.950828	0.018580	8.991278
7	0.944151	0.021133	7.562461
8	0.942907	0.023453	7.819172
9	0.942064	0.025411	10.004093

MLP

5. Conclusions

Thanks to this project we had the opportunity to work hands-on with some of the issues and handling techniques presented in the Data and Information Quality course. This has been really stimulating also because we had to look at the problematics from a new perspective: creating data quality issues allowed us to deeply reason on their meaning and consequences on the typical machine learning tasks.