DATA QUALITY PROJECT REPORT

PROJECT ID 19 - DUPLICATION and DIMENSIONALITY issues - CLASSIFICATION task

Simone Garbazio, 10666868 - Francesco Pastori, 10628982

---

1. Data Duplication

Our first data quality issue to address is data duplication. So, we inject into our dataset some duplicated rows which are more or less similar to other rows. Then, we analyze the effect that these duplicates have on some classification algorithms (Decision Tree, Logistic Regression, KNN, Random Forest, AdaBoost and MLP).

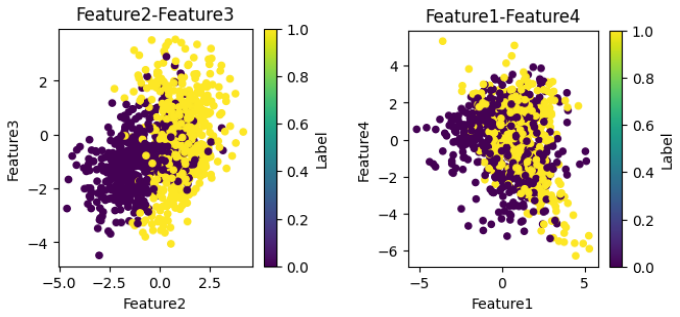
1.1. Setup choices

1.1.1. Data collection

The initial version of our dataset (the "clean" one) is created from the function *make_classification* contained in the *sklearn* package of Python for both the Data Quality issues that we have addressed. We create a dataset with 1000 samples and 5 features. All the features are *informative*, so we don't have *redundant* or *repeated* features. Furthermore, the generated points belong to 2 classes. The other parameters are basically the default ones.

Since the dataset is synthetic, we approximately know its characteristics. Therefore, we have not done a very in-depth exploratory phase. Some summary tables generated from the dataset just confirmed that we have a dataset balanced in the two classes, with floating point values that in magnitude do not exceed 10.

We also perform *scatter-plots* to visualize the dataset, by picking two features at a time. Some plots show a clear separation between the points of the two classes, while others show an overlap.



1.1.2. Experiments

As previously stated, our aim is to create some duplicated rows and add them to the initial dataset in order to pollute it. For all the experiments, the pollution is done on the same clean dataset. The choice of having only one "original" dataset allows us to compare the experiments with each other because they all have the same starting point. Thanks to this choice, we can observe how different algorithms behave, providing the same data but varying the number and level of similarity of the duplicates.

So, from the "original" dataset, we generate the 10 experiments:

- Each experiment has a fixed percentage of similarity (between the "original" rows and the duplicated ones).
    - The similarity is based on the mean euclidean distance between all points in the dataset. Thus, two points are 90% similar if the distance between them is 10% of the mean distance.

    - After some tests we have noticed that going below 80% similarity made little sense, so we vary this value from 80 to 98, in steps of 2.
- Inside each experiment, we generate 10 different datasets varying the percentage of duplicated rows.
    - Here instead we vary the value from 10% to 100% in steps of 10.

For example, an experiment with 90% of similarity has a dataset with 1000 samples plus 10% of duplicated rows (so, 100 rows) which are 90% similar to the "original" rows, then another dataset with 1000 samples plus 15% of duplicated rows that are still 90% similar to the "original" rows, etc.

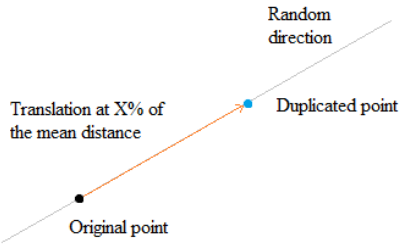
1.1.3. Data Pollution

The generation of the duplicates is divided into two phases: a first phase in which we select a set of candidate points to be duplicated and a second phase where we change these points based on the similarity measure.

For the selection of the rows to be duplicated, we preferred to create multiple duplicates of a small number of rows rather than a single duplicate for a large number of rows. So, once the total number of duplicates (a percentage of rows of the dataset) has been calculated, we compute the 5% of this quantity and we randomly pick this small set of "original" rows from the dataset. Then, we sample with replacement from this set of rows until we build a set with as many rows as we want.

So, for example, if a dataset with 1000 samples has to be polluted with the 10% of duplicates, the total number of duplicated rows is 100 but these rows are generated from a sample of 5 "original" rows. Then, to fill the 100 duplicates, the first original row is duplicated 11 times, the second is duplicated 8 times, and so on.

In the second step, we calculate the distance where the candidate vectors need to be moved, based on the similarity and the mean euclidean distance. So, we generate a random direction and the point is translated along this direction by the calculated distance. Once the operation has been performed on all points, the duplicates are added to the dataset.



Finally, we would like to point out that no disturbance has been made on the labels: the duplicates are assigned the same label as the sample from which they derive.

1.2 Pipeline implementation

The pipeline used to address the problem consists of:

1. Build of reference dataset.
2. Exploration of the dataset.
3. Creation of polluted datasets.
4. Establish the classification algorithms to be applied, we used:
    a. Decision Tree
    b. Logistic Regression
    c. K-Nearest Neighbors (KNN)
    d. Random Forest
    e. AdaBoost
    f. Multi Layer Perceptron (MLP)
5. Perform classification algorithms.
6. Evaluate classification results through metrics, we used:

a. Weighted F1 (performance): F1 score is a metric that combines precision and recall, providing a balance between false positives and false negatives. It is computed through cross-validation.
b. Distance between training and test set (distance): performs multiple iterations of train-test splits and computes the f1 score differences across these splits, then takes the mean of these differences.
c. Execution time (speed) of each algorithm.

7. Print the results on graphs and tables. Then store the values in csv files for possible future use.
8. Data preparation: remove duplicates with Record Linkage.
9. Perform classification, evaluation and visualization again for these new results.

1.2.2 Record linkage

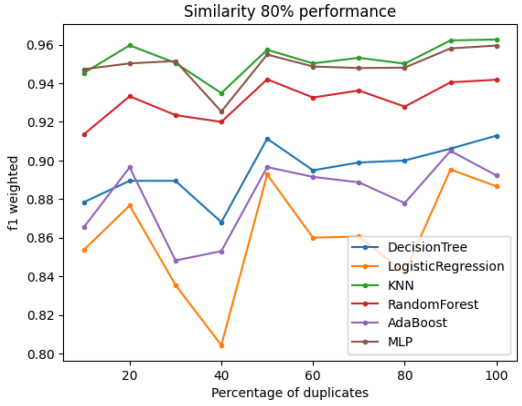
To identify duplicate records in our polluted datasets we use *Record Linkage*. It compares the features of each record between them and calculates a similarity score, ranging from 0 to 1. The higher the sum of scores in a row, the more likely it is a duplicate. After some experiments, we decided to put a *threshold* that is computed as 95% of the number of features (in our case 4.75): if the sum of the scores is above this threshold, the row is dropped from the dataset. Record linkage also uses several methods to compare rows, but we found that the default *linear* metric gave us the best performance.
To assess Record Linkage performance on our polluted datasets, we compute and print accuracy, precision and recall. As expected, Record Linkage has good performances on datasets where duplicates have a high similarity with the original rows while it struggles to find them when the similarity drops below the 90%. Some results are reported in the table below.

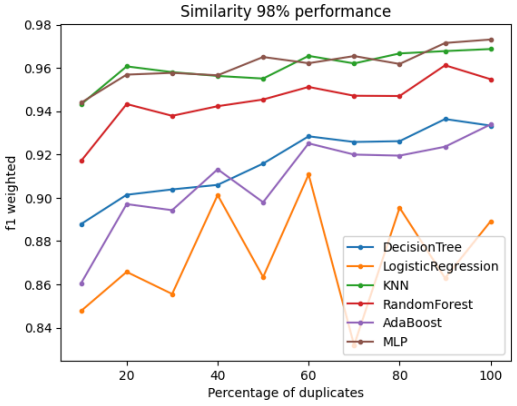
	80% Similarity 100% Duplicates	90% Similarity 100% Duplicates	98% Similarity 100% Duplicates
Accuracy	0.50	0.58	0.99
Precision	0.55	0.95	0.99
Recall	0.01	0.17	0.99

1.3 Results: data evaluation and comparison

1.3.1. Before data preparation

The most interesting result that we have observed is that, in general, the more duplicates, the better performance. This is more evident when the similarity between “original” points and duplicates increases. It can be seen in the plots below, followed by the numerical values of the performance with 98% similarity.

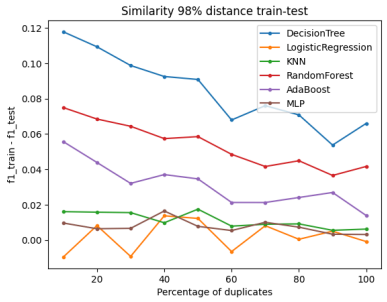




	10	20	30	40	50	60	70	80	90	100
DecisionTree	0.887873	0.901357	0.903847	0.905933	0.915791	0.928380	0.925768	0.926138	0.936361	0.933312
LogisticRegression	0.847727	0.865727	0.855537	0.901236	0.863361	0.910740	0.831840	0.895374	0.862846	0.889101
KNN	0.943183	0.960693	0.958039	0.956276	0.955068	0.965542	0.962057	0.966676	0.967758	0.968711
RandomForest	0.917046	0.943291	0.937852	0.942294	0.945388	0.951201	0.947116	0.947001	0.961202	0.954762
AdaBoost	0.860557	0.897058	0.894242	0.913141	0.897910	0.925122	0.919936	0.919455	0.923573	0.933872
MLP	0.943940	0.956883	0.957698	0.956573	0.965027	0.962160	0.965468	0.961810	0.971508	0.973107

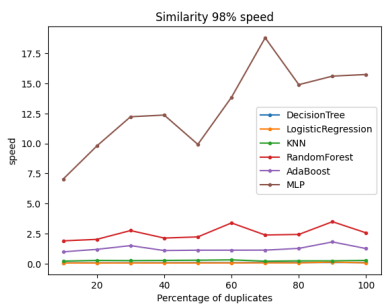
This result initially has surprised us because we expected the performance to decrease with more duplicates present in the dataset. For this reason, we also tried different approaches in the generation of duplicates, like adding more of them or by duplicating even less original rows multiple times or even by making duplicates for each row, but the results didn't change.
At this point, our hypothesis is that duplicates (especially if very similar to the original data) cause the models to overfit. We see an improvement in the performance because the metrics derive from a cross-validation that is carried out on the polluted dataset. Therefore, the various test sets also contain duplicates. As these models overfit, they display better performance on biased data while losing the ability to generalize. We are quite certain that performance will degrade when these models are tested on new data.

Decision Trees proved to have the biggest distance between performance on the test set and on the train set (although not excessively high compared to the others). This can be due to the fact that Decision Trees are prone to overfitting (if not combined with techniques like pruning) since they tend to grow very deep and complex, capturing every possible detail from the training data.



	10	20	30	40	50	60	70	80	90	100
DecisionTree	0.117825	0.109353	0.098767	0.092544	0.090864	0.067958	0.076052	0.070832	0.053766	0.066036
LogisticRegression	-0.009542	0.008247	-0.009201	0.013783	0.012287	-0.006444	0.008169	0.000496	0.005082	-0.000791
KNN	0.016118	0.015818	0.015573	0.009836	0.017521	0.007923	0.008994	0.009201	0.005562	0.006268
RandomForest	0.074976	0.068488	0.064403	0.057421	0.058546	0.048528	0.041653	0.044905	0.036597	0.041686
AdaBoost	0.055711	0.043937	0.032108	0.037081	0.034656	0.021318	0.021298	0.024125	0.026948	0.013913
MLP	0.009681	0.006452	0.006675	0.016501	0.007801	0.005439	0.010116	0.007342	0.003353	0.003196

MLP is much slower than other algorithms. This is not surprising since MLP is a Neural Network, thus it has more parameters to train and optimize.
From this point of view, it can be seen the efficiency of KNN, that provides performances that are similar to the ones of MLP while being one of the fastest algorithms. Again, this is not surprising since KNN doesn't even need a proper training (it just needs to compute distances) and we are in a binary classification problem where we have seen that, at least when looking at some features, the points belonging to different classes are well separated.

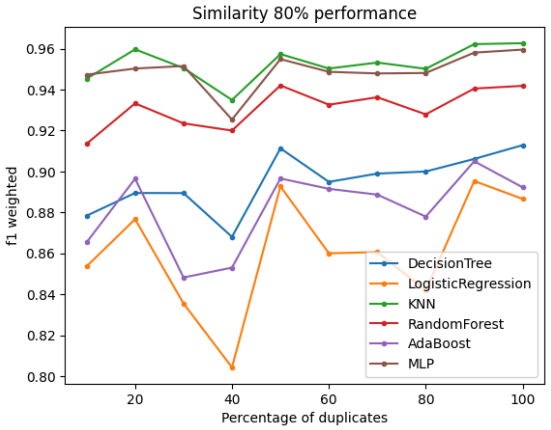


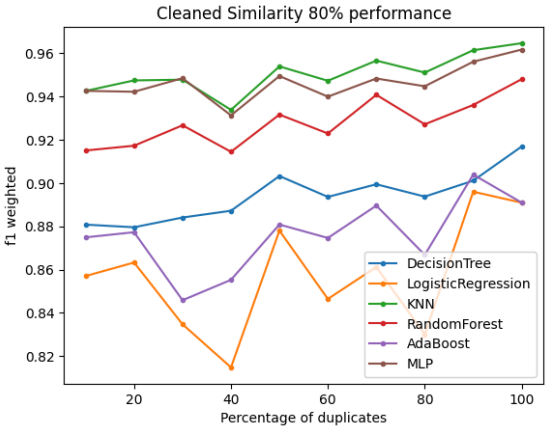
	10	20	30	40	50	60	70	80	90	100
DecisionTree	0.070617	0.070016	0.069084	0.076603	0.081351	0.075439	0.089797	0.087624	0.091408	0.087309
LogisticRegression	0.051125	0.048508	0.052311	0.050418	0.054079	0.057385	0.061921	0.059807	0.125398	0.060007
KNN	0.209183	0.265317	0.249666	0.265085	0.282640	0.311355	0.203198	0.229778	0.228137	0.267686
RandomForest	1.892421	2.020030	2.750790	2.127988	2.225220	3.386416	2.381031	2.428610	3.488441	2.559158
AdaBoost	0.982105	1.186405	1.497178	1.088290	1.116446	1.118149	1.122933	1.266991	1.804449	1.252150
MLP	7.038570	9.793984	12.224689	12.369180	9.926451	13.847855	18.798962	14.896989	15.606240	15.744950

1.3.2. After data preparation

Given the observations made above, we were not surprised to see a substantial lack of performance improvements in clean datasets. What can be interesting here is to visualize the difference between before and after data preparation, based on the percentage of similarity. In fact, Record Linkage finds (and therefore eliminates) duplicates only if the percentage of similarity is very high. Therefore, in datasets with low similarity we will practically not notice differences between before and after. Instead, in datasets with high similarity, we will see a "flattening" of performance, due to the fact that the 10 datasets with different percentages of duplicates become identical since all the duplicates are removed. Some examples are reported below.

Dataset with 80% similarity before and after: basically no difference.

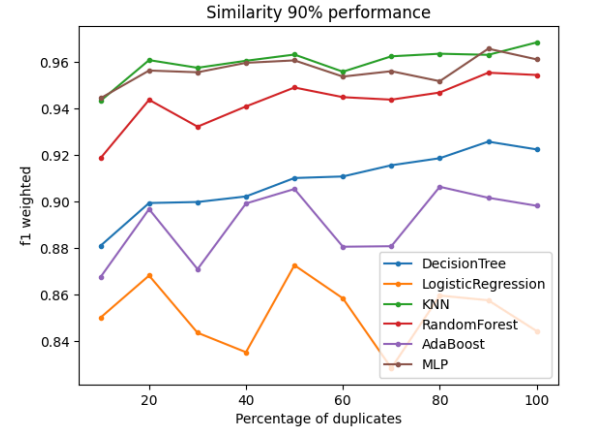


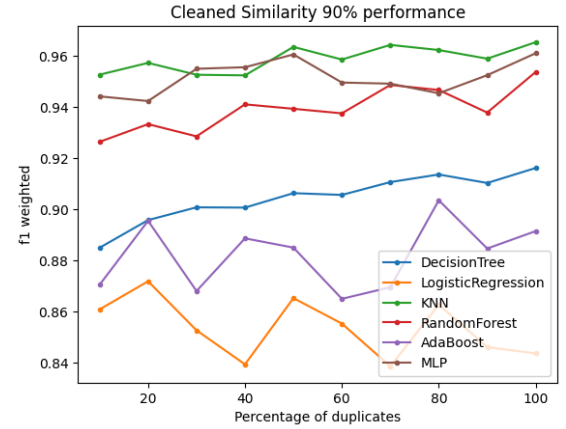


	10	20	30	40	50	60	70	80	90	100
DecisionTree	0.878399	0.889506	0.889446	0.868071	0.911317	0.894929	0.898973	0.900004	0.906139	0.912889
LogisticRegression	0.853649	0.876692	0.835516	0.804278	0.892730	0.860036	0.860600	0.841856	0.895337	0.886624
KNN	0.945369	0.959695	0.950550	0.934948	0.957386	0.950322	0.953237	0.950181	0.962242	0.962701
RandomForest	0.913563	0.933287	0.923522	0.920010	0.942062	0.932648	0.936282	0.927917	0.940531	0.941861
AdaBoost	0.865411	0.896476	0.848207	0.853025	0.896593	0.891533	0.888695	0.877963	0.904988	0.892257
MLP	0.947317	0.950325	0.951543	0.925387	0.954911	0.948712	0.947926	0.948143	0.958099	0.959574

	10	20	30	40	50	60	70	80	90	100
DecisionTree	0.880769	0.879547	0.884054	0.887181	0.903223	0.893584	0.899418	0.893722	0.901087	0.916876
LogisticRegression	0.857010	0.863256	0.834696	0.814775	0.877993	0.846414	0.861177	0.830052	0.895981	0.890918
KNN	0.942567	0.947432	0.947777	0.933812	0.953905	0.947268	0.956578	0.951053	0.961398	0.964632
RandomForest	0.915040	0.917242	0.926687	0.914431	0.931628	0.922903	0.940770	0.927139	0.936082	0.948002
AdaBoost	0.874890	0.877260	0.845827	0.855236	0.880809	0.874598	0.889562	0.866791	0.903894	0.890924
MLP	0.942604	0.942184	0.948453	0.931268	0.949445	0.939922	0.948313	0.944726	0.956077	0.961683

Dataset with 90% similarity before and after: still no significant difference.

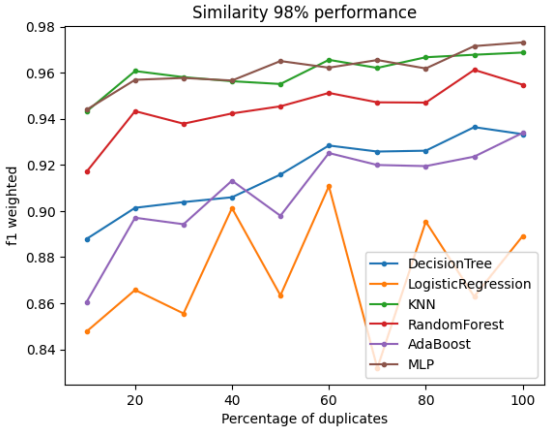


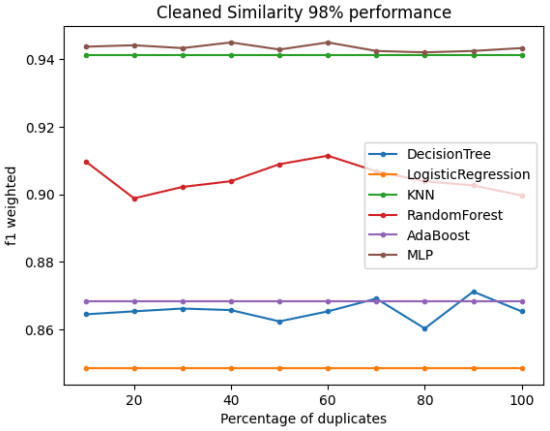


	10	20	30	40	50	60	70	80	90	100
DecisionTree	0.887873	0.901357	0.903847	0.905933	0.915791	0.928380	0.925768	0.926138	0.936361	0.933312
LogisticRegression	0.847727	0.865727	0.855537	0.901236	0.863361	0.910740	0.831840	0.895374	0.862846	0.889101
KNN	0.943183	0.960693	0.958039	0.956276	0.955068	0.965542	0.962057	0.966676	0.967758	0.968711
RandomForest	0.917046	0.943291	0.937852	0.942294	0.945388	0.951201	0.947116	0.947001	0.961202	0.954762
AdaBoost	0.860557	0.897058	0.894242	0.913141	0.897910	0.925122	0.919936	0.919455	0.923573	0.933872
MLP	0.943940	0.956883	0.957698	0.956573	0.965027	0.962160	0.965468	0.961810	0.971508	0.973107

	10	20	30	40	50	60	70	80	90	100
DecisionTree	0.884906	0.895704	0.900707	0.900591	0.906238	0.905552	0.910566	0.913539	0.910217	0.916076
LogisticRegression	0.860798	0.871785	0.852613	0.839311	0.865193	0.855249	0.838510	0.862654	0.846042	0.843581
KNN	0.952569	0.957170	0.952533	0.952301	0.963426	0.958489	0.964211	0.962196	0.958823	0.965263
RandomForest	0.926315	0.933225	0.928451	0.940960	0.939230	0.937450	0.948545	0.946571	0.937757	0.953616
AdaBoost	0.870444	0.895428	0.867971	0.888578	0.884952	0.864929	0.869449	0.903453	0.884592	0.891396
MLP	0.944078	0.942252	0.954870	0.955473	0.960482	0.949489	0.949052	0.945304	0.952363	0.960944

Dataset with 98% similarity before and after: performance flattened.





	10	20	30	40	50	60	70	80	90	100
DecisionTree	0.880910	0.899245	0.899681	0.902077	0.909990	0.910657	0.915476	0.918508	0.925627	0.922287
LogisticRegression	0.850034	0.868068	0.843475	0.835124	0.872496	0.858269	0.828294	0.859465	0.857445	0.844188
KNN	0.943205	0.960677	0.957379	0.960388	0.963058	0.955687	0.962303	0.963417	0.962937	0.968278
RandomForest	0.918589	0.943602	0.932069	0.940758	0.948892	0.944747	0.943682	0.946746	0.955256	0.954281
AdaBoost	0.867476	0.896627	0.870818	0.899048	0.905276	0.880453	0.880679	0.906231	0.901510	0.898059
MLP	0.944318	0.956198	0.955449	0.959509	0.960556	0.953593	0.955910	0.951608	0.965572	0.960993

	10	20	30	40	50	60	70	80	90	100
DecisionTree	0.864495	0.865381	0.866201	0.865748	0.862401	0.865365	0.869149	0.860310	0.871218	0.865351
LogisticRegression	0.848573	0.848573	0.848573	0.848573	0.848573	0.848573	0.848573	0.848573	0.848573	0.848573
KNN	0.941225	0.941225	0.941225	0.941225	0.941225	0.941225	0.941225	0.941225	0.941225	0.941225
RandomForest	0.909783	0.898850	0.902226	0.903921	0.908959	0.911479	0.906836	0.903884	0.902664	0.899688
AdaBoost	0.868295	0.868295	0.868295	0.868295	0.868295	0.868295	0.868295	0.868295	0.868295	0.868295
MLP	0.943761	0.944184	0.943340	0.945024	0.942918	0.945024	0.942490	0.942080	0.942498	0.943351

2. Data Dimensionality

Our second data quality issue to address is data dimensionality. The dimensionality problem in the context of data quality refers to the challenge posed by datasets that have a large number of features compared to the number of observations, or vice versa. So, we create datasets with varying numbers of features and samples. Then, we analyze the effect that these variations have on some classification algorithms (Decision Tree, Logistic Regression, KNN, Random Forest, AdaBoost and MLP).

2.1 Setup choices

2.1.1 Experiments

To show the dimensionality problem we created 10 different experiments, each with a fixed number of samples (i.e. the first experiment 1000 samples, the second experiment 2000, ...) and for each experiment we created 10 different datasets by varying for each dataset the number of features incrementally, starting from 5 up to 50 features. Thus being able to observe in each experiment how the different classification algorithms behave going from balanced to high dimensionality.
For completeness, we also approached the problem from the opposite point of view by creating 10 more different experiments, each with fixed number of features (i.e. the first experiment 5 features, the second experiment 10 features, ...) and for each experiment we created 10 different datasets, varying for each dataset the number of samples, starting from 1000 up to 10'000 samples. In this way, we are able to observe in each experiment how the classification algorithms behave by increasing the number of samples. Usually this scenario is not considered a dimensionality problem in the negative sense that high dimensionality often brings. Instead, it is a favorable scenario that can contribute to more stable and generalizable machine learning models.

2.1.2 Data collection and pollution

The collection and pollution phases have been implemented together given that the pollution phase is contemporary with that of creating a dataset, in accordance with what was said in the previous paragraph. The various datasets are created from the function *make_classification* contained in the *sklearn* package of Python that we have described in the first data quality issue chapter. The parameters *n_features* and *n_samples* have been modified for each experiment to create datasets with the desired shapes.

2.2 Pipeline implementation

The pipeline used to address the problem consists of:

1. Build the datasets which, in this case, is already "polluted" because they are constructed by varying number of features and number of samples.
2. Establish the classification algorithms to be applied, we used:
    g. Decision Tree
    h. Logistic Regression
    i. K-Nearest Neighbors (KNN)
    j. Random Forest
    k. AdaBoost
    l. Multi Layer Perceptron (MLP)
3. Perform classification algorithms.
4. Evaluate classification results through metrics, we used:

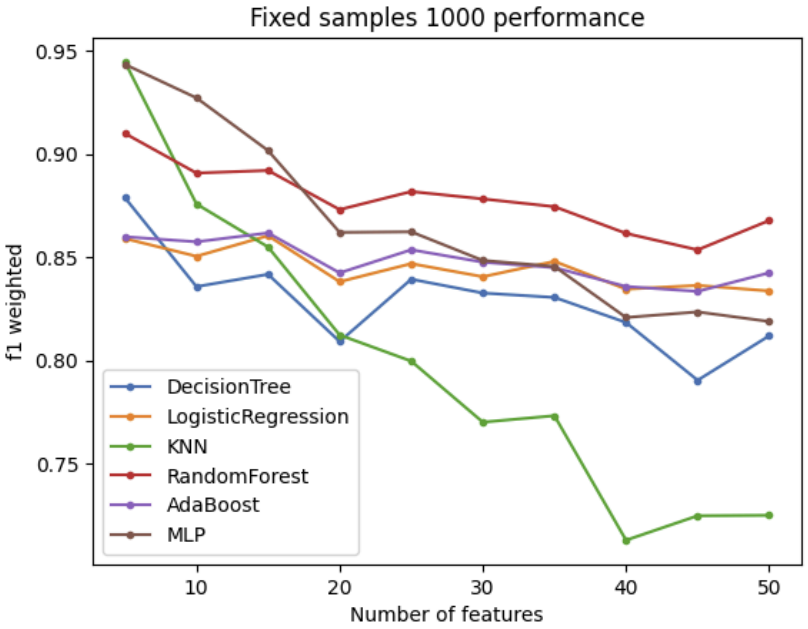
a. Weighted F1 (performance): F1 score is a metric that combines precision and recall, providing a balance between false positives and false negatives. It is computed through cross-validation.
b. Distance between training and test set (distance): performs multiple iterations of train-test splits and computes the f1 score differences across these splits, then takes the mean of these differences.
c. Execution time (speed) of each algorithm.

2.3 Results: data evaluation and comparison

2.3.1. Fixed number of samples (1000), increasing number of features

Consider the experiment with the number of samples set at 1000 samples.

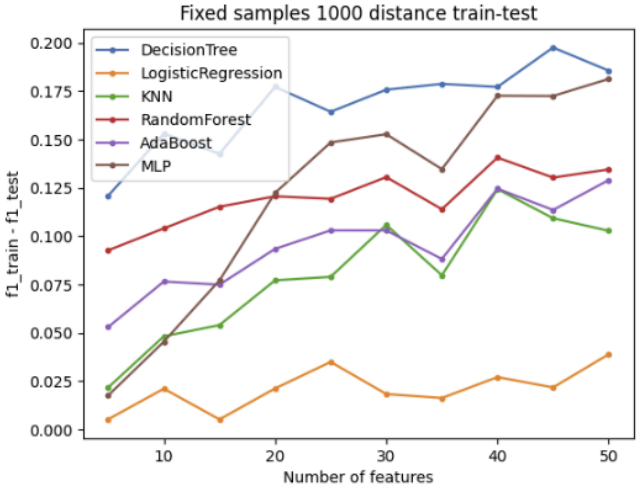
Analysis of the performance:



	5	10	15	20	25	30	35	40	45	50
DecisionTree	0.878767	0.835673	0.841631	0.809082	0.839257	0.832567	0.830447	0.818311	0.790336	0.811868
LogisticRegression	0.858947	0.850367	0.860313	0.838047	0.846797	0.840464	0.847903	0.834467	0.836259	0.833580
KNN	0.944547	0.875697	0.854878	0.812240	0.799586	0.770033	0.773147	0.712873	0.724706	0.724923
RandomForest	0.909877	0.890677	0.891920	0.872982	0.881712	0.878150	0.874397	0.861459	0.853476	0.867706
AdaBoost	0.859821	0.857379	0.861666	0.842338	0.853512	0.847506	0.844888	0.835743	0.833360	0.842363
MLP	0.943319	0.927078	0.901577	0.861951	0.862215	0.848358	0.845444	0.820738	0.823429	0.818785

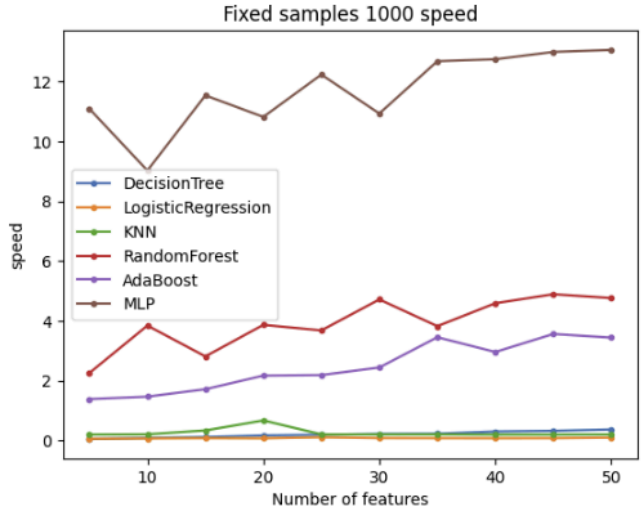
We notice that by adding features, the performance of KNN, DecisionTree and MLP worsens significantly while the performance of LogisticRegression, AdaBoost and RandomForest worsens very little.

This means that KNN, DecisionTree and MLP suffer from the "curse of dimensionality" i.e., when too many features are added compared to the number of samples, the models start to store the noise present in the training data instead of generalizing; in these cases it is said that they are overfitting the problem. While LogisticRegression, AdaBoost and RandomForest offer a more flexible structure that can handle new features without overfitting, this is due to the use of appropriate regularization techniques.



From this graph we infer the same thing as the previous one, the functions in this case grow because the performance distance between training set and test set grows. This is a classic symptom of overfitting.

Analysis of the speed:

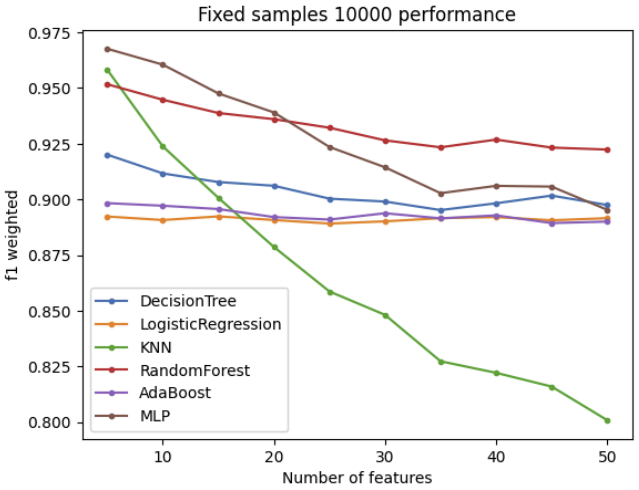


We notice that generally by increasing the features the execution time does not change. The execution time can be influenced by several factors, in this case it could be that the amount of added features is not enough to affect the speed considering that we have well designed and optimized algorithms. However, we can see that two algorithms are much faster than the others, KNN and LogisticRegression. This is not surprising, since they are the most simple ones.

2.3.2. Fixed number of samples (10'000), increasing number of features

If we consider an experiment with a much larger number of samples, say 10'000 samples (see the graph below), generally the performance of the algorithms is better than the experiment with fewer samples, and increasing the number of features does not lead to a marked deterioration in performance. This could be because increasing the features up to 50 is not enough to get into the problem of high dimensionality given such a large number of samples. We notice that the only algorithm to get much worse, despite the high number of samples, is KNN. This is due to the specific behavior of that algorithm, it assigns a label to a new instance based on the labels of its nearest

neighbors. So when the number of features is very high, the distance between any two instances becomes very similar, making it hard for KNN to distinguish between them.

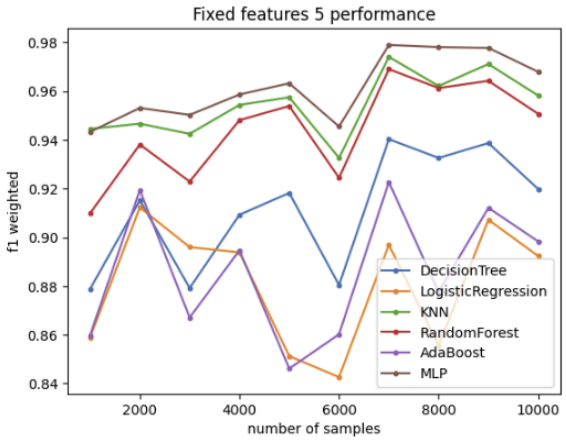


	5	10	15	20	25	30	35	40	45	50
DecisionTree	0.920084	0.911661	0.907789	0.906123	0.900332	0.899037	0.895211	0.898252	0.901707	0.897500
LogisticRegression	0.892315	0.890732	0.892372	0.890755	0.889144	0.890152	0.891563	0.892073	0.890617	0.891567
KNN	0.958208	0.923821	0.900714	0.878659	0.858660	0.848170	0.827327	0.822140	0.815970	0.800887
RandomForest	0.951704	0.944740	0.938786	0.936018	0.932175	0.926506	0.923388	0.926843	0.923256	0.922375
AdaBoost	0.898320	0.897156	0.895671	0.892060	0.890947	0.893782	0.891486	0.892826	0.889369	0.890030
MLP	0.967626	0.960499	0.947583	0.939036	0.923538	0.914416	0.902791	0.906081	0.905743	0.895334

2.3.3. Fixed number of features (5), increasing number of samples

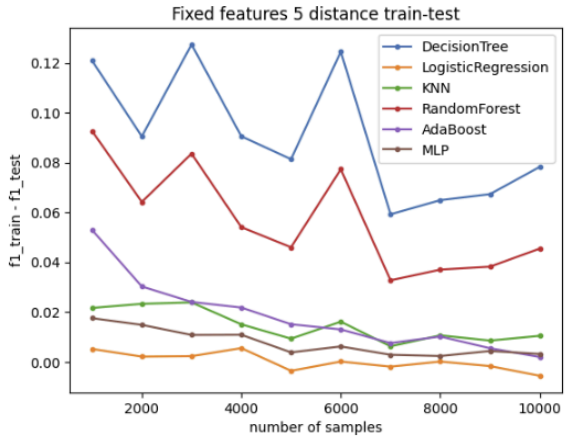
Consider the experiment with the number of features fixed at 5 features.

Analysis of the performance:



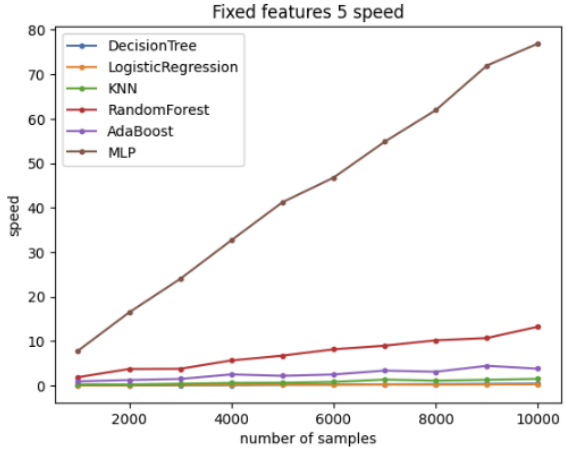
	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
DecisionTree	0.878767	0.915217	0.879183	0.909268	0.918241	0.880425	0.940294	0.932604	0.938703	0.919834
LogisticRegression	0.858947	0.912284	0.895985	0.893840	0.851237	0.842619	0.896944	0.855665	0.907222	0.892315
KNN	0.944547	0.946665	0.942502	0.954372	0.957500	0.932641	0.974107	0.962126	0.971153	0.958208
RandomForest	0.909877	0.938127	0.922905	0.948124	0.953914	0.924448	0.969048	0.961299	0.964256	0.950705
AdaBoost	0.859821	0.919396	0.867085	0.894677	0.846120	0.860196	0.922727	0.877650	0.912120	0.898320
MLP	0.943319	0.953124	0.950274	0.958643	0.963249	0.945488	0.978988	0.978123	0.977775	0.967959

We notice that performance improves in all algorithms, this is because by adding samples we are adding new information (that is “real”, not duplicates) so the model learns to generalize better. It is important to note that there may be a point at which the performance increase settles.



From this graph we infer the same thing as the previous one. In this case the function decreases a little because the difference between training set and test set decreases, this means that the model is learning better.

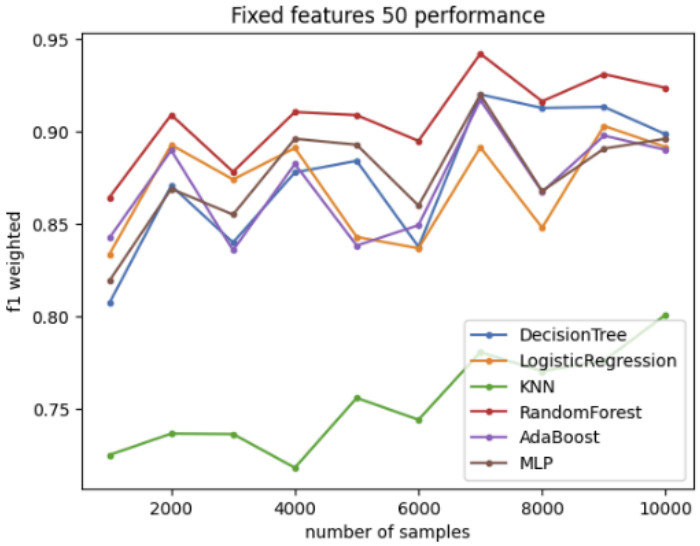
Analysis of the speed:



Obviously the execution time of the algorithms generally increases because we are increasing the amount of data to train on, we also notice here that some algorithms are well optimized and therefore their speed suffers less from the increase in data.
Instead, as stated in the previous Data Quality issue, MLP is a Neural Network and suffers particularly on very large datasets.

2.3.4. Fixed number of features (50), increasing number of samples

If we consider an experiment with a fixed number of features that is much larger, say 50 features (see the graph below), then we will notice that the increase in samples will lead to a greater increase in performance than in an experiment with few features. This is because by starting with few features our model already has good performance, while by starting with many features we must have many samples to have good performance.
For example, DecisionTree algorithm with 5 features, increasing the sample goes from 0.87 to 0.91, while with 50 features goes from 0.80 to 0.89.



	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
DecisionTree	0.807165	0.870654	0.839883	0.877830	0.884079	0.837566	0.920000	0.912707	0.913284	0.898457
LogisticRegression	0.833580	0.892733	0.873994	0.891025	0.842913	0.836747	0.891476	0.847868	0.903003	0.891567
KNN	0.724923	0.736483	0.736250	0.718003	0.755804	0.743986	0.780520	0.770251	0.775895	0.800887
RandomForest	0.864309	0.908979	0.878152	0.910516	0.908828	0.894867	0.942079	0.916297	0.930972	0.923547
AdaBoost	0.842363	0.889803	0.835703	0.882400	0.838244	0.849288	0.917022	0.867240	0.897714	0.890030
MLP	0.819181	0.868970	0.855006	0.896057	0.892831	0.859850	0.919940	0.867710	0.890687	0.896080

3. References

In this report, we reported only the most relevant graphs and tables. The complete results can be found in the attached Notebook or in the following Google Drive folder:

🗀 Data Quality Project Results Garbazio Pastori