UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

AUGUSTO EXENBERGER BECKER

A Systematic Study of the Impact of Data Leakage in Machine Learning Pipelines for Classification

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof[a]. Dr[a]. Mariana
Recamonde-Mendoza

Porto Alegre
March 2026

CIP — CATALOGING-IN-PUBLICATION

Becker, Augusto Exenberger

A Systematic Study of the Impact of Data Leakage in Machine Learning Pipelines for Classification / Augusto Exenberger Becker. – Porto Alegre: PPGC da UFRGS, 2026.

110 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2026. Advisor: Mariana Recamonde-Mendoza.

1. Data contamination. 2. Evaluation bias. 3. Classification. 4. Methodological pitfalls. 5. Performance overestimation. I. Recamonde-Mendoza, Mariana. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Prof[a]. Marcia Barbosa
Vice-Reitor: Prof[a]. Pedro Costa
Pró-Reitora de Pós-Graduação: Prof[a]. Claudia Wasserman
Diretor do Instituto de Informática: Prof. Luciano Paschoal Gaspary
Coordenador do PPGC: Prof. Gabriel Luca Nazar
Bibliotecário-chefe do Instituto de Informática: Alexsander Borges Ribeiro

ABSTRACT

Data leakage refers to the inadvertent contamination of training data with information from validation or test sets, leading to overoptimistic performance estimates and poor reproducibility of machine learning (ML) results. Although the general concept of data leakage is well known, its impact can manifest in subtle ways across different stages of the ML pipeline, particularly during data preparation steps. This dissertation presents a systematic experimental evaluation of the impact of data leakage introduced at different data preparation steps, including normalization, value imputation, feature selection, hyperparameter tuning, and class balancing, as well as a worst-case scenario in which leakage is simultaneously introduced across all preparation steps. The experimental study was conducted on 42 structured datasets from the Penn Machine Learning Benchmark, and average hardness measures were computed to investigate their relationship with the magnitude of performance inflation. Models were trained using eight widely used classification algorithms under a controlled paired experimental design, differing only in the presence or absence of data leakage, with F1-score adopted as the primary evaluation metric. The results reveal a consistent tendency of data leakage to inflate performance estimates, with substantial variability across datasets and leakage sources. Average F1-score increases reached up to 14% for specific datasets, and 35 out of the 42 datasets exhibited positive average performance gains. Among individual pipeline steps, class balancing using SMOTE produced the largest average impact, with an average F1-score increase of 3.97%, followed by feature selection and hyperparameter tuning, with average increases of 0.54% and 0.43%, respectively, while normalization and value imputation did not exhibit critical effects. The worst-case scenario analysis revealed markedly larger impacts, with average performance increases of 65%, 16%, and 10% in specific datasets, while also leading to an incorrect choice of the best performing algorithm in 40.48% of our Worst-case experiments. Overall, these findings demonstrate that data leakage can substantially distort empirical evaluations in ML. Beyond quantifying these effects, this work derives a set of evidence-based guidelines to mitigate data leakage, prioritizing high-risk pipeline stages and promoting leakage-aware experimental protocols to improve the reliability and reproducibility of ML results.

Keywords: Data contamination. evaluation bias. classification. methodological pitfalls. performance overestimation.

Um Estudo Sistemático sobre o Impacto do *Data Leakage* em Pipelines de Aprendizado de Máquina para Classificação

RESUMO

Data leakage refere-se à contaminação inadvertida dos dados de treinamento com informações provenientes dos conjuntos de validação ou teste, levando a estimativas de desempenho excessivamente otimistas e à baixa reprodutibilidade dos resultados em aprendizado de máquina (ML). Embora o conceito geral de *data leakage* seja bem conhecido, seus impactos podem se manifestar de formas sutis ao longo das diferentes etapas do pipeline de ML, particularmente durante as tarefas de preparação dos dados. Esta dissertação apresenta uma avaliação experimental sistemática do impacto do *data leakage* introduzido em diferentes tarefas de preparação dos dados, incluindo normalização, imputação de valores, seleção de atributos, ajuste de hiperparâmetros e balanceamento de classes, bem como um cenário de pior caso no qual o vazamento é introduzido simultaneamente em todas as tarefas de preparação. O estudo experimental foi conduzido sobre 42 conjuntos de dados estruturados do Penn Machine Learning Benchmark. Para cada conjunto de dados, medidas médias de dificuldade foram calculadas com o objetivo de investigar sua relação com a magnitude da inflação de desempenho causada pelo *data leakage*. Os modelos foram treinados utilizando oito algoritmos de classificação amplamente empregados, sob um desenho experimental pareado e controlado, diferindo apenas pela presença ou ausência de *data leakage*, sendo a métrica F1-score adotada como principal critério de avaliação. Os resultados revelam uma tendência consistente de o *data leakage* inflar as estimativas de desempenho, com variação substancial entre conjuntos de dados e fontes de vazamento. Aumentos médios no F1-score atingiram valores de até 14% para conjuntos de dados específicos, e 35 dos 42 conjuntos de dados apresentaram ganhos médios de desempenho positivos. Entre as tarefas individuais de preparação dos dados, o balanceamento de classes utilizando SMOTE apresentou o maior impacto médio, com um aumento médio de 3.97% no F1-score, seguido pela seleção de atributos e pelo ajuste de hiperparâmetros, com aumentos médios de 0.54% e 0.43%, respectivamente, enquanto normalização e imputação de valores não apresentaram efeitos críticos. A análise do cenário de pior caso revelou impactos significativamente maiores, com aumentos médios de desempenho de 65%, 16% e 10% em conjuntos de dados específicos, ao mesmo tempo em que causou mudanças no algoritmo que melhor performou para a tarefa em 40,48%

do experimentos de pior caso. De forma geral, esses resultados demonstram que o *data leakage* pode distorcer substancialmente avaliações empíricas em ML. Além de quantificar esses efeitos, este trabalho deriva um conjunto de diretrizes baseadas em evidências para mitigar o *data leakage*, priorizando etapas de maior risco no pipeline e promovendo protocolos experimentais conscientes de vazamento, com o objetivo de melhorar a confiabilidade e a reprodutibilidade dos resultados em aprendizado de máquina.

Palavras-chave: contaminação de dados, viés de avaliação, classificação, falhas metodológicas, superestimação de desempenho.

LIST OF FIGURES

Figure 2.1 An example of KNN-based classification, assuming $k = 5$. 18
Figure 2.2 An example of a linear SVC decision boundary for binary classification. 19
Figure 2.3 Plot of the Sigmoid function, with the characteristic "S"-shaped curve. 20
Figure 2.4 Decision diagram of a Decision Tree 21
Figure 2.5 Decision diagram of a Random Forest 22
Figure 2.6 Diagram of a Neural Network 23

Figure 4.1 Diagram illustrating one cross-validation repetition with preprocessing applied outside the cross-validation loop (leakage-prone configuration). 40
Figure 4.2 Diagram illustrating one cross-validation repetition with preprocessing applied inside each fold of the cross-validation loop (leakage-free configuration). 40
Figure 4.3 Diagram illustrating the leakage-prone hyperparameter tuning configuration, where a single-level cross-validation is used both to select hyperparameters and to evaluate the model. 41
Figure 4.4 Diagram illustrating the leakage-free hyperparameter tuning configuration, where nested cross-validation separates hyperparameter selection (inner loop) from model evaluation (outer loop). 42
Figure 4.5 Diagram illustrating the experimental protocol for the experiments with data leakage insertion through individual pipeline steps. 43

Figure 5.1 Distribution of F1-score paired differences across experiments. 47
Figure 5.2 PCA for generated clusters based on the perceived difference differences in paired performance metrics 48
Figure 5.3 Distribution of F1-score paired differences by pipeline step. 51
Figure 5.4 Proportion of experiments from each pre-training pipeline step across leakage-impact clusters. 52
Figure 5.5 Distribution of F1-score paired differences by learning algorithm. 54
Figure 5.6 Proportion of experiments from each learning algorithm across leakage-impact clusters. 55
Figure 5.7 Distribution of F1-score paired differences for datasets with highest $\overline{\Delta F1}$. . 58
Figure 5.8 Distribution of F1-score paired differences for six additional datasets with highest $Max\Delta F1$. 58
Figure 5.9 Correlation between dataset class imbalance and mean F1-score difference. 59
Figure 5.10 Distributions of selected hardness measures across clusters. 61
Figure 5.11 Heatmap of $\overline{\Delta F1}$ for combinations of algorithms and pipeline steps. 62
Figure 5.12 Distribution of F1-score paired differences for algorithms considering only leakage introduced in the hyperparameter tuning step. 63
Figure 5.13 Heatmap of $\overline{\Delta F1}$ for combinations of datasets and pipeline steps. 64
Figure 5.14 Distribution of F1-score paired differences for datasets with highest $\overline{\Delta F1}$ in the class balancing step. 65
Figure 5.15 Heatmap of $\overline{\Delta F1}$ for combinations of datasets and algorithms. 66

Figure C.1 Plots of the values used for selecting the best value of K 86

Figure D.1 Counts of experiment executions according to $\Delta F1$ values 87
Figure D.2 Counts of experiment executions according to $\Delta F1$ values grouped by pipeline step used to insert data leakage 88

Figure E.1 Distribution F1-Score differences for different percentages of missing values on the subset of 30 datasets 90
Figure E.2 Distribution F1-Score differences for different imputation strategies on the subset of 30 datasets 90
Figure E.3 Distribution F1-Score differences for different percentages of selected features 90

Figure F.1 Performance difference distribution for datasets (Part 1) 91
Figure F.2 Performance difference distribution for datasets (Part 2) 92
Figure F.3 Performance difference distribution for datasets (Part 3) 92
Figure F.4 Performance difference distribution for datasets (Part 4) 93
Figure F.5 Performance difference distribution for datasets (Part 5) 93
Figure F.6 Performance difference distribution for datasets (Part 6) 94
Figure F.7 Performance difference distribution for datasets (Part 7) 94

Figure H.1 Correlation between class imbalance and mean F1-Score difference for each of the 42 datasets. 98
Figure H.2 Correlation between number of Classes and mean F1-score difference for each of the 42 datasets. 99
Figure H.3 Correlation between features per class and mean F1-score difference for each of the 42 datasets. 99
Figure H.4 Correlation between number of Features and mean F1-score difference for each of the 42 datasets. 100
Figure H.5 Correlation between instances per class and mean F1-score difference for each of the 42 datasets. 100
Figure H.6 Correlation between number of Instances and mean F1-score difference for each of the 42 datasets. 101

Figure I.1 Distribution of complexity features across clusters (Part 1). 102
Figure I.2 Distribution of complexity features across clusters (Part 2). 103
Figure I.3 Distribution of complexity features across clusters (Part 3). 103
Figure I.4 Distribution of complexity features across clusters (Part 4). 104
Figure I.5 Distribution of complexity features across clusters (Part 5). 104

Figure K.1 Distribution of F1-Score differences for datasets with highest $\overline{\Delta F1}$ for the hyperparameter tuning step. 107
Figure K.2 Distribution of F1-Score differences for datasets with highest $\overline{\Delta F1}$ for the feature selection step. 108
Figure K.3 Distribution of F1-Score differences for datasets with highest $\overline{\Delta F1}$ for the missing-value imputation step. 108
Figure K.4 Distribution of F1-Score differences for datasets with highest $\overline{\Delta F1}$ for the normalization step. 109
Figure K.5 Distribution of F1-Score differences for datasets with highest $\overline{\Delta F1}$ for the worst-case scenario. 109

Figure L.1 Cluster analysis overview showing distribution of algorithms, pipeline steps, and PCA visualization. These results are obtained considering the experiments for the worst-case scenario. 110

LIST OF TABLES

Table 3.1 Summary of the related work and expected contributions of this study. 34

Table 4.1 Datasets selected from PMLB and their metadata. 37

Table 5.1 Overall analysis of F1-scores in leakage-prone and leakage-free models across all executions (10,080 paired comparisons). 46
Table 5.2 Statistical summary of paired performance differences within each cluster. 49
Table 5.3 Analysis of F1-score differences between leakage-prone and leakage-free models aggregated by pipeline step. Dark green means that $\Delta F1$ is positive and greater than 1%, light green means that $\Delta F1$ is positive but lower than 1%, and white means that the p-value shows no statistically significant difference for the comparison (*i.e.,* p > 0.05). 50
Table 5.4 Analysis of F1-score differences between leakage-prone and leakage-free models aggregated by learning algorithm. Dark green means that $\Delta F1$ is positive and greater than 1%, light green means that $\Delta F1$ is positive but lower than 1%. 53
Table 5.5 Percentage of experiments in which data leakage changed the identity of the best-performing algorithm (*winner flip*) for overall experiments and for each pipeline step. 55
Table 5.6 Analysis of F1-score differences between leakage-prone and leakage-free models aggregated by dataset. Row colors are encoded as follows: dark green indicates an average influence above 1%; light green indicates a positive influence below 1%; yellow marks negative average influence; and white denotes non-significant results (p-value > 0.05). 57
Table 5.7 Spearman correlation among average hardness measures of datasets and their mean F1-score differences comparing leakage-prone and leakage-free models. 60
Table 5.8 Summary of F1-scores differences in models with and without leakage for all experiments in the worst-case scenario (n=2,016). 65
Table 5.9 Dataset-level analysis of F1-score differences between leakage-prone and leakage-free models under the worst-case scenario. Row colors are encoded as follows: dark green indicates an average influence above 1%; light green indicates a positive influence below 1%; yellow marks negative average influence; and white denotes non-significant results (p-value > 0.05). 67
Table 5.10 Algorithm-level analysis of F1-score differences between leakage-prone and leakage-free models under the worst-case scenario. Dark green row color indicate an average influence above 1%. 68

Table E.1 Analysis of F1-scores for different percentages of selected features. 89
Table E.2 Analysis of F1-scores for different percentages of missing values. 89
Table E.3 Analysis of F1-scores for different imputation strategies. 89

Table G.1 Wilcoxon signed-rank analysis analysis of F1-scores in models with and without leakage across all executions. 95
Table G.2 Wilcoxon signed-rank analysis of F1-scores for each dataset. 96
Table G.3 Wilcoxon signed-rank analysis of F1-scores for each algorithm. 97
Table G.4 Wilcoxon signed-rank analysis of F1-scores for each pipeline step. 97

Table G.5 Wilcoxon signed-rank analysis of F1-scores in models with and without leakage for worst-case scenario. 97

Table H.1 Spearman correlation between dataset metadata and mean F1-score difference comparing leakage-prone and leakage-free models. 98

Table J.1 Datasets and their average value for the hardness measures (Part 1) 105
Table J.2 Datasets and their average value for the hardness measures (Part 2) 106

LIST OF ABBREVIATIONS AND ACRONYMS

CB	Class Balance
CL	Class Likelihood
CLD	Class Likelihood Difference
DCP	Disjunct Class Percentage
DS	Disjunct Size
DT	Decision Tree
F1	Overlapping Feature Fraction
F2	Minimum Feature Overlap
F3	Mean Feature Overlap
F4	Maximum Feature Overlap
H	Harmfulness
KNN	K-Nearest Neighbors
kDN	k-Disagreeing Neighbors
LR	Logistic Regression
LSC	Local Set Cardinality
LSR	Local Set Radius
ML	Machine Learning
MV	Minority Value
N1	Borderline Points
N2	Intra-Extra Ratio
NB	Naive Bayes
NN	Neural Networks
PMLB	Penn Machine Learning Benchmarks
RF	Random Forest

SVC	Support Vector Classifier
TD_P	Tree Depth Pruned
TD_U	Tree Depth Unpruned
U	Usefulness
XGB	XGBoost

CONTENTS

1 INTRODUCTION 14
2 THEORETICAL BACKGROUND 17
2.1 Machine learning 17
2.2 Supervised learning algorithms 18
2.2.1 K-nearest neighbors 18
2.2.2 Support vector classifier 19
2.2.3 Logistic regression 19
2.2.4 Naive Bayes 20
2.2.5 Decision trees 20
2.2.6 Random forest 21
2.2.7 XGBoost 21
2.2.8 Neural networks 22
2.3 Pre-training pipeline steps 23
2.3.1 Data preprocessing and transformation 23
2.3.2 Feature selection 24
2.3.3 Hyperparameter tuning 25
2.4 Model evaluation 25
2.5 Data leakage 26
2.6 Instance hardness measures 28
3 RELATED WORK 30
3.1 Data leakage: definitions, taxonomies, and prevention 30
3.2 Empirical studies on the effects of data leakage 31
3.3 Summary 34
4 METHODOLOGY 35
4.1 Datasets selection 35
4.2 Learning algorithms 36
4.3 Pre-training pipeline steps 36
4.4 Data leakage introduction 39
4.5 Model training and evaluation 41
4.6 Experimental protocol 41
4.7 Hardness measures extraction 42
4.8 Data analysis 43
5 RESULTS 45
5.1 Overall impact of data leakage on performance estimates 46
5.2 Cluster-based analysis of data leakage effects 48
5.3 Effects across different pre-training pipeline steps 49
5.4 Effects across learning algorithms 53
5.5 Effects across datasets 56
5.6 Relationship between dataset hardness and leakage effects 59
5.7 Analysis of pipeline step, algorithm and dataset interactions 62
5.8 Worst-case scenario analysis 65
6 EVIDENCE-BASED GUIDELINES FOR AVOIDING DATA LEAKAGE 70
6.1 Rationale and scope 70
6.2 High-risk data preparation steps 70
6.3 Medium- and low-risk steps 71
6.4 Worst-case scenarios and compounding effects 72
6.5 Practical recommendations 72
7 CONCLUSION 74

REFERENCES 77
APPENDIX A — RESUMO EXPANDIDO 81
APPENDIX B — FULL LIST OF HYPERPARAMETERS EVALUATED 84
B.1 Logistic Regression 84
B.2 Support Vector Classifier 84
B.3 K-Nearest Neighbors 84
B.4 Random Forest 84
B.5 Decision Tree 84
B.6 Naive Bayes 84
B.7 Neural Networks 85
B.8 XGBoost 85
APPENDIX C — METRICS USED TO SELECT NUMBER OF CLUSTERS K-MEANS 86
APPENDIX D — COUNTS OF PAIRED EXPERIMENTS FOR $\triangle F1$ VALUES 87
APPENDIX E — ADDITIONAL ANALYSIS FOR MISSING-VALUE IMPUTATION AND FEATURE SELECTION STEPS 89
APPENDIX F — DISTRIBUTION OF PERFORMANCE DIFFERENCES ACROSS ALL DATASETS 91
APPENDIX G — DETAILED STATISTICS RESULTS 95
APPENDIX H — CORRELATIONS OF DATASETS METADATA AND PERCEIVED F1-SCORE DIFFERENCE 98
APPENDIX I — DISTRIBUTION OF HARDNESS MEASURES 102
APPENDIX J — DATASETS AND AVERAGE HARDNESS MEASURES 105
APPENDIX K — DISTRIBUTION OF F1-SCORE DIFFERENCES FOR HIGHEST IMPACTED DATASETS IN EACH PIPELINE STEP 107
APPENDIX L — CLUSTER ANALYSIS CONSIDERING THE WORST-CASE SCENARIO 110

1 INTRODUCTION

With the increasing adoption of machine learning (ML) models in various areas of knowledge, such as finance(DIXON; HALPERIN; BILOKON, 2020), medicine(NAQA; MURPHY, 2015), and education (LUAN; TSAI, 2021), ensuring the reliability and robustness of such models is rapidly becoming a central concern. To have confidence in decision models, it is important to ensure that the performance estimates obtained during training and validation are reproduced in real-life scenarios. However, Kapoor and Narayanan (2023) investigated the reproducibility crisis, a phenomenon that highlights the difficulty of matching model performance estimates due to methodological mishaps during training and evaluation. The authors examined papers from 17 fields and reported that many studies present overoptimistic performance estimates as a result of overlooked methodological flaws.

A recurring methodological flaw that has gained increasing attention is data leakage. Data leakage occurs when training data becomes contaminated with information from the testing and/or validation sets, leading to an overestimation of performance during model evaluation (KAUFMAN et al., 2012). Although the general concept of data leakage is well known, more subtle forms of leakage, beyond an explicit failure to separate training and test data, have become increasingly prevalent. These forms of leakage are often introduced during different stages of the machine learning pipeline, particularly, particularly within the pre-training pipeline steps, which are the focus of this work.

In this study, we use the term *pre-training pipeline steps* to refer to the operations performed prior to model fitting. These include data preprocessing steps (*e.g.,* normalization, missing-value imputation), data transformation steps (*e.g.,* class balancing), feature selection procedures, and model configuration steps such as hyperparameter tuning. When improperly executed, these steps may allow information from the validation or test sets to leak into the training process, resulting in data leakage and the related issue of inflated performance values (APICELLA; ISGRÒ; PREVETE, 2025; BERNETT et al., 2024; KAPOOR; NARAYANAN, 2023).

Recent efforts have focused on identifying the causes of data leakage, understanding its effects in specific applications, and proposing strategies to detect and mitigate it. For instance, Kaufman et al. (2012) propose formulations and techniques to detect and avoid leakage in machine learning models. Taxonomies of different leakage types have also been proposed to categorize how leakage is introduced and how it can be prevented

(KAPOOR; NARAYANAN, 2023). Additionally, static code analysis tools have been developed to identify potential leakage patterns in machine learning implementations (DROBNJAKOVIĆ; SUBOTIĆ; URBAN, 2024; YANG et al., 2022). Other studies investigate the effects of data leakage in specific application domains, reporting substantial performance inflation under contaminated evaluation protocols (SAMALA et al., 2020; SHIM; LEE; HWANG, 2021; YAGIS et al., 2021).

Despite these advances, existing studies remain largely fragmented and limited in scope. Most investigations focus on isolated sources of data leakage or on narrowly defined application domains, often considering a small number of datasets or learning algorithms. There is still a lack of systematic and controlled evaluations that quantify the impact of data leakage across multiple pre-training pipeline steps, learning algorithms, and datasets under a unified experimental protocol. As a result, although data leakage is widely recognized as a major methodological flaw, its actual consequences for model evaluation remain insufficiently understood. In practice, several fundamental questions are still unanswered. For instance: *How strongly and how consistently does data leakage inflate performance estimates? Do certain pre-training pipeline steps pose higher risks for performance inflation? Do all algorithms suffer equally, or are some more vulnerable than others? Is the magnitude of overestimation dependent on dataset characteristics (such as instance hardness measures), or is leakage uniformly harmful across domains?*

While it is broadly assumed that leakage leads to optimistic and unreliable results, these assumptions are rarely supported by large-scale, systematic evidence. Existing claims are typically based on intuition, isolated case studies, or domain-specific anecdotes. Consequently, practitioners lack concrete guidance on how severe leakage can be, which steps pose the greatest risk, and how its effects interact with algorithm, pipeline steps, and dataset properties.

To address this gap, this work presents a systematic experimental investigation of the effects of data leakage on the performance estimates of machine learning classifiers. The proposed analysis evaluates leakage introduced at different stages of the machine learning pipeline, including normalization, value imputation, feature selection, hyperparameter tuning, and class balancing. Multiple classification algorithms are compared under controlled and paired experimental settings, ensuring that observed performance differences can be attributed to the presence of data leakage. Furthermore, the experimental evaluation spans a large collection of publicly available structured datasets, enabling the analysis of variability in leakage effects across different data characteristics.

The study also investigates worst-case scenarios in which multiple sources of data leakage are combined, as well as the relationship between dataset hardness measures and the magnitude of performance inflation. Together, these contributions provide a comprehensive and comparative view of how data leakage impacts performance estimates in practical machine learning pipelines focused in classification tasks, generating quantitative evidence that aim to help shed light on several open questions in the field. Partial results of this work have been published on the 35th Brazilian Conference on Intelligent Systems (BECKER; RECAMONDE-MENDOZA, 2025).

This work is organized as follows. Chapter 2 presents the theoretical background. Chapter 3 discusses related literature. Chapter 4 details the methodology used in our experiments. Chapter 5 presents the results and analysis, Chapter 6 presents a set of experiment based guidelines, and Chapter 7 concludes the work and outlines future research directions.

2 THEORETICAL BACKGROUND

This chapter presents the theoretical basis for this work, including supervised learning algorithms, model development pipeline, data leakage, and instance hardness measures.

2.1 Machine learning

Machine Learning (ML) is a subfield of artificial intelligence focused on developing computational models that improve their performance through experience. More formally, Mitchell (1997) defines ML as the study of algorithms that improve at a task T with respect to a performance measure P based on experience E. This is accomplished by using a variety of algorithms to approximate a target function to describe data behavior by leveraging example data or past experience (ALPAYDIN, 2020). Nowadays, applications of ML algorithms and techniques can be seen in many areas of knowledge, such as bioinformatics (TAN et al., 2021), medicine (NAQA; MURPHY, 2015), smart cities (ULLAH et al., 2020), education (LUAN; TSAI, 2021), and finance (DIXON; HALPERIN; BILOKON, 2020).

Two major groups of ML algorithms are available when it comes to their process of training and their goals: supervised and unsupervised learning. Supervised learning algorithms aim to create a 'mapping' function based on given inputs and outputs for known data points. The learned function can then be used to predict the output for a given, unseen input data (CUNNINGHAM; CORD; DELANY, 2008). This output can be either categorical for classification tasks or numerical for regression tasks. In contrast, unsupervised learning techniques work with data points that do not have a previously defined output, aiming to find patterns within the data. These patterns can be used to create clusters or for organizing these values (GREENE; CUNNINGHAM; MAYER, 2008), for example.

This works focus on classifications tasks. Thus, in what follows we revise selected supervised learning algorithms commonly applied for this purpose.

2.2 Supervised learning algorithms

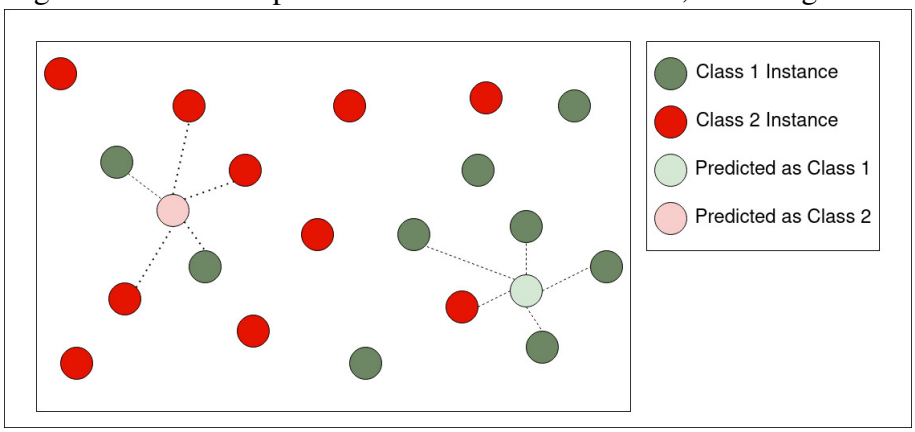
This section presents an overview of supervised learning algorithms relevant to this work: k-nearest neighbors, support vector classifier, logistic regression, naive Bayes, decision trees, random forests, XGBoost, and neural networks.

2.2.1 K-nearest neighbors

The K-Nearest Neighbors (KNN) algorithm is a classification algorithm that uses similar instances with a known classes to predict the class of given instance. As the name suggests, it uses the K nearest data instances for the classification, and the predicted class is the most common class across these K neighbors (KRAMER, 2013). It relies on the premise that similar instances will probably have similar classification. Figure 2.1 shows an example of how instances would be classified in a context with two classes (*i.e.,* binary problem) and K=5.

The KNN approach can create a very complex decision plane, as for every instance the classification depends on the know classes of it's surrounding. This algorithm is also very sensitive to it's three hyperparameters: number of nearest Neighbors (K), the distance function used to select the nearest neighbors, and the weighting function (in case of using weighted voting among nearest neighbors)(KANG, 2021)

Figure 2.1: An example of KNN-based classification, assuming $k = 5$.

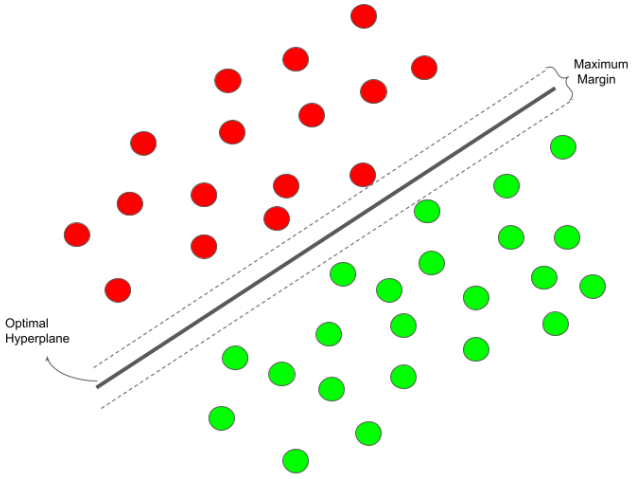


Source: The Author

2.2.2 Support vector classifier

The Support Vector Classifier (SVC) algorithm uses a hyperplane to divide the decision space, maximizing the merging between the decision plane and the nearest data instances from distinct classes(NOBLE, 2006). Figure 2.2 exemplifies how this is occurs on a two-dimensional space with two distinct classes. The plane generated by the algorithm splits the data and keeps a margin from the data instances.

Figure 2.2: An example of a linear SVC decision boundary for binary classification.



Source: The Author

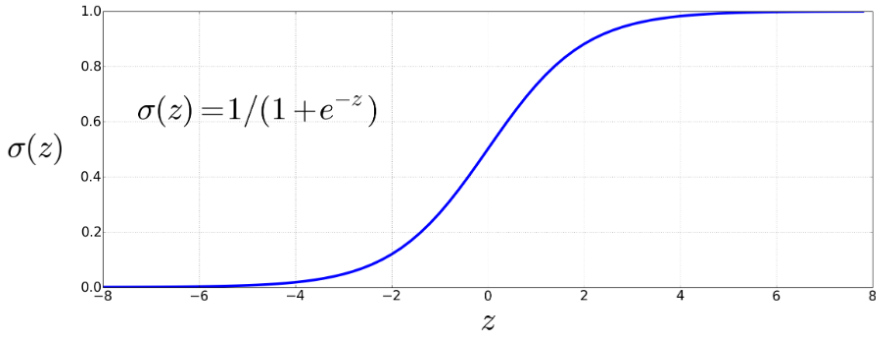
Originally the SVC algorithm was only able to work with a linear separating plane. But different kernels and techniques allow SVC to have good performance with non-linear tasks. One of these techniques is knows as the kernel trick, which consists in projecting the data points into higher dimensional spaces The algorithm then obtains linear separability on this higher dimension space and transforms it back to the original vector space (SUTHAHARAN; SUTHAHARAN, 2016).

2.2.3 Logistic regression

Logistic Regression (LR) is statistical algorithm that models the probability of a instance being of a class as a linear combination of one or more independent variables. For binary classification problems, LR employs the sigmoid function (Figure 2.3) to make decisions about new observations (JURAFSKY et al., 2014). Our implementation, based on scikit-learn (PEDREGOSA et al., 2011), adopts the One-vs-Rest approach to determine the predicted class. In this framework, the sigmoid function is applied to each class

individually against all other classes. The class corresponding to the highest predicted score is then selected as the final prediction

Figure 2.3: Plot of the Sigmoid function, with the characteristic "S"-shaped curve.



Source: Jurafsky et al. (2014)

2.2.4 Naive Bayes

The Naive Bayes (NB) algorithm is a classification algorithm based on the Bayes Theorem. The algorithm calculates the *a posteriori* probability for each class for a given instance using the *a priori* and conditional probabilities (BERRAR, 2018).

Equation 2.1 describe the calculation of the probability of a data instance x belonging to class y_i, where x has d features. This probability is calculated by multiplying the probability of each feature value x_j conditioned on the class. It also uses the *a priori* probability of the class, $P(y_i)$. The ultimate class prediction is made by selecting the class with the *a posteriori* probability $P(y_i|x)$.

$$P(y_i|x) = P(y_i) \prod_{j=1}^{d} P(x_j|y_i) \tag{2.1}$$

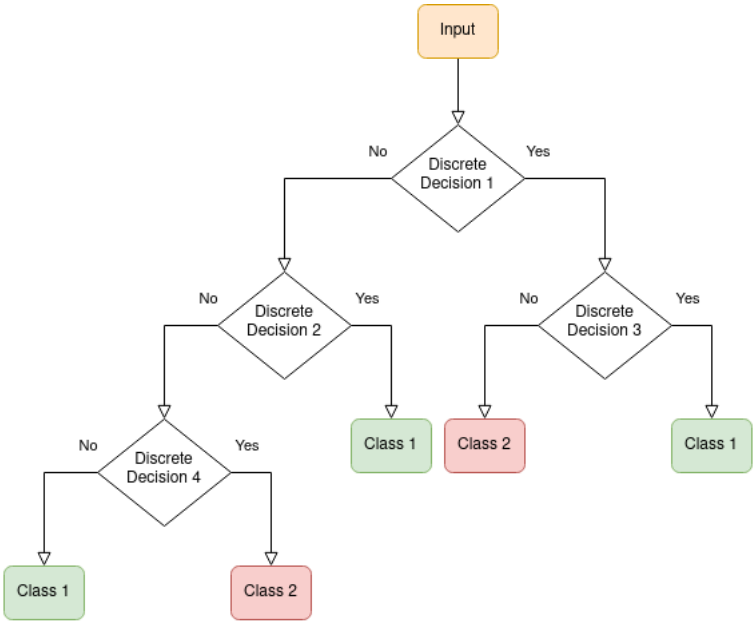
2.2.5 Decision trees

The Decision Tree (DT) classifier is an algorithm based on sequential divisions of the decision space. The algorithm uses discrete functions based on input features to split the decision in each of the sequential levels of the tree (ROKACH; MAIMON, 2005). The model training selects the feature whose split produces the highest impurity decrease at each step.

The classification is performed by following the sequential discrete functions until

a termination(leaf) node is reached. Each leaf node is related to a specific class and determines the classification of the instance whose inputs lead to the specific node when applied to the tree's discrete functions. Figure 2.4 exemplifies a DT, with termination nodes and the successive decisions.

Figure 2.4: Decision diagram of a Decision Tree



Source: The Author

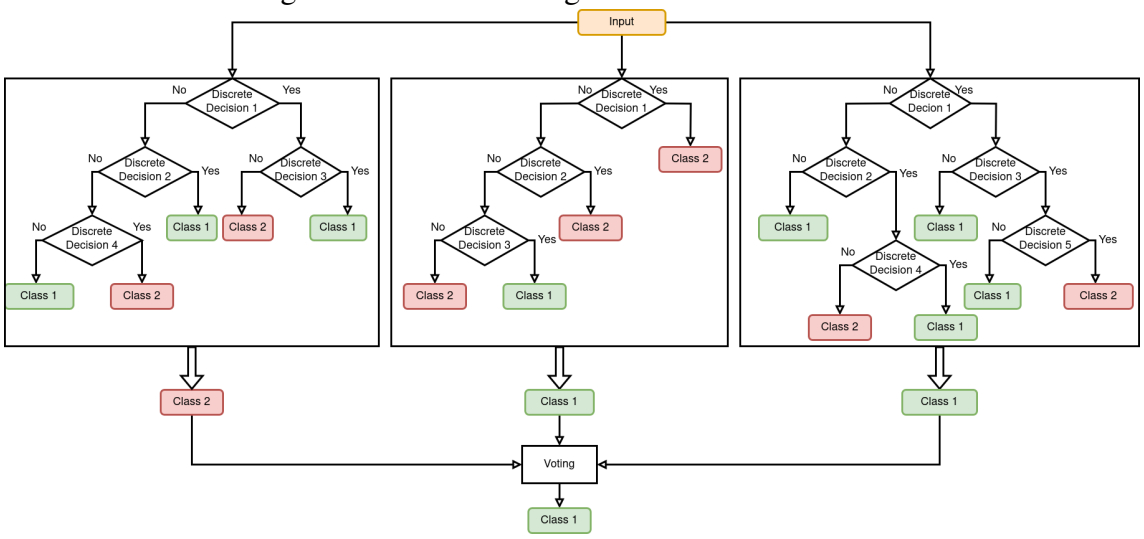
2.2.6 Random forest

A Random Forest (RF) is an ensemble of DTs. This specific type of ensemble is characterized by the composition of different trees trained with a random selection of instances and features (CUTLER; CUTLER; STEVENS, 2012). The training process and overall classification steps for each of the parts of the ensemble is similar to the ones of a DT, detailed in subsection 2.2.5. The main difference is that on the classification, as the model is an ensemble, the RF output predicted class will be selected by a majority voting of the individual DTs. Figure 2.5 exemplifies the composition of a RF ensemble.

2.2.7 XGBoost

XGBoost (XGB) is a special case of a Decision Tree ensemble that leverages the concept of Gradient Boosting. This concept refers to the process of iteratively training

Figure 2.5: Decision diagram of a Random Forest



Source: The Author

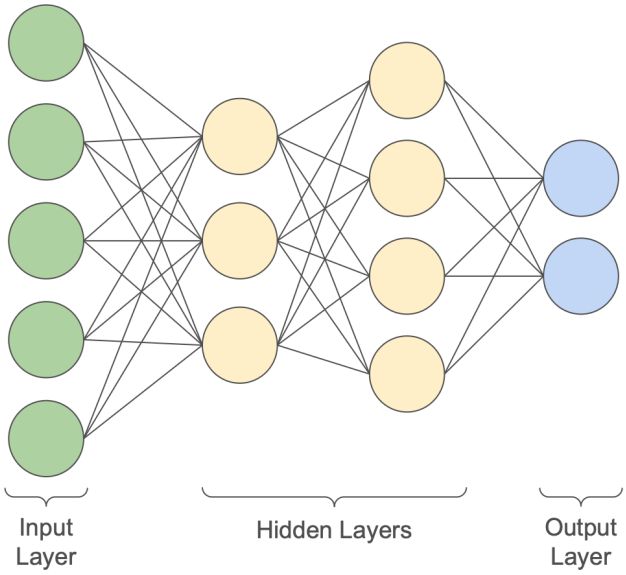
models to solve the errors of it's predecessors. The XGB algorithm is, therefore, an implementation of tree boosting with very relevant optimizations and state-of-the-art performance(CHEN; GUESTRIN, 2016). The diagram to exemplify XGB is very similar to the RF presented in Figure 2.5, but each of the trees was trained based on the results of the ones trained beforehand.

2.2.8 Neural networks

Neural Networks (NN) are a well known machine learning algorithm. Inspired by behavior of the human brain, the algorithm gets it's name from the artificial Neurons that compose it structure. The basic composition of a NN contains an input layer of neurons, an output layer of neurons, and with possibly multiple hidden layers of neurons that connect these two.

Figure 2.6 presents a basic architecture of a NN. The input layer receives and processes the features of the data set, and forwards them to the hidden layers, who forward them to the output layer. The outputs values are dependent variables of the function that the NN approximates (WANG, 2003). Each neuron executes an activation function, based on its weighted inputs and a bias value. The result of the activation is passed to the following neurons with an associated weight. The training of a NN is based on the gradient descent and uses backpropagation to pass the feedback of the current performance of the network.

Figure 2.6: Diagram of a Neural Network



Source: The Author

2.3 Pre-training pipeline steps

In the context of this work, *pre-training pipeline steps* refer to the sequence of operations performed before fitting the model to the training data. These steps prepare both the data and the model configuration, and may include data preprocessing procedures (e.g., normalization, missing-value imputation), data transformation operations (e.g., class balancing), feature selection methods, and model-configuration tasks such as hyperparameter tuning. Subsection 2.3.1 details the preparation steps that are relevant to our analysis, Subsection 2.3.2 explains the rational fo the feature selection procedure, and Subsection 2.3.3 introduces the concept of hyperparameter tuning and explains its role within the overall pipeline.

2.3.1 Data preprocessing and transformation

Preprocessing refers to the techniques executed over the data before the model is trained. This encompasses several steps in the training pipeline that manipulate the data in different ways. Examples of preprocessing steps include missing-value imputation, normalization, and class balancing.

Value imputation is a process for dealing with missing values in datasets by replacing the missing entries with reasonable estimates (YOZGATLIGIL et al., 2013). There

are multiple algorithms used to perform this substitution, ranging from simple strategies—such as replacing missing values using the mean or median of the feature—to more sophisticated approaches that rely on KNN or other ML algorithms (GARCÍA; LUENGO; HERRERA, 2015).

Normalization consists in transforming the input data in way that features with larger variability on their values do not have higher impact on the approximated model (GARCÍA; LUENGO; HERRERA, 2015). Various normalization techniques exist. In the context of this work, we adopt the commonly used z-score standardization, which rescales each feature by subtracting its mean and dividing by its standard deviation, producing features with mean 0 and variance 1. Other normalization techniques,such as Min–Max scaling also exist. This technique uses the maximum and minimum values of each feature as the upper and lower bounds when calculating the normalized feature value. However, min-max scaling was not used in our experiments.

Class balancing is an approach used to mitigate the issue of class imbalance, *i.e.,* disproportionate number of instances per class in the dataset, in real-world datasets (SHA et al., 2022). This process can be done in a variety of ways, for example by oversampling the minority classes or by under-sampling the majority classes. One largely used approach of class balancing is SMOTE. This technique oversamples the minority class by creating synthetic instances through interpolation between a sample and its nearest neighbors (CHAWLA et al., 2002).

2.3.2 Feature selection

Feature selection is the process of reducing the dimensionality of the data by selecting a subset of informative features to be used during model training. This reduction aims to improve model generalization, decrease computational cost, and remove noisy or redundant attributes (GARCÍA; LUENGO; HERRERA, 2015).

Among the different families of feature-selection methods, this work focuses on filter-based approaches, which evaluate each feature independently of the learning algorithm. These methods rely on statistical criteria to quantify the relevance of attributes with respect to the target variable, selecting those that most strongly satisfy a given scoring measure. Examples of filter criteria include dependence-based measures (*e.g.,* correlation, mutual information), statistical measure (*e.g.,* ANOVA F-value), separability measures, consistency measures, and information-based metrics (GARCÍA; LUENGO; HERRERA,

2015). Because filter methods do not rely on model training, they are computationally efficient and less prone to overfitting compared to wrapper or embedded methods, making them suitable for high-dimensional scenarios or for systematic studies involving a wide range of experiments.

2.3.3 Hyperparameter tuning

Hyperparameter tuning is the process of optimizing the configurable parameters of ML models to better fit the needs of the target data set and boost performance. The optimization is made through iteratively changing the parameters of models, and choosing the values that yield the best performance(RASCHKA, 2018). The hyperparameters change according to the ML algorithm being used, examples are the number of nearest neighbors to be evaluated in KNN, the number and size of hidden layers in NN, and the max depth of a DT.

2.4 Model evaluation

Model evaluation aims to estimate how well a trained model will generalize to unseen data. To enable this a general approach is to create partitions of the dataset called *Training*, *Validation*, and *Test* sets, each having a specific purpose (BURZYKOWSKI et al., 2023):

- **Training set**: Used to approximate the model that will be evaluated.
- **Validation set**: Used to estimate performance during training process.
- **Test set**: Used to estimate final performance of the model.

Cross-validation is one of the most widely used approaches for this purpose. In k-fold cross-validation, the dataset is partitioned into k equally sized folds; the model is trained on k-1 folds and tested on the remaining one, and the procedure is repeated until each fold has served as the test set once. The final performance is obtained by averaging the results across folds. When hyperparameter tuning is required, nested cross-validation is known to offer a more reliable assessment by separating the hyperparameter optimization (inner loop) from the performance estimation (outer loop), thus ensuring that the evaluation reflects the true generalization ability of the tuned model. With generalization

as the main goal, overfitting becomes a central issue in ML. Domingos (2012) highlights the need of such evaluation strategies that reliably estimate performance on unseen data.

A variety of metrics can be used to quantify predictive performance, including Accuracy, Balanced Accuracy, Precision, Recall, and F1-score. While Accuracy measures the proportion of correctly classified samples, Balanced Accuracy averages the recall across classes, offering a fairer view in imbalanced settings. Precision quantifies how many predicted positives are correct, Recall measures how many actual positives are recovered by the model, and the F1-score combines both into a single harmonic-mean indicator. Together, these evaluation procedures and metrics provide a comprehensive understanding of model behavior under different conditions.

2.5 Data leakage

Data Leakage is the name given to the inadvertent contamination of training data with information from data that will be used to test and validate a given machine learning model. This contamination breaks the assumption that the validation and test data can be seen as equivalent, or close to equivalent, to real-world unseen data. Therefore the estimates generated based on these sets may not represent the true performance the model would have when tested against completely unknown data points.

At a minimum, data leakage leads to overestimation of the performance of the trained models. However, it can also lead to overfitting when the estimate perceived during training fails to translate into real-world performance. This discrepancy arises due to a a failure of using the full generalization power of the model, once it has had extra information from the data it was tested against. Consequently, models trained with data leakage usually are less generalized then the ones where data leakage was not inserted (KAUFMAN et al., 2012).

Data leakage can happen in various ways, a few being more straight forward, such as having proxy features for the target, or failing to avoid intersection between the training and tests. It can also happen in more subtle ways, such as through incorrect methodology used during the machine learning pipeline (KAPOOR; NARAYANAN, 2023). In this work we focus on the data leakage that is inserted during what we call pre-training pipeline steps, the set of transformation and preparatory steps that are taken before the process of fitting model to the data. These steps encompass normalization, value imputation, class balancing, feature selection, and hyperparameter tuning.

In what follows, we summarize how the introduction of data leakage may happen in the five steps addressed in the current work:

- **Normalization:** Data leakage occurs in normalization when this step is performed before the dataset is split into the test and train sets. This allows information about the test data to be leaked into the training set. To avoid contamination, the normalization algorithm should be fitted using the training set, and later used to transform the training and test data separately (DROBNJAKOVIĆ; SUBOTIĆ; URBAN, 2024).
- **Missing-value imputation:** Similarly to normalization, data leakage can be inserted through missing value imputation when this step is performed before the data set is split into training and test data. When this occurs, the imputer uses information from the test data to define which values should be generate to fill what is missing. The solution would be to use only training data to fit the imputer and use this imputer to fill the missing values in the training and test sets (KAPOOR; NARAYANAN, 2023).
- **Class Balancing (SMOTE):** Since class balancing through SMOTE incites the creation of new synthetic instances. Data Leakage is inserted when the data used to estimate the model performance is also explored by the oversampling method to generate data (APICELLA; ISGRÒ; PREVETE, 2025). Therefore, to avoid leakage, the oversampling should only happen inside the specific training and testing sets, and never before the data has been split.
- **Feature Selection:** The insertion of leakage through feature selection varies according to the approach used to select the best features. For feature selection based on feature importance scores, leakage occurs when this is executed before the separation of test and training data. Since this would let test data contribute to the selection of features. To avoid it, simply separating the test data before the execution of feature selection would prevent data leakage. In more advance techniques, that rely on the training and evaluation of models with different combinations of features, using a single test/train split to select the features would lead to leakage, once the features would also be optimized for data it should not have access to (SAMALA et al., 2020). In this setup, avoiding leakage would require a nested cross-validation loop, with the inner loop being used for feature selection and the outer loop to train and evaluate the performance of the models based on selected features.

- **Hyperparameter Tuning:** Similarly to the advanced method of feature selection, hyperparameter tuning relies on evaluating multiple hyperparameter configurations through strategies such as grid search. Therefore, using a single train/test split for tuning the hyperparameters and for evaluating the performance, would lead to the contamination of the training set (YANG et al., 2022). The solution would also be to use a nested cross-validation approach, with the inner loop implementing the grid search for the hyperparameters and the outer loop being used to evaluate the model performance.

2.6 Instance hardness measures

Instance hardness measures are metrics used to evaluate how hard it is for a particular instance to be classified. Rather than summarizing model performance at the global level, these metrics characterize the local difficulty of individual samples, capturing factors such as neighborhood ambiguity, class overlap, imbalance, or the complexity required for a model to correctly separate a point in the feature space. Understanding these aspects is useful for interpreting model behavior, diagnosing challenging regions of the data, and analyzing how instance-level properties relate to predictive performance.

In the context of this work, we consider the hardness measures implemented by (PAIVA et al., 2022). Below is a list of measures relevant to our analysis and their description.

- **k-Disagreeing Neighbors (kDN):** Percentage of k nearest neighbors with different class labels than the instance in analysis
- **Disjunct Size (DS):** Measures how much a algorithm has to divide the decision space to correctly predict an instance
- **Disjunct Class Percentage (DCP):** Measures the percentage of instances of the class of the evaluated instance on the same classification space
- **Tree Depth Unpruned (TD_U):** Normalized depth of the leaf node in an unpruned decision tree
- **Tree Depth Pruned (TD_P):** Normalized depth of the leaf node in a pruned decision tree
- **Class Likelihood (CL):** Measures the likelihood of an instance belonging to a class. To Provide a higher value for harder instance the complement of this value is

provided by pyhard

- **Class Likelihood Difference (CLD):** Difference between the instance's class likelihood and the highest likelihood for other classes. The complement of this value is provided on th pyhard implementation
- **Minority Value (MV):** Imbalance ratio of the instance's class relative to the majority class
- **Class Balance (CB):** Skewness of the instance's class distribution in the dataset
- **Borderline Points (N1):** Fraction of different-class neighbors in the minimum spanning tree
- **Intra-Extra Ratio (N2):** Ratio of intra-class distance to nearest enemy distance
- **Local Set Cardinality (LSC):** Complement of the relative size of the instance's local set
- **Local Set Radius (LSR):** Complement of the normalized radius of the instance's local set
- **Harmfulness (H):** Proportion of instances having this instance as their nearest enemy
- **Usefulness (U):** Complement of the fraction of instances with this instance in their local sets
- **Overlapping Feature Fraction (F1):** Percentage of features whose values lie in class-overlapping regions
- **Minimum Feature Overlap (F2):** Minimum degree of overlap across all features for the instance
- **Mean Feature Overlap (F3):** Average degree of overlap across all features for the instance
- **Maximum Feature Overlap (F4):** Maximum degree of overlap across all features for the instance

3 RELATED WORK

This chapter reviews the main studies related to data leakage in machine learning. We begin by discussing papers that define this issue and propose taxonomies for different types of data leakage. Next, we examine previous efforts to assess its impact on the performance of ML models. Finally, we provide an objective comparison between our work and prior research in the literature.

3.1 Data leakage: definitions, taxonomies, and prevention

Several studies have investigated how data leakage occurs and the ways it can affect performance estimates. The study on *Leakage and the Reproducibility Crisis*, conducted by Kapoor and Narayanan (2023), presents an analysis of how mishandling data during the preprocessing steps can lead to over-optimistic results. The authors provide evidence of what they called the "*Reproducibility Crisis*", referring to the difficulty of reproducing the results obtained during model evaluation due to inflated performance estimates. Their survey covers 20 studies across 17 areas of knowledge, identifying important methodological pitfalls in more than 329 papers.The authors reproduced and corrected the tests on selected papers to prove the influence of data leakage on the performance estimates and, in some cases, in the conclusions about the best algorithm to a given problem. Alongside this investigation of leakage and its effects, they provide a taxonomy to classify the types of data leakage.

Other works expand this discussion by organizing and defining data-leakage scenarios more systematically. Sasse et al. (2025) present an overview of scenarios that can lead to data leakage. They compile a set of leakage types and provide concrete examples illustrating how each can affect trained models. In another foundational study, Kaufman et al. (2012) introduce formal definitions of data leakage, which serve as the basis for proposing leakage detection techniques and a methodology to avoid leakage.

Another line of work focuses on automated detection and prevention strategies through static analysis. Drobnjaković, Subotić and Urban (2024) proposed a static analysis tool within the open-source NBLyzer static analysis framework, defining concrete semantics for identifying leakage and evaluating their approach on 200 Kaggle competition notebooks with promising results. Another proposed static analysis tool (YANG et al., 2022) tracks data flows through the code, aiming to identify situation where leakage

was inserted. Their results indicate good performance on identifying the target types of leakage.

Domain-specific investigations further illustrate the practical challenges of detecting leakage in real-world applications. On a study of data leakage in biological ML applications, Bernett et al. (2024) propose a set of guiding questions intended to help practitioners identify subtle forms of leakage. They provide use-case examples that corroborate the usefulness of their questions in identifying data leakage signs in non-trivial and highly specialized scenarios. Likewise, Apicella, Isgrò and Prevete (2025) study the introduction of data leakage through what they call the "push the button" approach to ML model training, in which users rely on automated tools without proper consideration of pipeline design. They define this approach as the inadvertent use of convenient ML tools without taking into consideration the pipeline definition and the tasks that should be executed, a practice that can inadvertently allow test data to influence the training process.

Finally, concerns about systematic leakage have also emerged in applied ML fields such as medical imaging. An investigation of Convolutional Neural Networks for Alzheimer's disease classification (WEN et al., 2020) found that more than half of the surveyed studies may have been affected by data leakage, resulting in substantially overestimated performances. The authors identified multiple sources of contamination, including incorrect or late data splitting, biased transfer learning, and the absence of independent test sets, underscoring the prevalence and impact of leakage even in high-stakes domains.

3.2 Empirical studies on the effects of data leakage

Beyond taxonomies and definitions, several empirical studies have evaluated the consequences of data leakage under different contexts and methodological designs. These works typically compare performance estimates obtained under contaminated and leakage-free experimental protocols, often within specific application domains, and consistently report substantial inflation in evaluation metrics when leakage is present.

An investigation of preprocessing practices in cognitive classification methods (FARROW; MOORE; GAsEVIć, 2019) showed that common preprocessing steps may lead to over-optimistic results when contamination occurs. In their case study, cross-validation alone was insufficient to prevent leakage, and the data needed to be split by processing session to ensure reliable estimates. Similarly, a study on model validation for datasets with limited instances (VABALAS et al., 2019) examined how incorrect val-

idation strategies can introduce bias in small-sample contexts. Using synthetic data, the authors compared five validation schemes—train/test split, k-fold cross-validation, nested cross-validation, and two variants of partially nested CV—and found that relying on a single level of cross-validation during preprocessing led to substantial bias. Their findings also highlight that, when training and test data were pooled, feature selection was the largest contributor to biased performance evaluation, surpassing even hyperparameter tuning.

Several studies investigated leakage effects introduced through feature selection. Samala et al. (2020) assess breast cancer classification models and show that performing feature selection using validation data can inflate the area under the ROC curve (AUC) from approximately 75% to values as high as 99%. When evaluated on an independent and leakage-free test set, the same models achieved AUC values closer to 72%. The authors also report that the magnitude of performance inflation increases as the sample size decreases. Related work by Kuncheva and Rodríguez (2018) focuses on feature selection in low-sample scenarios and proposes an evaluation protocol designed to avoid overestimated performance estimates. Their results indicate that leakage-aware procedures produce estimates closer to those obtained using independent test sets when compared to contaminated evaluation protocols.

Similarly, Shim, Lee and Hwang (2021) compare contaminated and uncontaminated feature selection procedures using two data sources—synthetic data generated from Gaussian distributions and real medical data focused on neuro-psychiatric diseases—and observe inflated performance estimates in both scenarios.

Data leakage effects have also been reported in medical imaging applications due to the improper handling of correlated samples. Yagis et al. (2021) show that using slices derived from the same 3D MRI scans during both training and testing can inflate classification accuracy from 29% to up to 55%. Similarly, Bussola et al. (2021) report that, in digital pathology applications, performance estimates can be inflated by up to 41% when images from the same patient are included in both training and test sets.

Leakage introduced during model selection has been investigated by Cawley and Talbot (2010), who analyze bias resulting from improper hyperparameter tuning. Their study shows that reusing validation data both during hyperparameter optimization and final model evaluation leads to overfitting and biased performance estimates. The authors propose an evaluation methodology aimed at producing unbiased estimates by accounting for overfitting during model selection.

The effects of introducing data leakage through class balancing using SMOTE were investigated by Wen et al. (2025). The authors compared the effects of data leakage introduced through value imputation and SMOTE in the context of stroke prediction. Their results indicated that SMOTE caused a substantially higher impact on performance estimates when applied before the data were split into training and test folds. The authors suggested class-weight adjustment as a leakage-free alternative for handling imbalanced datasets.

More recent empirical analyses explore data leakage effects across multiple algorithms and datasets. Bouke and Abdullah (2023) investigate intrusion detection systems using three datasets and six classification algorithms, reporting that models trained with data leakage consistently achieved higher performance estimates, although the magnitude of the effect varied across algorithms. In their experiments, decision trees exhibited a larger sensitivity to leakage when compared to k-nearest neighbors, XGBoost, random forests, support vector classifiers, and logistic regression.

In the context of deep learning applied to electroencephalography data, Brookshire et al. (2024) demonstrates that incorrect splitting of correlated data can severely inflate performance estimates. Their study evaluates two applications—Alzheimer's disease detection and seizure classification—by comparing splits performed at the image level and at the patient level. When improper splitting strategies were used, reported performance increased from 53% to 99% for Alzheimer's disease classification and from 65% to 79% for seizure detection.

Finally, Rosenblatt et al. (2024) investigate the effects of multiple leakage sources in neuro-imaging applications, analyzing four different datasets. Their results indicate that leakage introduced through feature selection and repeated subjects leads to substantial increases in performance estimates. In particular, models with lower baseline predictive performance experienced more pronounced performance inflation. Complementary to these findings, Gupta, Christopher and Ramisetty (2025) evaluate mitigation strategies across eight classification algorithms and ten publicly available datasets, showing that techniques such as overlap dropping and feature dropping can reduce leakage effects under specific experimental conditions.

3.3 Summary

This chapter reviewed studies related to data leakage in machine learning, covering conceptual definitions, detection and prevention strategies, and empirical analyses of leakage effects. Table 3.1 summarizes the scope of the reviewed works and contrasts them with the dimensions investigated in this thesis. The table highlights that, while prior studies address individual aspects of data leakage, a comprehensive evaluation spanning multiple pipeline steps, learning algorithms, datasets (from distinct domains), and worst-case scenarios remains still unexplored.

Table 3.1: Summary of the related work and expected contributions of this study.

	Define and Understand Data Leakage	Propose Strategy to Avoid Data Leakage	Assess Effects of Data Reuse	Assess Effects on Feature Selection	Assess Effects on Hyperparameter Tuning	Assess Effects on Normalization	Assess Effects on Value Impute	Assess Effects on Class Balancing	Assess Effects on a Worst-Case Scenario	Investigate Effects variation among Pipeline Steps	Investigate Effects variation among Algorithms	Investigate Effects on different Datasets	Investigate Effects on Deep Learning
Cawley and Talbot (2010)		X			X								
Kaufman et al. (2012)	X	X											
Kuncheva and Rodríguez (2018)			X	X									
Farrow, Moore and Gašević (2019)		X											
Vabalas et al. (2019)		X											
Samala et al. (2020)				X									
Wen et al. (2020)												X	X
Bussola et al. (2021)		X	X										
Shim, Lee and Hwang (2021)				X									
Yagis et al. (2021)			X										
Drobnjaković, Subotić and Urban (2024)		X											
Yang et al. (2022)		X											
Kapoor and Narayanan (2023)	X	X		X								X	
Bouke and Abdullah (2023)				X							X	X	
Rosenblatt et al. (2024)		X		X						X		X	
Brookshire et al. (2024)			X										X
Bernett et al. (2024)	X	X											
Apicella, Isgrò and Prevete (2025)	X												X
Wen et al. (2025)							X	X	X	X			
Gupta, Christopher and Ramisetty (2025)	X	X	X	X							X	X	
Sasse et al. (2025)	X												
This Work	X	X		X	X	X	X	X	X	X	X	X	

Source: The Author

4 METHODOLOGY

This chapter describes the methodology used to conduct the experiments and analyses presented in this work. The experimental design was structured to enable controlled comparisons between model training pipelines with data leakage ("leakage-prone" pipeline) and without data leakage ("leakage-free" pipeline) across multiple datasets, algorithms, and preparation steps. We detail the process of dataset selection, the definition of the learning algorithms included in the study, the procedures used to introduce and avoid data leakage in our experimental pipelines, and the evaluation metrics and experimental protocol adopted. Finally, we describe the analyses performed on the collected results to investigate the effects of data leakage on performance estimates.

4.1 Datasets selection

A total of 42 datasets from the Penn Machine Learning Benchmark (PMLB) (ROMANO et al., 2022) were selected for our experiments. PMLB was selected as the main source because it provides datasets without missing data, with no additional preprocessing steps applied beforehand, and through a unified API that easily connected to our experimental pipeline. The benchmark database also provides metadata about the datasets.

The selection of datasets was made in two phases: first, a selection of 30 datasets and next an additional 12 datasets. Both phases relied on multiple Principal Component Analysis (PCA) runs over the metadata provided by PMLB for each dataset. A PCA consists in reducing the dimensions of a dataset while retaining most of the variability from the data (RINGNÉR, 2008).

For the initial set of 30 datasets, we applied filters based on dataset size and class imbalance. The imbalance metric used by PMLB is defined as the squared distance of each class's instance proportion from a perfectly balanced distribution, where a value of 0 indicates perfect balance and values closer to 1 indicate more extreme imbalance (ROMANO et al., 2022). Specifically, we restricted the selection to datasets with at most 10,000 instances and with an imbalance value no greater than 0.2. These filters were used to avoid the influence of extreme imbalance on the experimental results and to prevent excessive computational overhead.

The selection process was split into six rounds of dataset selection guided by PCA. Each round consisted of selecting the two main metadata with PCA and selecting the five

datasets farthest from the remaining datasets when considering the two selected columns. The selection of the additional 12 datasets was made in a similar way, based on three selection rounds, but without any filter applied over the dataset size or imbalance.

Table 4.1 shows the list of the 42 selected datasets alongside their metadata. The number of instances ranges from 106 to 105,908, the number of features on the selected datasets ranges from 2 to 1000, and the class imbalance ranges from 0.0 to 0.94.

4.2 Learning algorithms

The models trained during our experiments were based on the algorithms presented in Section 2.2: KNN, SVC, LR, NB, DT, RF, XGB, and NN. The implementation of XGB used the `xgboost` Python package (CHEN; GUESTRIN, 2016), that is compatible with the `scikit-learn` package (PEDREGOSA et al., 2011). All other algorithms were implemented using the `scikit-learn` package.

4.3 Pre-training pipeline steps

Our experiments evaluated the effects of inserting leakage through five operations commonly performed prior to model fitting, which we refer to as *pre-training pipeline steps*: missing-value imputation, normalization, feature selection, class balancing, and hyperparameter tuning. This section describes the implementation details of each step, while the following section (*i.e.,* Section 4.4) explains how data leakage was deliberately introduced in our experiments.

Since there were no values missing on the original datasets, it was necessary to synthetically introduce missing data for the experiments focused on value imputation. This was achieved using the Missing at Random approach implemented in the `r-miss-tastic` package (MAYER et al., 2019), which randomly inserts missing values into selected columns (*i.e.,* features). This procedure was applied for each dataset, before running our experiments. Different percentages of missing values were introduced to evaluate whether the amount of missing data influenced the impact of data leakage: 5%, 10%, 20%, and 30%.

For missing-value imputation, the main strategy applied to all 42 datasets was median imputation, which replaces missing values using the median along each column,

Table 4.1: Datasets selected from PMLB and their metadata.

Dataset	**Instances**	**Features**	**Classes**	**Imbalance**
adult	48842	14	2	0.2719
agaricus_lepiota	8145	22	2	0.0015
analcatdata_authorship	841	70	4	0.0834
banana	5300	2	2	0.0107
chess	3196	36	2	0.0020
clean1	476	168	2	0.0170
clean2	6598	168	2	0.4785
coil2000	9822	85	2	0.7756
connect_4	67557	42	3	0.2547
dis	3772	29	2	0.9394
dna	3186	180	3	0.0777
GAMETES_Epistasis_2_Way_1000atts..._1	1600	1000	2	0.0000
Hill_Valley_with_noise	1212	100	2	0.0000
Hill_Valley_without_noise	1212	100	2	9.80e-05
kr_vs_kp	3196	36	2	0.0020
krkopt	28056	6	18	0.0517
letter	20000	16	26	3.49e-05
magic	19020	10	2	0.0881
mfeat_factors	2000	216	10	0.0000
mfeat_fourier	2000	76	10	0.0000
mfeat_karhunen	2000	64	10	0.0000
mfeat_pixel	2000	240	10	0.0000
mnist	70000	784	10	3.25e-04
molecular_biology_promoters	106	57	2	0.0000
movement_libras	360	90	15	0.0000
mushroom	8124	22	2	0.0013
optdigits	5620	64	10	1.47e-05
phoneme	5404	5	2	0.1706
ring	7400	20	2	9.47e-05
satimage	6435	36	6	0.0276
shuttle	58000	9	7	0.5852
sleep	105908	13	5	0.1485
sonar	208	60	2	0.0045
soybean	675	35	18	0.0363
spambase	4601	57	2	0.0449
spectf	349	44	2	0.2076
splice	3188	60	3	0.0777
texture	5500	40	11	0.0000
tokyo1	959	44	2	0.0775
twonorm	7400	20	2	6.57e-07
waveform_21	5000	21	3	8.04e-05
waveform_40	5000	40	3	5.79e-05

as implemented in the `SimpleImputer` class provided by the `scikit-learn` package. For comparison purposes, we also evaluated mean imputation (also implemented by `SimpleImputer`) and KNN-based imputation using the `KNNImputer`. Due to computational limitations on the larger datasets, `KNNImputer` was only evaluated on the initial group of 30 smaller datasets. Although multiple imputation strategies and missing-value percentages were explored, the aggregated analyses presented in later sections (including comparisons with other pipeline steps) rely on a standardized configuration of median imputation applied to 20% missing values. This choice ensures consistency in the number of instances across methods and allows a fair comparison of the impact of leakage across different pre-training pipeline steps.

For normalization, we adopted the z-score approach, which standardizes each feature by subtracting its mean and dividing by its standard deviation, helping to balance the influence of features with different numeric scales. This method was implemented using the `StandardScaler` class in the `scikit-learn` package.

For feature selection, the `SelectPercentile` class in the `scikit-learn` package was used. This method ranks features using a scoring function (in our case, the default `f_classif` function) and selects a subset of features according to a chosen percentile of the highest scores. In this work, we tested multiple selection levels (1%, 5%, 10%, and 20%) to assess whether the percentage of selected features influences the extent of data leakage. For datasets with very few features, a minimum of one selected feature was enforced. Although different selection percentages were explored to evaluate their effect on leakage, the aggregated analyses and the comparisons with other pipeline steps rely on a standardized configuration of selecting 10% of the features. This default choice ensures consistent input dimensionality across methods and allows a fair comparison of the impact of data leakage across different pre-training pipeline steps.

For the class balancing experiments, we adopted the SMOTE algorithm using the implementation provided by the `imblearn` package (LEMAîTRE; NOGUEIRA; ARIDAS, 2017). Due to the variability in class sample sizes across datasets and folds, the number of k-neighbors used by SMOTE was adjusted when necessary, ranging from 5 down to 1 whenever a given class did not contain enough samples within a fold.

The hyperparameter tuning experiments relied on a predefined set of hyperparameters for each target algorithm. The complete list of evaluated hyperparameters is provided in Appendix B.

4.4 Data leakage introduction

In this work, we evaluate the impact of data leakage introduced in each of the pre-training pipeline steps described in the previous section. In addition to analyzing these steps individually, we also consider a worst-case scenario in which leakage is introduced cumulatively across all pre-training pipeline steps investigated.

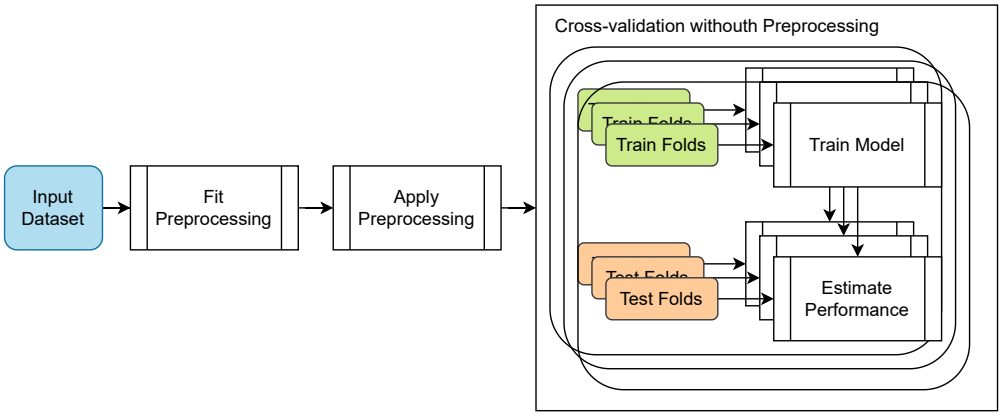
Data leakage was introduced by intentionally violating the correct separation between training and validation data during the execution of each pipeline step, allowing information from the test folds to contaminate the training process. This section presents the diagrams illustrating the training procedures for both leakage-free (*i.e.,* without data leakage) and leakage-prone (*i.e.,* with data leakage) configurations. To create models with and without data leakage, two main approaches were used, depending on the investigated pipeline step.

For the experiments involving missing-value imputation, feature selection, normalization, and SMOTE, data leakage was introduced by applying the processing step to the entire dataset before performing any type of train–test split, such as k-fold cross-validation (selected for this work). This is incorrect because statistics computed from the full dataset (such as feature medians, most relevant features, or synthetic samples generated using SMOTE) incorporate information from the test folds, which is then leaked into the training folds during cross-validation. As a result, the model indirectly gains access to data that should remain unseen during training, potentially leading to overly optimistic performance estimates. In contrast, in the leakage-free configuration, the model was trained using a traditional k-fold cross-validation setup, in which the data is first split into folds and all processing steps are applied independently inside each fold.

Figure 4.1 illustrates the leakage-prone configuration. We can notice that the complete dataset is used to run the processing step, and only afterwards split into cross-validation folds. Figure 4.2 presents the correct approach, where the split is performed before any processing, ensuring that no information from the test folds influences the transformations applied to the training data. Both configurations were implemented using the `Pipeline` class from `scikit-learn`.

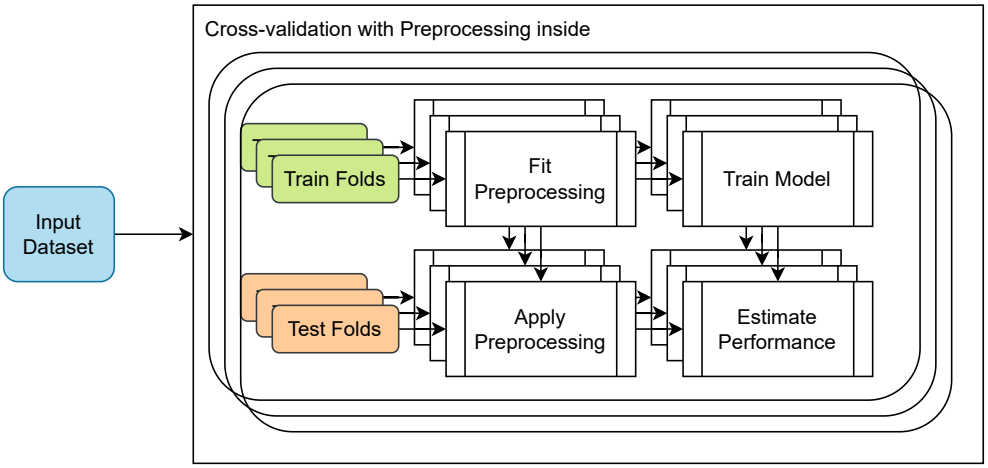
The experiments for hyperparameter tuning and for the worst-case scenario follow a slightly different approach. In the leakage-prone configuration, hyperparameters are selected using a single level of cross-validation, meaning that the same data is used both to choose the hyperparameters and to evaluate model performance. This setup is incorrect

Figure 4.1: Diagram illustrating one cross-validation repetition with preprocessing applied outside the cross-validation loop (leakage-prone configuration).



Source: The Author

Figure 4.2: Diagram illustrating one cross-validation repetition with preprocessing applied inside each fold of the cross-validation loop (leakage-free configuration).

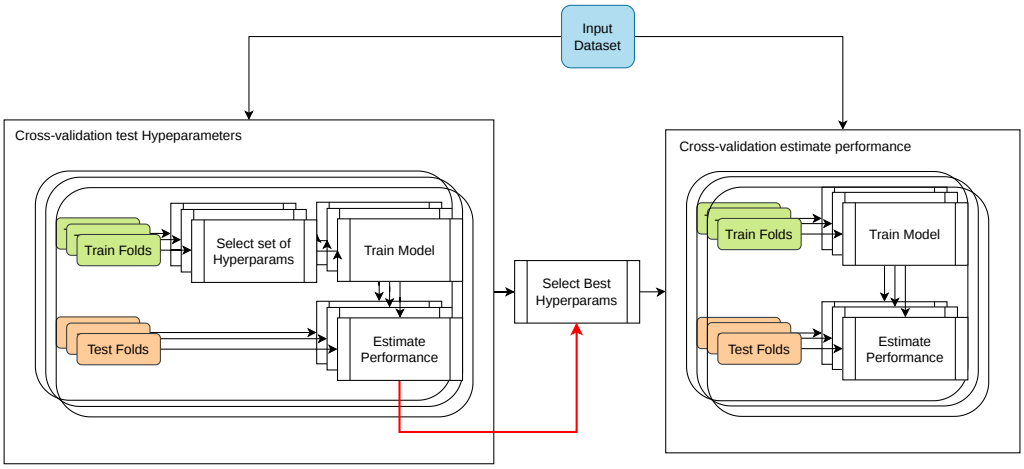


Source: The Author

because the tuning process indirectly accesses information from the validation folds, causing the selected hyperparameters to be biased toward the evaluation data, which can result in inflated performance estimates. Figure 4.3 illustrates this leakage-prone configuration.

To avoid data leakage during hyperparameter tuning, we adopt a nested cross-validation strategy. In this setup, the inner loop is responsible for selecting the hyperparameters using only the training portion of the data, while the outer loop evaluates the model on held-out data that is never used for tuning. This separation ensures that hyperparameter selection is based exclusively on training data at each iteration, preventing contamination of the evaluation folds. Figure 4.4 shows the leakage-free, nested cross-validation configuration.

Figure 4.3: Diagram illustrating the leakage-prone hyperparameter tuning configuration, where a single-level cross-validation is used both to select hyperparameters and to evaluate the model.



Source: The Author

4.5 Model training and evaluation

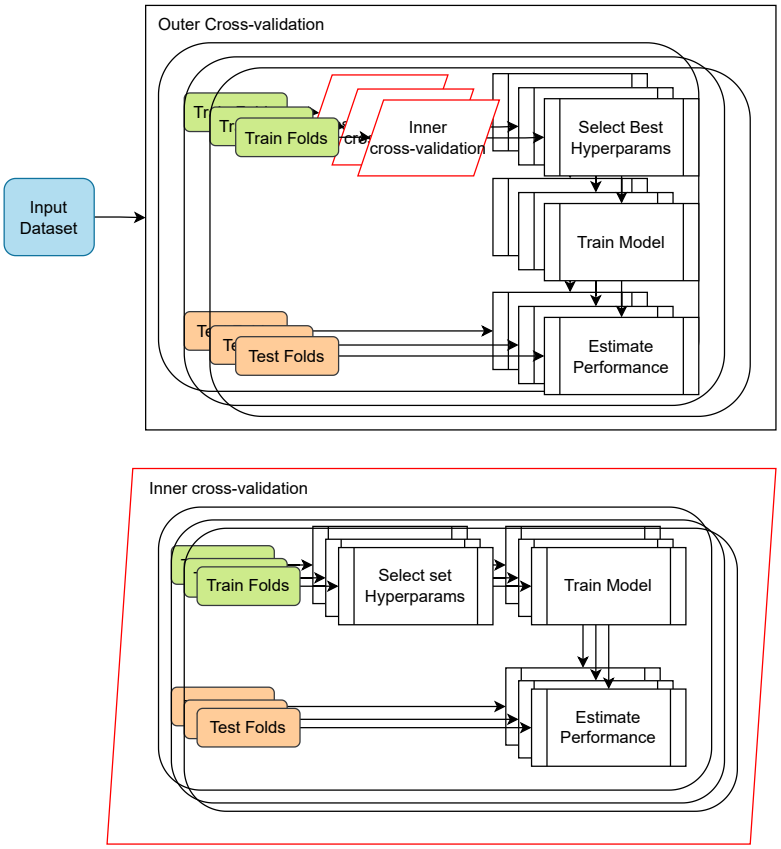
The model training was executed for each model with the target dataset, target algorithm, and target pre-training pipeline step. In this process, to allow the evaluation of performance, the following metrics were extracted in all experiments: Accuracy, Balanced Accuracy, F1-score, Precision, and Recall. The extraction of this metrics allowed the comparison of the performance estimates between model with and without the insertion of data leakage.

4.6 Experimental protocol

To enable fair comparisons between leakage-free and leakage-prone models, all experiments were executed in paired form, using the same random seeds and random-state configurations for each pair of models (*i.e.,* trained with a pipeline without or with data leakage). This pairing ensures that both models are evaluated under identical data splits and random conditions, so that any performance difference can be attributed exclusively to the presence or absence of data leakage.

Thus, for each combination of pipeline step, algorithm, and dataset, we ran six paired training and evaluation runs, one related to the leakage-free model and the other related to the leakage-prone model. Considering the five pipeline steps, 42 datasets, and

Figure 4.4: Diagram illustrating the leakage-free hyperparameter tuning configuration, where nested cross-validation separates hyperparameter selection (inner loop) from model evaluation (outer loop).



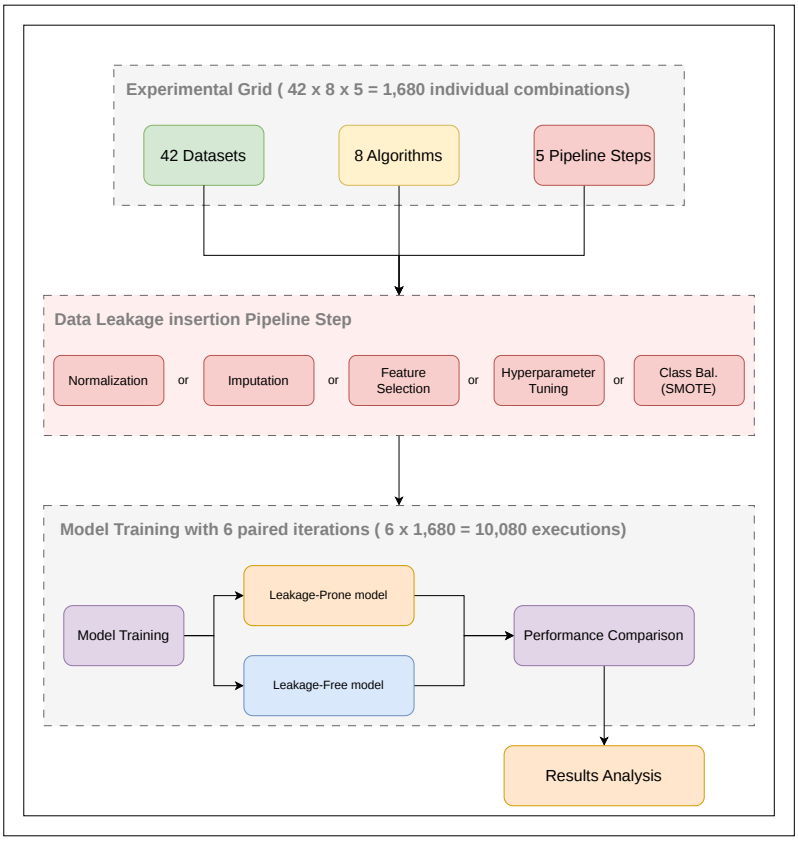
Source: The Author

eight algorithms evaluated, our experimental protocol generates a total of 10,080 paired comparisons, this process is presented in Figure 4.5. All analyses presented in this work are therefore based on these paired results. In addition, the worst-case scenario was evaluated for all 42 datasets and eight algorithms, also using six paired runs per combination, resulting in 2,016 additional paired experiments that are analyzed and discussed in the next chapter.

4.7 Hardness measures extraction

To relate patterns observed in the data leakage effects to intrinsic dataset characteristics, we computed instance-level hardness measures using the `pyhard` library (PAIVA et al., 2022). These measures quantify how difficult individual instances are to classify,

Figure 4.5: Diagram illustrating the experimental protocol for the experiments with data leakage insertion through individual pipeline steps.



Source: The Author

based on properties such as class overlap, local neighborhood structure, and feature-space complexity. The aggregated hardness values were later compared with the magnitude of performance inflation to investigate whether certain dataset characteristics are associated with stronger leakage effects.

4.8 Data analysis

The large amount of performance evaluation data generated by our experiments enabled the use of multiple complementary analysis approaches to investigate the effects of introducing data leakage. In the scope of this work, we rely on three main analyses to support our conclusions: statistical analysis, visual analysis of performance differences, and a clustering analysis based on paired execution results.

Statistical analysis was used to assess whether the differences in performance estimates between leakage-free and leakage-prone models were statistically significant. For

each paired execution group, we applied the Wilcoxon signed-rank test, which is appropriate for paired data and does not assume normality. In this test, the null hypothesis states that there is no difference between the performance estimates of the leakage-free and leakage-prone models. A p-value lower than 0.05 leads to rejecting this null hypothesis, indicating that the performance differences observed in the paired executions (*i.e.,* a leakage-free model and its corresponding leakage-prone model) are statistically significant. In addition to p-values, we report the average F1-score obtained with and without leakage, as well as the percentage of executions presenting performance overestimation, in order to provide insight into the magnitude and consistency of the observed effects.

Moreover, visual analysis was conducted by plotting the distributions of paired performance differences. Violin plots were used to illustrate both the distribution shape and the variability of the leakage effects, enabling a direct comparison of magnitude and dispersion across different pipeline steps, algorithms, and datasets.

Finally, clustering analysis was employed to identify common patterns in the impact of data leakage across each dimension analyzed, such as pipeline step or algorithm. For this purpose, we applied the k-means clustering algorithm using the paired differences in performance metrics as input features. The number of clusters was selected as $k = 4$ based on the elbow method, using inertia values, which are reported in Appendix C. Each dataset was represented by the average value of its paired performance differences, allowing comparisons between datasets within and across clusters.

5 RESULTS

This chapter presents a comprehensive analysis of how data leakage affects model performance in controlled experimental settings. All results are derived from paired executions, in which each model trained with data leakage is directly compared to an equivalent leakage-free model under identical conditions: same dataset, algorithm, pipeline step (and configuration), and random seed.

To support these analyses, each paired execution yields two performance estimates: one from a leakage-free model and one from its corresponding leakage-prone version. Although several performance metrics are extracted (Section 4.5), throughout this chapter, we quantify the impact of leakage using the paired difference in F1-score, denoted as $\Delta F1$, defined in Equation 5.1.

$$\Delta F1 = F1_{\text{leak}} - F1_{\text{correct}} \tag{5.1}$$

Positive values of $\Delta F1$ indicate that the leakage-prone pipeline inflated the estimated performance of the model relative to its corresponding leakage-free version. Our results will also discuss the mean difference in F1 score, denoted as $\overline{\boldsymbol{\Delta F1}}$, as well as the mean F1 scores for models trained with the data leakage pipeline and with the correct (*i.e.,* leakage-free) pipeline, denoted as $\overline{\boldsymbol{F1leak}}$ and $\overline{\boldsymbol{F1correct}}$, respectively.

Our investigation progresses from a high-level overview of leakage effects to increasingly granular analyses, as follows:

- Section 5.1 presents an analysis of the overall impact of data leakage.
- Section 5.2 provides a cluster-based analysis.
- Section 5.3 investigates whether the effects vary across different pre-training pipeline steps.
- Section 5.4 examines how the effects differ across algorithms.
- Section 5.5 analyzes the effects across different datasets.
- Section 5.6 explores the relationship between data leakage effects and data characteristics, including hardness measures.
- Section 5.7 investigates the interactions among datasets, algorithms, and pipeline steps.
- Section 5.8 presents a worst-case scenario analysis.

5.1 Overall impact of data leakage on performance estimates

The goal of this section is to provide an aggregated view of how data leakage affects model performance across all experiments, independent of dataset, algorithm, or pipeline step. By analyzing the overall distribution of paired F1-score differences, $\Delta F1$, we assess the general tendency of leakage to inflate performance estimates and quantify the typical and extreme magnitudes of this effect.

Table 5.1: Overall analysis of F1-scores in leakage-prone and leakage-free models across all executions (10,080 paired comparisons).

	$\overline{\boldsymbol{\Delta F1}}$	$\overline{\boldsymbol{F1leak}}$	$\overline{\boldsymbol{F1correct}}$	$\boldsymbol{Max\Delta F1}$	$\boldsymbol{Min\Delta F1}$	**p-value**
Overall	0.99%	79.21%	78.22%	86.24%	-33.88%	$p < 0.0001$

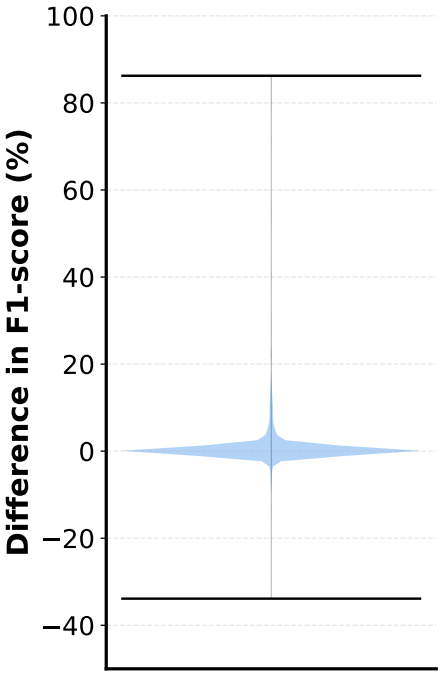
Table G.1 presents a summary of the overall results, considering the average F1-scores and the paired F1-score differences across all experiments, excluding the worst-case scenario. Across the 10,080 paired comparisons 4,040 had positive values for $\Delta F1$, 4,127 had no difference in the paired F1-score estimate, and 1913 experiments showed negative values for $\Delta F1$. Appendix D shows a plot of experiment executions with positive, negative and neutral values.

The mean difference in F1-score, $\overline{\Delta F1}$, was 0.99%. This positive value indicates that, on average, the deliberate introduction of data leakage leads to inflated performance estimates. Models trained under leakage-prone configurations achieved an average F1-score of 79.21%, compared to an average F1-score of 78.22% for leakage-free models.

Although the overall mean impact of data leakage, as summarized by $\overline{\Delta F1}$ in Table G.1, is relatively small, the associated p-value indicates that the difference between the performance estimate distributions of leakage-prone and leakage-free models is statistically significant ($p < 0.0001$), detailed results of the statistical analysis are presented in Appendix G. Moreover, the results reveal substantial variability across individual runs, with some paired executions exhibiting very large performance inflation, as shown in Figure 5.1. In particular, the maximum observed paired difference in F1-score, reported as $Max\ \Delta F1$, reaches 86.24% — an extremely large inflation that would create a strong and misleading impression of model effectiveness, which would not materialize under proper, leakage-free evaluation.

Examining the distribution of the paired differences in Figure 5.1, we can see that most of the differences are close to 0%, meaning that in several cases, the effect of data leakage was not very large. Despite that, there is a significant amount of paired differences

Figure 5.1: Distribution of F1-score paired differences across experiments.



Source: The Author

ranging from 0% to 20%, and even cases were the paired difference showed an increase of over 80% on the F1-Score, as already mentioned. The distribution also shows the occurrence of negative values. Negative differences occur when the leakage-prone configuration yields lower performance estimates than the leakage-free counterpart, which can be attributed to the stochastic nature of the learning process. Random variations in data partitioning, model initialization, and optimization dynamics may cause the additional information introduced by leakage to interact unfavorably with the training procedure in specific runs, leading to occasional performance degradation.

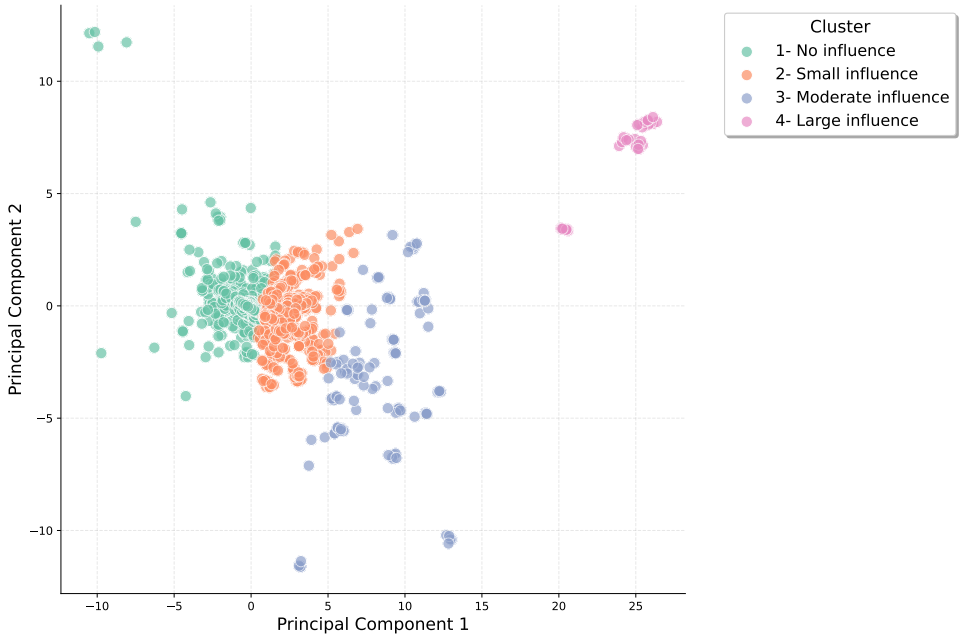
Despite this variability, the aggregation of results across a large number of paired executions shows a clear positive trend. It is important to note that negative and positive deviations partially cancel each other out in aggregated analyses; however, the remaining balance is consistently positive, resulting in an average increase in F1-score under the presence of data leakage in ML pipelines.

Main takeaway. On average, data leakage increased F1-score estimates by 0.99%, and in extreme cases the paired difference exceeded 80%. Overall, leakage leads to consistent and non-negligible overestimation across a large and diverse set of datasets, algorithms, and pre-training pipeline steps. While not every execution shows large inflation, the aggregated evidence is clear: data leakage tends to bias performance estimates upward, which can lead to misleading conclusions about model effectiveness.

5.2 Cluster-based analysis of data leakage effects

Given the large number of experimental configurations (over 10,000 paired comparisons), interpreting results individually is impractical and makes it difficult to identify broader behavioral patterns in how data leakage affects model performance. To address this challenge, we applied a k-means clustering algorithm that groups experiments according to the magnitude of the paired differences in performance metrics extracted (considering the metrics listed in Section 4.5), allowing us to uncover global structures in the results and to contextualize subsequent analyses by pipeline steps and algorithms. The optimal number of clusters was determined using the elbow method based on inertia; the supporting plots are presented in Appendix C. Following this procedure, we selected $k = 4$ as it provided a meaningful separation of patterns while avoiding overfragmentation.

Figure 5.2: PCA for generated clusters based on the perceived difference differences in paired performance metrics



Source: The Author

Figure 5.2 shows the PCA representation of the clusters, in which we can visually observe that four distinct groups were identified by the algorithm. The naming of the clusters was defined due to the perceived tendency of the impact of data leakage over the distributions of paired performance differences within each cluster, as summarized in Table 5.2.

The resulting four clusters reveal distinct behaviors regarding the impact of data leakage. Cluster *No Influence* (n = 9188) concentrates instances with the lowest aver-

Table 5.2: Statistical summary of paired performance differences within each cluster.

		1- No influence	2- Small influence	3- Moderate influence	4- Large influence
Δ F1-Score	mean	0.049	5.346	23.888	79.825
	std	1.329	4.386	14.619	5.026
	min	-33.876	-6.336	-4.057	70.518
	max	14.046	21.705	56.272	86.238
Δ Accuracy	mean	0.019	4.026	16.534	5.625
	std	1.280	3.111	8.931	2.674
	min	-46.093	-3.743	1.577	3.352
	max	6.683	12.500	39.313	10.973
Δ Balanced Accuracy	mean	0.123	5.166	13.645	38.461
	std	0.939	5.274	8.716	6.087
	min	-11.220	-2.177	-0.617	26.450
	max	27.547	35.150	30.602	43.757
Δ Precision	mean	0.102	7.604	23.221	76.274
	std	1.773	9.062	14.837	4.331
	min	-50.000	-10.712	-0.768	68.714
	max	17.846	77.083	59.238	81.946
Δ Recall	mean	0.134	5.331	15.302	78.061
	std	2.166	6.220	9.933	11.112
	min	-58.400	-10.000	-3.313	56.517
	max	31.111	43.166	53.279	87.785

age difference margins between the paired executions of models with and without data leakage, with a mean of close to 0% in all metrics, and a standard deviation between 1%-2%, it concentrates the bulk of experiments with differences close to 0%, or with small increases/decreases in performance. Clusters *Small Influence* (n=724), *Moderate Influence* (n=138) and *Large Influence* (n=30) contain the instances with more significant differences between the paired tests executions. We can see a relevant average F1-score increase of 5.346% in the *Small Influence* cluster, and very large mean F1-score differences of 23.888% and 79.825% for the *Moderate influence* and *Large influence* clusters, respectively.

Main takeaway. The cluster based analysis highlighted distinct behavior across our experiments. With groups of executions having different levels of perceived impact from data leakage insertion. The clusters for small, moderate and large influence showed that performance estimates clearly can be affected by data leakage and that the effect can vary drastically.

5.3 Effects across different pre-training pipeline steps

In this section, we examine how the impact of data leakage differs depending on where it occurs in the pipeline. Each step—normalization, missing-value imputation, feature selection, class balancing with SMOTE, and hyperparameter tuning—is evaluated

Table 5.3: Analysis of F1-score differences between leakage-prone and leakage-free models aggregated by pipeline step. Dark green means that $\Delta F1$ is positive and greater than 1%, light green means that $\Delta F1$ is positive but lower than 1%, and white means that the p-value shows no statistically significant difference for the comparison (*i.e.,* p > 0.05).

Pipeline step	$\overline{\boldsymbol{\Delta F1}}$	$\overline{\boldsymbol{F1leak}}$	$\overline{\boldsymbol{F1correct}}$	$\boldsymbol{Max\Delta F1}$	$\boldsymbol{Min\Delta F1}$	**p-value**
Class balanc. (SMOTE)	3.97%	85.42%	81.44%	86.24%	-16.33%	$p < 0.0001$
Feature selection	0.54%	66.50%	65.96%	42.27%	-33.88%	$p < 0.0001$
Hyperparam. tuning	0.43%	83.29%	82.87%	22.13%	-8.43%	$p < 0.0001$
Normalization	0.02%	80.95%	80.93%	12.22%	-14.66%	0.0157
Missing-value imputation	0.00%	79.90%	79.90%	15.57%	-16.96%	0.7092

independently, allowing us to determine which operations are most prone to severe leakage effects and which tend to have milder or negligible impact.

The results of this analysis are summarized in Table 5.3. To facilitate interpretation, we adopt the following color scheme for table rows: dark green indicates an average influence over F1-score (*i.e.,* $\overline{\Delta F1}$) positive and greater than 1%, light green indicates an influence over 0%, and white denotes cases with no statistically significant difference (*i.e.,* p > 0.05). Also, whenever negative $\overline{\Delta F1}$ values appear in a given step, the row is colored light yellow. The *Max* and *Min* $\Delta F1$ columns report the largest positive and largest negative paired differences observed within each pipeline step category.

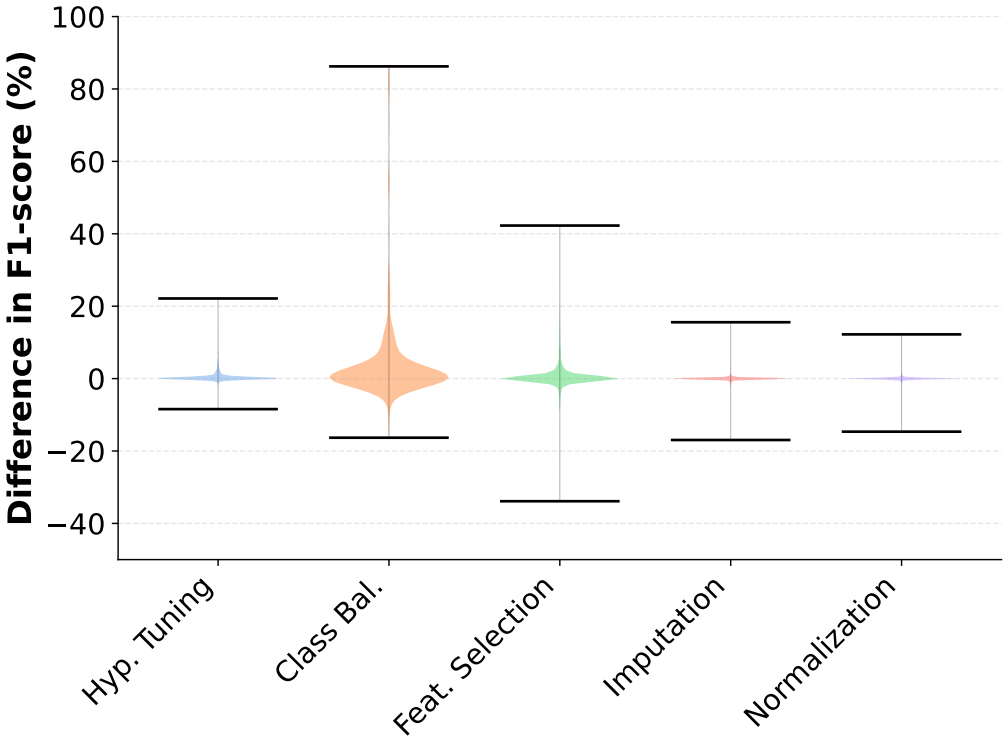
Our results show that the magnitude of performance distortion caused by data leakage varies substantially depending on the pipeline step where leakage was introduced. In terms of mean difference in performance, class balancing using SMOTE showed the strongest effect and a meaningful overestimation in performance, with a $\overline{\Delta F1}$ of almost 4%. Feature selection and hyperparameter tuning yielded moderate impacts, with an average F1-score increase close to 0.5%, while the normalization experiments showed a slight average increase of 0.02%. Experiments involving missing-value imputation did not exhibit statistically significant differences between leakage-prone and leakage-free models, with $\overline{\Delta F1}$ equals to 0%. Except for imputation, all other pipeline steps showed statistically significant paired differences ($p < 0.05$), confirming that even small implementation mistakes in these steps can lead to systematic overestimation of model performance.

However, because averaging can mask opposing effects, canceling out large positive and negative impacts, it is also important to examine the extreme cases of performance impact due to data leakage and the overall distribution of its effects for each pipeline step. The values of $Max\Delta F1$ in Table 5.3 also show different degrees of impact according to the pipeline step in which leakage was introduced. As seen previously, class balancing produced the the largest inflation, with a maximum paired increase of 86.24%, followed

by feature selection with a maximum increase of 42.27%. This means that, in some cases, a model contaminated by leakage appeared dramatically better than its true performance, turning a mediocre or even poor model into one that seems nearly perfect under a flawed model development pipeline. Hyperparameter tuning and normalization also showed relevant maximum increases of 22.13% and 12.22% on specific executions. This highlights the facts that even through the pipeline steps that lead to the smaller average impact, data leakage can lead to significant overestimation in specific cases. It is also possible to see that the values of $Min\Delta F1$a indicate that data leakage introduces a lot of variability of the model, occasionally harming rather than inflating performance, further reinforcing its destabilizing influence.

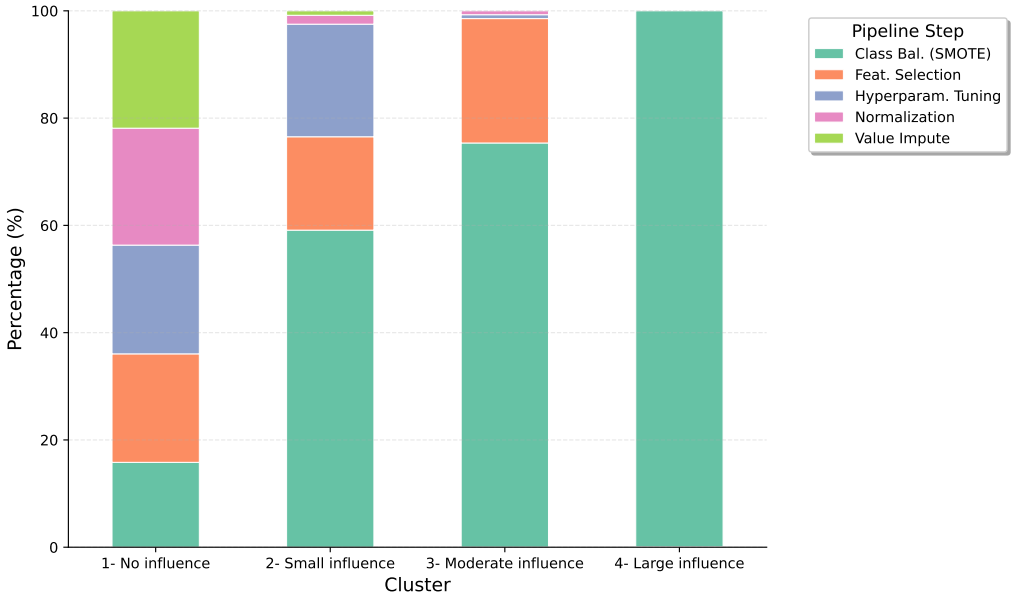
The distribution of paired performance differences, shown in Figure 5.3, makes the discrepancy among pipeline steps even more evident. Class balancing yields both the largest individual inflations and the greatest concentration of positive differences, with many runs showing F1-score increases in the 0%–20% range. Hyperparameter tuning exhibits a smaller but still noticeable proportion of mild inflations. The distribution of the differences for feature selection shows that it inserts a lot of variability into the model, with values ranging from -30% to over +40%. The distributions for missing-value imputation and normalization align with the earlier analyses: their effects are generally small and concentrated near zero, though still capable of producing occasional outliers

Figure 5.3: Distribution of F1-score paired differences by pipeline step.



These results are also consistent with the distribution of pipeline steps across clusters. Figure 5.4 contains a stacked bar plot with the percentages of experiments related to each pipeline step within each cluster. It is clear that the cluster with *large influence* instances is predominantly composed by instances where data leakage was introduced via SMOTE. For the *moderate influence* cluster, the most common pipeline steps are SMOTE and feature selection. The *small influence* cluster shows a majority presence of SMOTE, followed by hyperparameter tuning and feature selection. This cluster also contains a small number of instances related to normalization.

Figure 5.4: Proportion of experiments from each pre-training pipeline step across leakage-impact clusters.



Source: The Author

We note that in the case of missing-value imputation and feature selection, the discussions in this section are focused in the median imputer considering 20% missing values and the selection of the top 10% features ranked by the `f_classif` scoring function. Our analysis of different percentages of artificially introduced missing values and different imputation strategies did not reveal any substantial differences for the selected set of 30 datasets. A similar pattern was observed for the percentage of features selected: varying the percentile thresholds did not lead to significant changes in the overall magnitude or distribution of leakage effects. Detailed results for these experiments are provided in Appendix E.

Main takeaway. The effects of data leakage depend strongly on the specific pre-training pipeline step in which it occurs. Class balancing showed the largest impact, with a $Max\Delta F1$ of 86.24%. On average, models contaminated during this step outperformed

their leakage-free counterparts by 3.97%. Feature selection and hyperparameter tuning also posed substantial risks: although their average increases were modest (0.54% and 0.43%, respectively), they produced large overestimations in specific executions, reaching $Max\Delta F1$ values of 42.27% and 22.13%. Normalization and value imputation exhibited minimal average effects but still yielded distortions of up to 10% in individual cases.

5.4 Effects across learning algorithms

Table 5.4: Analysis of F1-score differences between leakage-prone and leakage-free models aggregated by learning algorithm. Dark green means that $\Delta F1$ is positive and greater than 1%, light green means that $\Delta F1$ is positive but lower than 1%.

Algorithm	$\overline{\Delta F1}$	$\overline{F1leak}$	$\overline{F1correct}$	$Max\Delta F1$	$Min\Delta F1$	p-value
KNN	1.26%	79.66%	78.40%	71.49%	-7.30%	$p < 0.0001$
NN	1.19%	82.67%	81.48%	81.35%	-33.88%	$p < 0.0001$
XGB	1.07%	84.62%	83.56%	86.24%	-7.62%	$p < 0.0001$
DT	1.06%	79.60%	78.54%	80.77%	-8.43%	$p < 0.0001$
RF	1.02%	84.31%	83.29%	85.48%	-5.53%	$p < 0.0001$
NB	0.81%	69.68%	68.87%	56.27%	-31.32%	$p < 0.0001$
SVC	0.80%	76.62%	75.82%	55.19%	-14.66%	$p < 0.0001$
LR	0.73%	76.54%	75.80%	52.91%	-7.72%	$p < 0.0001$

This section evaluates whether learning algorithms differ in their susceptibility to performance inflation caused by data leakage and whether model selection decision can be impacted under the occurrence of this methodological flaw. Algorithms vary widely in how they exploit training data, adapt decision boundaries, and generalize under small perturbations, which may modulate the degree of leakage-induced distortion of performance estimates. We therefore compared paired performance differences for each algorithm to determine whether some models are inherently more fragile under leakage-prone pipelines.

Table 5.4 summarizes the results of this analysis. When examining the effects of data leakage separately for each learning algorithm, small variations emerge, but the general trend suggests that all eight evaluated algorithms exhibited measurable performance inflation due to data leakage. The average increase in F1-score ranged from 0.73% to 1.26%, with KNN showing the largest mean effect and LR the smallest. For every algorithm, p-values indicate statistically significant differences between leakage-prone and leakage-free performance estimates.

The observation of $Max\Delta F1$ values (Table 5.4) further highlight the magnitude of potential distortion due to data leakage: large increases were perceived for all algorithms in at least some runs, with values ranging from 52.91% (LR) to 86.24% (XGB) – all very significant increases that would lead to unreproducible results. Figure 5.5 shows the distribution of F1-score differences for each of the tested algorithms. We can note that all algorithms systematically exhibited a tendency toward positive $\Delta F1$, although a large number of executions show differences close to 0%.

Figure 5.5: Distribution of F1-score paired differences by learning algorithm.

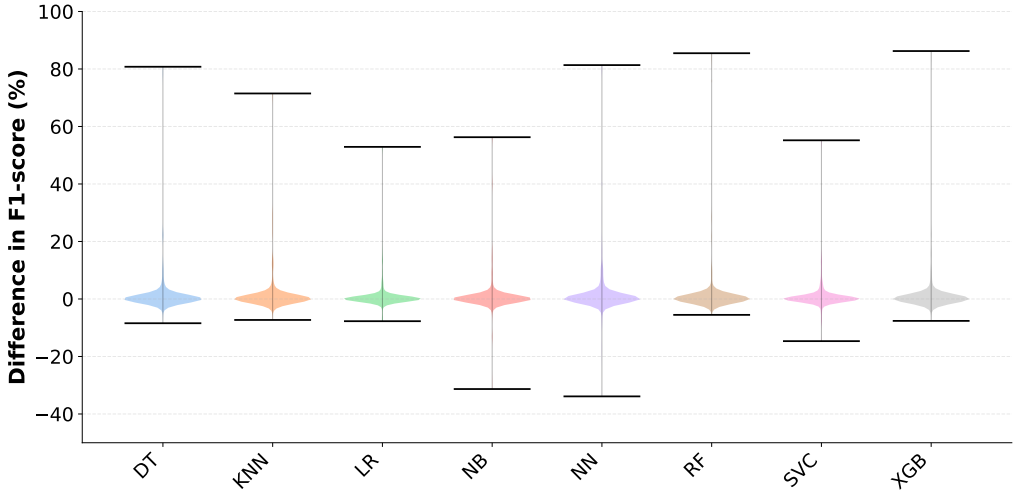
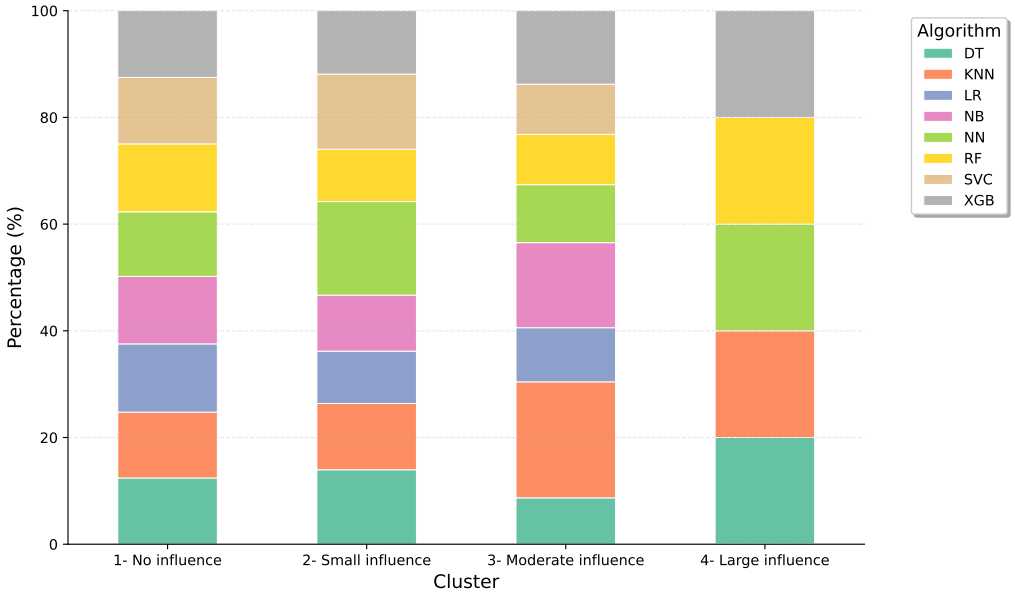


Figure 5.6 provides the relative frequency of learning algorithms within each leakage-impact cluster (introduced in Section 5.2), reinforcing the conclusions drawn from the paired-difference analysis. For the clusters with slighter influences of data leakage, *i.e., no influence* and *small influence*, there is no noticeable difference between the representation of algorithms. For the *moderate influence* cluster, we observe a slightly increase in the represetation of KNN in relation to the other algorithms and to the others clusters distributions. Finally, the three algorithms with the smallest average leakage impact (*i.e.,* SVC, LR, and NB) are absent from the *large influence* cluster, whereas algorithms such as KNN, NN, DT, RF, and XGB appear more frequently in this cluster associated with more substantial overestimation.

Beyond performance inflation, we investigated another important impact of data leakage, referring to model selection decision. We analyzed the proportion of executions in which the leakage-prone pipeline resulted in a distinct conclusion regarding the top-performing algorithm in relation to the corresponding leakage-free model. This phenomenon, which we refer to as *winner flip*, indicates situations where an algorithm appears superior solely due to contamination in the evaluation pipeline. Table 5.5 reports

Figure 5.6: Proportion of experiments from each learning algorithm across leakage-impact clusters.



Source: The Author

the proportion of runs in which the identity of the best-performing algorithm (under a correct pipeline) changes when leakage is introduced. We can see that considering all pipeline steps, there was a 'flip' on the best evaluated algorithm on 12.86% of experiments. This effect is most pronounced when we restrict the analysis for the executions with data leakage inserted during class balancing (23.81% of winner flip), hyperparameter tuning (16.67% of winner flip), and feature selection (14.29% of winner flip), which are the pipeline steps that exhibited the highest average and maximum inflation in performance estimates for leakage-prone models.

Table 5.5: Percentage of experiments in which data leakage changed the identity of the best-performing algorithm (*winner flip*) for overall experiments and for each pipeline step.

Pipeline step	**Percentages of top-performing model change**
All	12.86%
Class Balancing	23.81%
Hyperparameter Tuning	16.67%
Feature Selection	14.29%
Imputation	4.76%
Normalization	4.76%

Main takeaway. Data leakage affects all evaluated algorithms, producing average F1-score inflation between 0.73% and 1.26%, and substantial maximum distortions (52.91–86.24%). While LR, SVC, and NB are comparatively less sensitive to leakage, they are still affected. Critically, leakage does not merely inflate performance values, but

it may change decision regarding best model selection, causing a winner flip in 12.86% of experiments. Thus, leakage can mislead both the assessment of individual algorithms and the comparative choice of the "best" model for a dataset.

5.5 Effects across datasets

This section examines how the magnitude of leakage-induced performance inflation varies across datasets. Because datasets differ in dimensionality, class balance, separability, and several other structural characteristics, the sensitivity to leakage is expected to be heterogeneous. By analyzing paired performance differences for each dataset, we aim to identify which datasets are particularly prone to substantial overestimation and which remain relatively robust.

A summary of the F1-scores differences across the 42 different datasets is depicted in Table 5.6. Looking at the $\overline{\Delta F1}$, we can clearly see the effects changing according to the dataset. Out of the 42 datasets analyzed, 12 exhibited an $\overline{\Delta F1}$ larger than 1% (dark green rows), 23 had $\overline{\Delta F1}$ positive but lower than 1% (light green rows), three had negative mean differences (yellow rows), and four showed no statistically significant difference (p-value > 0.05) between leakage-prone and leakage-free evaluations (white rows).

The dataset with the largest mean performance difference ($\overline{\Delta F1}$) was *coil2000*, with an average overestimation of 14.33%. Other datasets also had relevant increases such as *connect_4*, with an average overestimation of 3.47%. Figure 5.7 shows the distribution of performance differences for the six datasets with the highest values of $\overline{\Delta F1}$, including the two previously discussed. To complement this view, Figure 5.8 presents the distributions for six additional datasets selected based on their highest $Max\Delta F1$ values, excluding those already included in the top-mean group. Together, these figures highlight both datasets that are consistently sensitive to leakage (high average inflation) and those that exhibit extreme but less frequent spikes in performance overestimation. The plots for the remaining datasets are presented in Appendix F.

Table 5.6: Analysis of F1-score differences between leakage-prone and leakage-free models aggregated by dataset. Row colors are encoded as follows: dark green indicates an average influence above 1%; light green indicates a positive influence below 1%; yellow marks negative average influence; and white denotes non-significant results (p-value > 0.05).

Dataset	$\overline{\Delta F1}$	$\overline{F1leak}$	$\overline{F1correct}$	$Max\Delta F1$	$Min\Delta F1$	**p-value**
coil2000	14.33%	23.16%	8.82%	86.24%	-7.14%	$p < 0.0001$
connect_4	3.47%	50.84%	47.36%	29.78%	-3.08%	$p < 0.0001$
sleep	3.02%	59.62%	56.60%	26.52%	-3.74%	$p < 0.0001$
shuttle	2.55%	75.98%	73.42%	40.49%	-8.64%	$p < 0.0001$
phoneme	2.38%	68.65%	66.28%	14.43%	-0.59%	$p < 0.0001$
krkopt	2.35%	48.37%	46.02%	23.47%	-0.77%	$p < 0.0001$
GAMETES_Epistasis...	2.18%	51.33%	49.15%	19.97%	-3.25%	$p < 0.0001$
magic	1.33%	67.95%	66.61%	11.37%	-6.60%	$p < 0.0001$
soybean	1.22%	84.49%	83.27%	8.27%	-0.47%	$p < 0.0001$
sonar	1.10%	75.96%	74.86%	17.04%	-7.62%	$p < 0.0001$
splice	1.06%	84.43%	83.37%	14.22%	-3.03%	$p < 0.0001$
molecular_biology...	1.02%	81.54%	80.52%	19.63%	-6.19%	$p < 0.0001$
clean2	0.95%	93.70%	92.75%	17.68%	-2.01%	$p < 0.0001$
movement_libras	0.93%	65.73%	64.81%	20.36%	-1.98%	$p < 0.0001$
dna	0.90%	91.36%	90.46%	12.11%	-0.44%	$p < 0.0001$
satimage	0.85%	84.14%	83.29%	6.64%	-0.30%	$p < 0.0001$
spambase	0.59%	88.11%	87.52%	5.84%	-7.30%	$p < 0.0001$
banana	0.57%	62.51%	61.94%	27.45%	-7.72%	$p < 0.0001$
Hill_Valley_with_noise	0.46%	52.29%	51.83%	42.27%	-33.88%	0.0135
optdigits	0.44%	91.78%	91.33%	5.90%	-0.31%	$p < 0.0001$
analcatdata_authorship	0.40%	97.35%	96.95%	4.68%	-0.75%	$p < 0.0001$
waveform_21	0.34%	78.07%	77.73%	7.12%	-0.63%	$p < 0.0001$
clean1	0.33%	94.91%	94.59%	7.80%	-3.64%	$p < 0.0001$
twonorm	0.21%	91.05%	90.83%	2.51%	-0.88%	$p < 0.0001$
spectf	0.21%	86.73%	86.52%	7.42%	-3.48%	0.0227
mfeat_factors	0.14%	94.06%	93.92%	10.93%	-1.51%	$p < 0.0001$
mnist	0.14%	85.99%	85.85%	1.24%	-3.10%	$p < 0.0001$
letter	0.07%	74.00%	73.93%	0.73%	-0.51%	$p < 0.0001$
waveform_40	0.07%	79.54%	79.47%	1.38%	-0.79%	$p < 0.0001$
mfeat_fourier	0.06%	79.72%	79.66%	1.08%	-0.89%	0.0042
mushroom	0.02%	95.06%	95.04%	1.00%	-0.06%	$p < 0.0001$
mfeat_karhunen	0.01%	91.72%	91.71%	0.95%	-1.27%	0.0180
texture	0.01%	92.29%	92.28%	0.20%	-0.07%	0.0360
agaricus_lepiota	0.01%	94.83%	94.83%	0.28%	-0.41%	0.0396
chess	0.01%	92.83%	92.83%	0.59%	-2.40%	$p < 0.0001$
kr_vs_kp	-0.01%	92.68%	92.68%	0.75%	-2.26%	0.0006
ring	-0.07%	84.02%	84.09%	7.95%	-7.84%	0.0112
adult	-0.88%	87.40%	88.28%	5.55%	-14.66%	$p < 0.0001$
Hill_Valley_without_noise	0.23%	61.30%	61.07%	30.88%	-16.96%	0.8917
tokyo1	-0.05%	87.94%	87.98%	6.66%	-4.76%	0.7115
mfeat_pixel	-0.36%	90.75%	91.11%	0.97%	-8.01%	0.1830
dis	-0.88%	92.75%	93.63%	0.55%	-31.32%	0.1194

Figure 5.7: Distribution of F1-score paired differences for datasets with highest $\overline{\Delta F1}$.

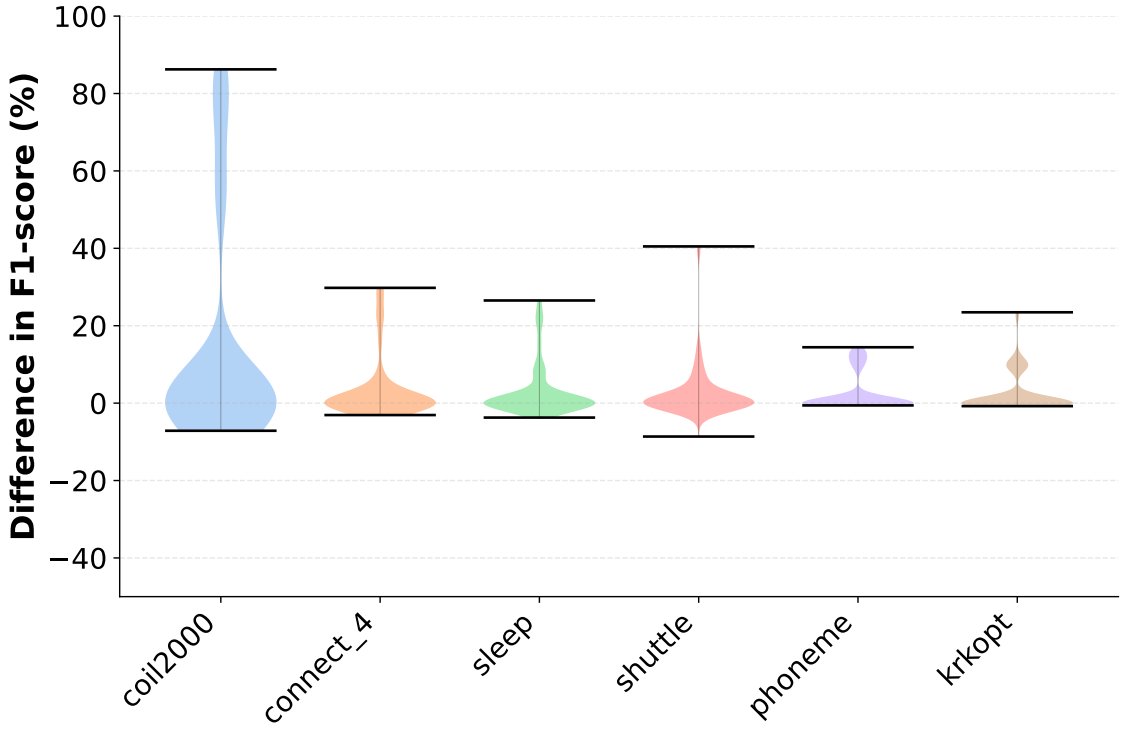
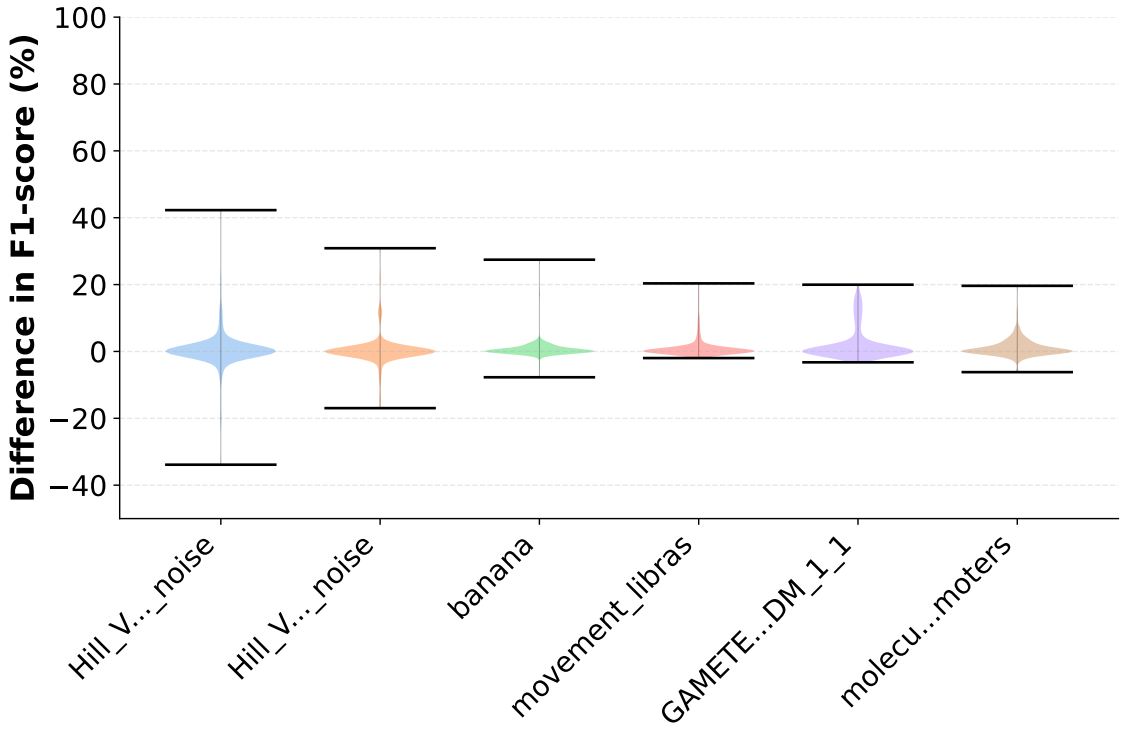


Figure 5.8: Distribution of F1-score paired differences for six additional datasets with highest $Max\Delta F1$.



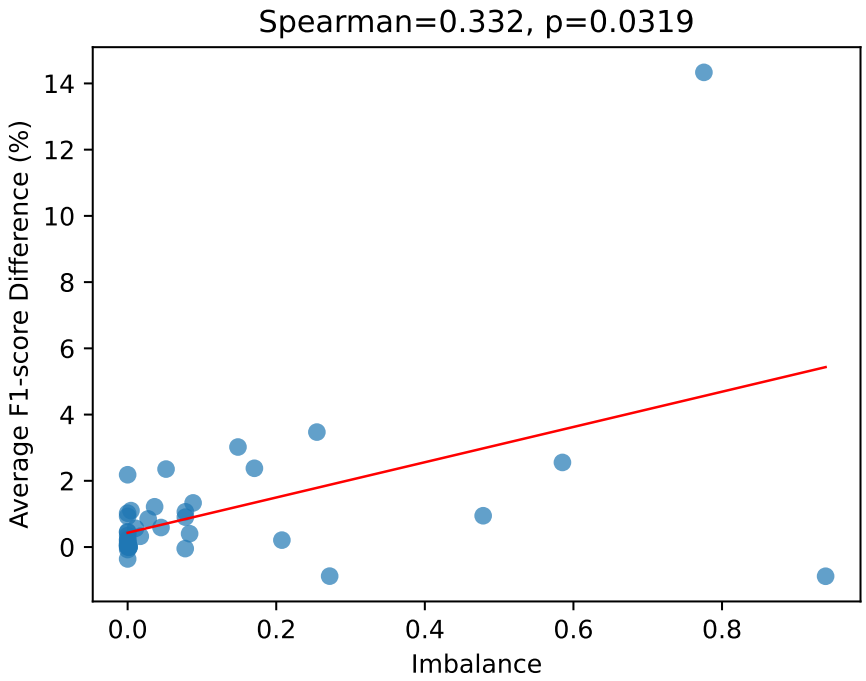
Main takeaway. Models trained for distinct datasets had different sensitivity for data leakage insertion. Across the 42 datasets, 35 showed positive $\overline{\Delta F1}$ values, indicating that leakage typically inflates performance, while 12 of these datasets exhibited average

overestimation above 1%. The $Max\Delta F1$ value ranged from 0.2% to 86.26%, while $\overline{\Delta F1}$ varied from -0.88% to 14.33%. Overall, these results suggest that the degree to which leakage distorts model evaluation is highly dataset-dependent.

5.6 Relationship between dataset hardness and leakage effects

To better understand if the characteristics of the data had influence on the impact of data leakage, we conducted two analysis, one based on the datasets metadata, as extracted or derived from PMLB, and a second based on the hardness measures. Figure 5.9 shows the correlation between the class imbalance and the effects of data leakage (given by $\overline{\Delta F1}$). This metric was the only one that presented statistically significant difference (p-value = 0.0319) in the correlation analysis based on metadata, which suggests that class imbalance has a moderate positive correlation with the impact of data leakage. This result is likely related to the high impact of the class balancing step, as discussed in Section 5.3. The remaining distributions and calculated correlations are presented in Appendix H.

Figure 5.9: Correlation between dataset class imbalance and mean F1-score difference.



Regarding the analysis of hardness measures, the correlation and p-values presented in Table 5.7 show that six evaluated hardness measures had a weak to moderate positive correlations with the mean F1-score differences: MV, DS, kDN, N1, U, and LSC. F2 had a p-value just slightly above the significance threshold (p=0.0559). These correla-

tions indicate a slight tendency of datasets with harder features to perceive larger effects of leakage.

Table 5.7: Spearman correlation among average hardness measures of datasets and their mean F1-score differences comparing leakage-prone and leakage-free models.

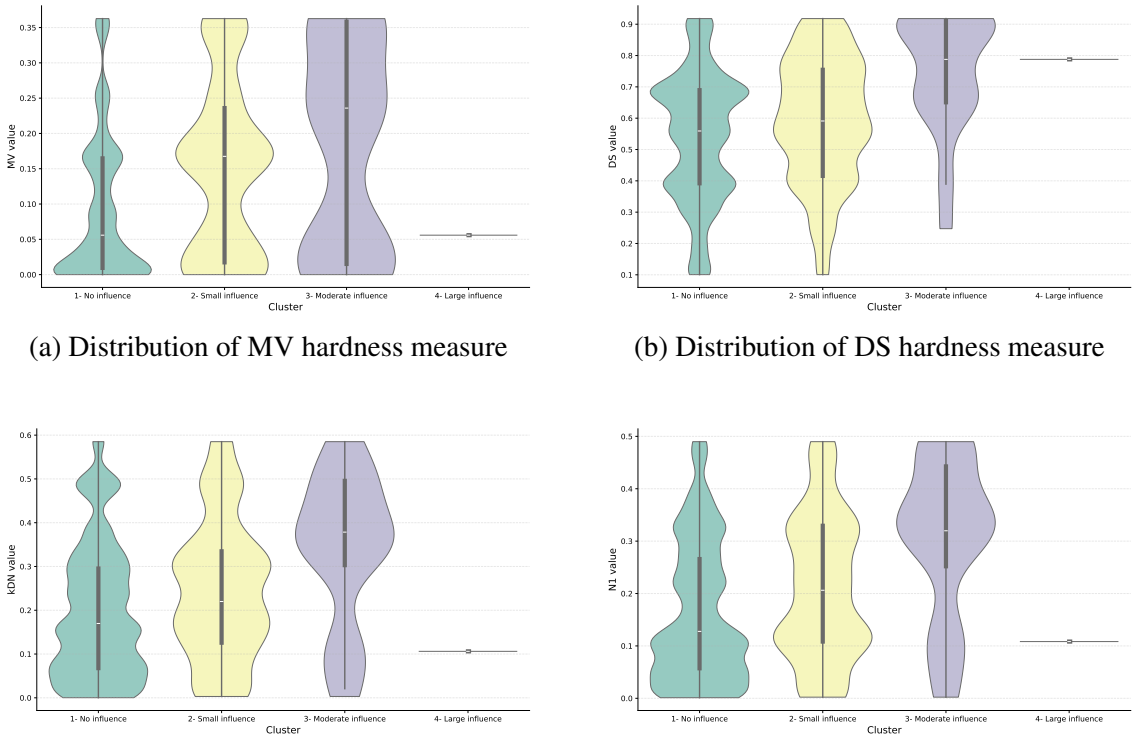
Hardness measure	**Spearman Correlation**	**p-value**
MV	0.462	0.0021
DS	0.375	0.0145
kDN	0.369	0.0163
N1	0.364	0.0179
U	0.339	0.0283
LSC	0.338	0.0288
F2	0.297	0.0559
DCP	0.290	0.0623
CLD	0.235	0.1337
CL	0.229	0.1451
F1	0.210	0.1817
N2	0.168	0.2867
TD_P	0.167	0.2909
H	-0.138	0.3829
CB	-0.127	0.4245
LSR	0.120	0.4481
F3	0.065	0.6819
TD_U	0.054	0.7335
F4	0.003	0.9846

Figure 5.10 shows the distributions for a few selected hardness measures across the impact-based clusters (defined in Section 5.2). These plots consider the average hardness measures per dataset. The other hardness measures showed slight differences in the distributions, but not at the same level as the ones presented in this section. The distributions for all hardness measures are available in Appendix I, while the complete list of datasets and their average values for the hardness measures is provided in appendix J.

Figures 5.10a (MV), 5.10b (DS), 5.10c (kDN), and 5.10d (N1) show examples where the distributions of hardness measures present noticeable differences across clusters. We can see that for the *Large Influence* cluster, all values are constant. This is due to the fact that all instances in that cluster correspond to the same dataset, *coil2000*. The MV violin plot shows difference in the distribution, with the cluster for small and moderate influence showing a higher median and proportionally more instances with high values for this hardness measure.

The changes in DS, kDN, and N1 are even more pronounced. These hardness measures show substantially higher values for the *Moderate Influence* cluster relative to both

Figure 5.10: Distributions of selected hardness measures across clusters.



(a) Distribution of MV hardness measure

(b) Distribution of DS hardness measure

(c) Distribution of kDN hardness measure

(d) Distribution of N1 hardness measure

Source: The Author

the *No Influence* and *Small Influence* clusters, indicating that datasets situated in more difficult regions of the hardness space tend to be more susceptible to leakage-induced overestimation. Overall, these patterns suggest that dataset difficulty, as captured by certain hardness measures, may amplify the magnitude of leakage effects, particularly for datasets whose instances are harder to classify or lie near decision boundaries.

Main takeaway. Most dataset metadata showed no meaningful correlation with leakage impact, except for class imbalance, which exhibited a moderate positive association, likely driven by the strong effects of SMOTE-based class balancing. The analysis of hardness suggested that datasets with higher intrinsic difficulty (higher MV, DS, kDN, and N1, for example) appeared more often in clusters with higher leakage-induced inflation. Together, these findings suggest that harder and more imbalanced datasets are more vulnerable to substantial overestimation when leakage occurs.

5.7 Analysis of pipeline step, algorithm and dataset interactions

While previous sections analyze factors individually, such as dataset, algorithm, and pipeline step where leakage was introduced, this section explores their interactions. Our aim is to identify combinations of pre-training pipeline steps, datasets, and algorithms that produce particularly strong or particularly mild leakage effects. The heatmap in Figure 5.11 depicts the $\overline{\Delta F1}$ for combinations of learning algorithms and pipeline steps.

Figure 5.11: Heatmap of $\overline{\Delta F1}$ for combinations of algorithms and pipeline steps.



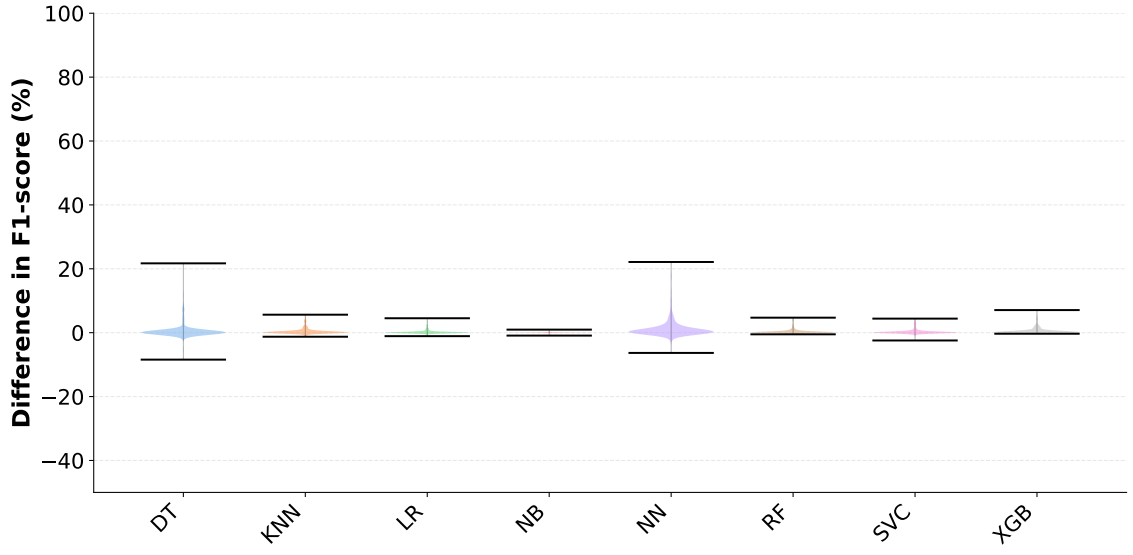
Source: The Author

The combination with the highest average increase was when the KNN model had data leakage introduced through SMOTE-based class balancing, with 5.75% increase. This overestimation is more than 1% higher than any other algorithm under the same step. This suggests that the synthetic minority samples generated by SMOTE disproportionately distort the neighborhood structure exploited by KNN. Because KNN relies directly on the local geometry of the feature space, by looking at the features for the nearest instances, incorporating synthetic samples created using information from the full dataset (including test folds, when leakage occurs) reshapes decision boundaries in a way that unrealistically favors the model. As a result, KNN becomes especially prone to optimistic overestimation under contaminated SMOTE pipelines. While all algorithms were impacted by class balancing, LR and SVC suffered the lowest impact.

Another relevant combination is between hyperparameter tuning and the NN algorithm. The average increase of over 1% is significantly higher than the average for the same step when optimizing for other algorithms. This is likely explained by the high sensitivity of neural networks to their hyperparameters: choices such as the number of layers, hidden-unit counts, activation functions, and learning rate can dramatically reshape the learned decision surface. When leakage is present, its leads the NN to adjust to patterns it would not have access to in a correct pipeline. Consequently, NNs become particularly susceptible to inflated performance estimates when hyperparameter tuning is contaminated by leakage.

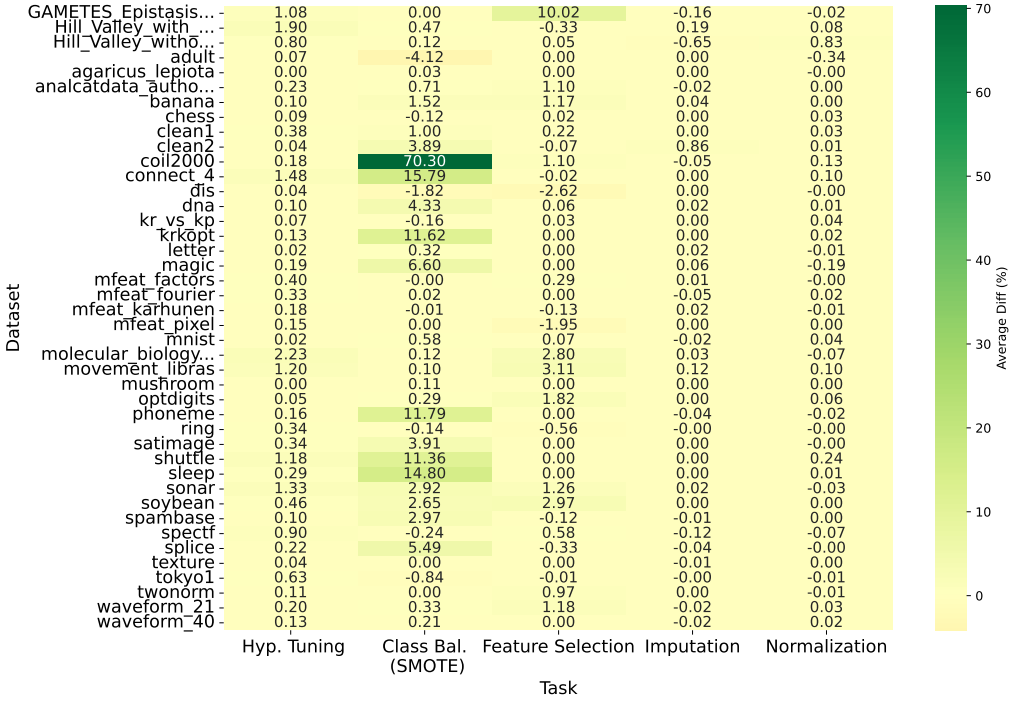
Figure 5.12 shows the distribution of mean F1 differences for each algorithm when considering only experiments for the hyperparameter tuning step. The distributions vary clearly for different algorithms, with NN and DT having higher variability in terms of F1-score differences and larger values of maximum impact caused by data leakage during this pipeline step.

Figure 5.12: Distribution of F1-score paired differences for algorithms considering only leakage introduced in the hyperparameter tuning step.



When examining the interaction between datasets and pipeline steps (Figure 5.13), we observe that the dataset with the greatest overall impact from data leakage shows that most of its difference comes from SMOTE, with an extremely high $\overline{\Delta F1}$ of 70.3%. It is also possible to see that the effects of pipeline steps varies significantly according to the dataset. For example, feature selection can produce $\overline{\Delta F1}$ values of up to 10%, depending on the dataset, while leakage introduced during hyperparameter tuning yields average increases exceeding 1%,and reaching more than 2% for some datasets.

Figure 5.13: Heatmap of $\overline{\Delta F1}$ for combinations of datasets and pipeline steps.



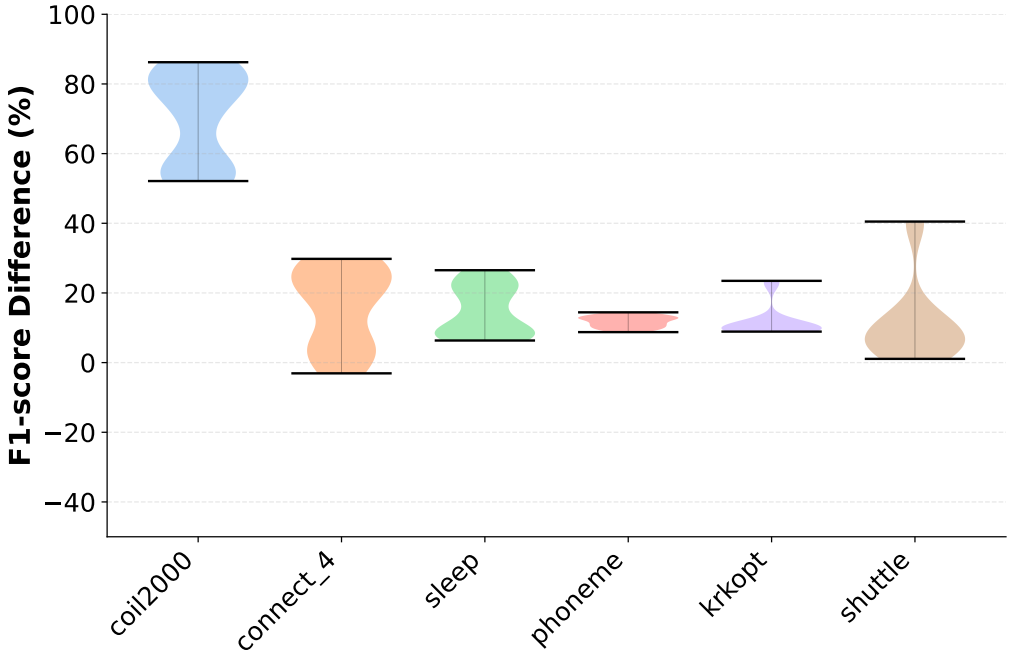
Source: The Author

The distribution of the datasets with the highest average impact seen in the class balancing pipeline step are presented in Figure 5.14. We can see that for this step, specific datasets had a very clear overestimation tendency,. Distributions for the most affected datasets in other pipeline steps are presented in appendix K.

Lastly, the heatmap of $\overline{\Delta F1}$ by dataset–algorithm combination (Figure 5.15) further highlights the variability of leakage effects. Different datasets exhibited their largest performance inflations with different algorithms. For instance, the *Gametes Epistases* dataset had the largest $\overline{\Delta F1}$ increase with the LR algorithm and the lowest increase with the DT algorithm. The *Connect 4* dataset, on the other hand, had it highest increase in $\overline{\Delta F1}$ with the DT algorithm, and its lowest increases with the LR and NB algorithms. These patterns reinforce that the interaction between dataset characteristics and model architecture plays a substantial role in determining the magnitude of leakage-induced overestimation.

Main takeaway. Data leakage effects vary considerably across algorithm-pipeline step and dataset–pipeline step combinations. Both NN and DT showed higher susceptibility to leakage during hyperparameter tuning compared to the other algorithms. Strong variations were also observed across dataset–algorithm and dataset–pipeline step combinations. No single algorithm or pipeline step consistently produced the highest impact

Figure 5.14: Distribution of F1-score paired differences for datasets with highest $\overline{\Delta F1}$ in the class balancing step.



across all datasets, indicating that leakage severity is strongly dependent on the specific characteristics of each dataset–algorithm pair.

5.8 Worst-case scenario analysis

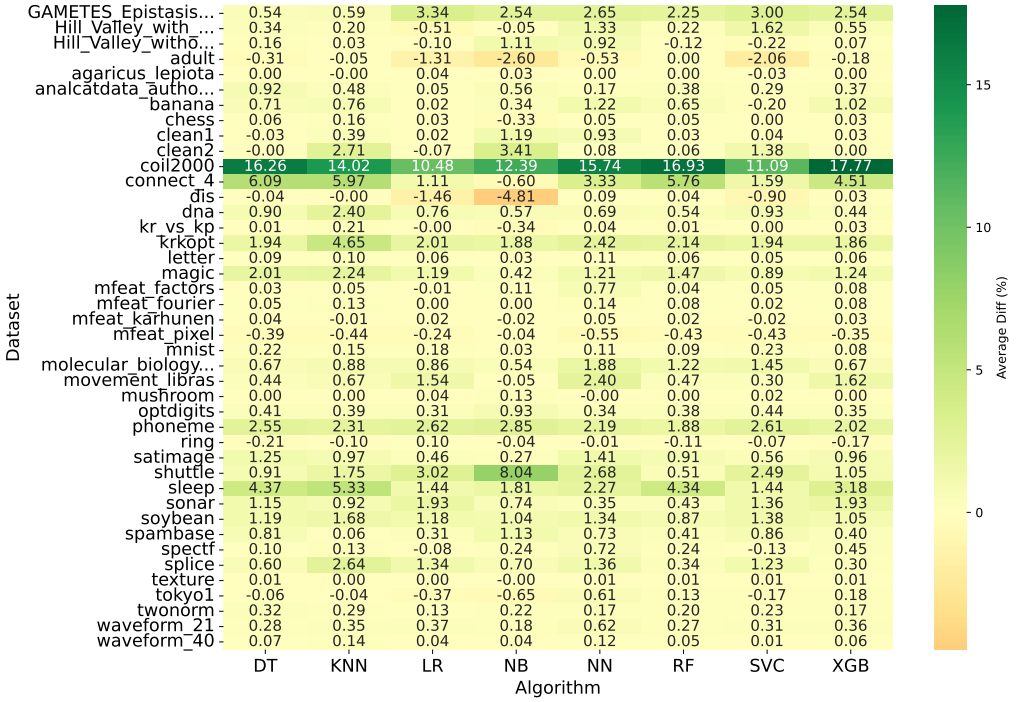
In addition to the experiments where data leakage was inserted through each of the individual model pre-training pipeline steps, we evaluated the case in which data leakage was simultaneously introduced through all of the steps investigated, what we refer to as the *worst-case* scenario. Table 5.8 shows the average effects caused by the worst-case leakage over performance estimates, as well as the maximum and minimum performance differences. The mean difference in performance, $\overline{\Delta F1}$, reached 4.08%, which is higher than what was perceived for any of the individual pipeline steps, as expected. We also notice that the maximum overestimation reached 79.25%, and the difference between the leakage-prone and leakage-free models remained statistically significant ($p < 0.0001$).

Table 5.8: Summary of F1-scores differences in models with and without leakage for all experiments in the worst-case scenario (n=2,016).

	$\overline{\Delta F1}$	$\overline{F1leak}$	$\overline{F1correct}$	$Max\Delta F1$	$Min\Delta F1$	p-value
Worst-case	4.08%	72.53%	68.45%	79.25%	-31.48%	$p < 0.0001$

When examining the dataset-level effects of this worst-case contamination, we

Figure 5.15: Heatmap of $\overline{\Delta F1}$ for combinations of datasets and algorithms.



Source: The Author

observe substantial performance inflation across many datasets: 24 datasets showed average increases greater than 1%, ten datasets exceeded 5%, and four datasets exhibited increases above 10%. The largest average increase was for the *coil200* dataset, with $\overline{\Delta F1}$ of 65.11%. Despite the large increases, the variability introduced by the worst-case leakage also resulted in decreases for six datasets. The most notable decrease occurred in the *magic* dataset, where the average F1-score dropped by 4.39%.

The $Max\Delta F1$ and $Min\Delta F1$ values also show interesting results. For *coil2000*, the maximum paired difference reached 79.25%, and even the minimum difference remained extremely high at 40.39%, thus indicating that this dataset had large increases in performance estimates for all executions. A similar behavior was seen for other datasets, such as *phoneme*, *soybean*, and *analcatdata_authorship*, but with lower values for $Max\Delta F1$ and $Min\Delta F1$. It is also important to notice that 13 datasets had maximum increases of over 10%, illustrating that compounded leakage can dramatically inflate performance estimates in a wide range of conditions.

An analysis of the effects of the worst-case scenario across the different algorithms, presented in Table 5.10, shows that all algorithms were affected to some degree and in a statistically significant manner. All algorithms had average increases greater than 3%. Models trained with NB, LR, and KNN had slightly lower $\overline{\Delta F1}$ compared to the other algorithms. The effects on the $Max\Delta F1$ was consistent with the results discussed

Table 5.9: Dataset-level analysis of F1-score differences between leakage-prone and leakage-free models under the worst-case scenario. Row colors are encoded as follows: dark green indicates an average influence above 1%; light green indicates a positive influence below 1%; yellow marks negative average influence; and white denotes non-significant results (p-value > 0.05).

Dataset	$\overline{\Delta F1}$	$\overline{F1leak}$	$\overline{F1correct}$	$Max\Delta F1$	$Min\Delta F1$	**p-value**
coil2000	65.11%	82.54%	17.42%	79.25%	40.39%	$p < 0.0001$
phoneme	16.84%	69.22%	52.38%	44.09%	6.41%	$p < 0.0001$
soybean	10.92%	53.21%	42.29%	14.44%	7.82%	$p < 0.0001$
GAMETES_Epistasis...	10.22%	60.00%	49.78%	24.93%	-3.63%	$p < 0.0001$
dis	7.17%	94.67%	87.50%	27.56%	0.65%	$p < 0.0001$
banana	6.48%	49.12%	42.63%	19.17%	-4.69%	$p < 0.0001$
sleep	6.15%	51.14%	44.98%	10.60%	-5.69%	$p < 0.0001$
adult	5.77%	88.47%	82.70%	8.10%	-0.24%	$p < 0.0001$
sonar	5.56%	73.91%	68.35%	14.36%	-13.65%	$p < 0.0001$
molecular_biology...	5.54%	85.26%	79.72%	18.28%	-2.61%	$p < 0.0001$
splice	4.79%	83.89%	79.10%	6.47%	1.29%	$p < 0.0001$
analcatdata_authorship	4.75%	96.65%	91.90%	9.27%	1.76%	$p < 0.0001$
connect_4	4.32%	44.62%	40.29%	16.29%	-6.07%	$p < 0.0001$
krkopt	3.78%	5.75%	1.98%	4.95%	0.00%	$p < 0.0001$
satimage	2.81%	82.06%	79.25%	4.90%	0.70%	$p < 0.0001$
spambase	2.50%	83.29%	80.79%	7.52%	-2.75%	$p < 0.0001$
clean2	2.38%	98.18%	95.79%	18.68%	0.00%	$p < 0.0001$
mfeat_pixel	2.12%	78.93%	76.81%	5.24%	-0.82%	$p < 0.0001$
dna	1.88%	94.42%	92.54%	3.73%	0.80%	$p < 0.0001$
waveform_21	1.63%	59.92%	58.29%	4.64%	-0.32%	$p < 0.0001$
Hill_Valley_with_noise	1.59%	41.51%	39.91%	15.09%	-14.39%	0.0123
spectf	1.59%	87.23%	85.65%	5.54%	-4.30%	$p < 0.0001$
movement_libras	1.12%	43.17%	42.05%	5.11%	-4.53%	0.0016
twonorm	1.05%	74.33%	73.28%	2.40%	-0.33%	$p < 0.0001$
clean1	0.66%	97.74%	97.08%	4.81%	-1.40%	0.0061
agaricus_lepiota	0.62%	83.73%	83.11%	1.94%	-0.95%	$p < 0.0001$
mfeat_fourier	0.57%	80.31%	79.74%	2.48%	-0.73%	$p < 0.0001$
waveform_40	0.42%	69.79%	69.37%	2.73%	-1.46%	$p < 0.0001$
optdigits	0.33%	75.30%	74.97%	2.11%	-1.15%	0.0013
shuttle	0.32%	40.76%	40.44%	3.13%	-0.46%	0.0031
mushroom	0.29%	92.14%	91.86%	1.23%	-0.74%	$p < 0.0001$
mnist	-0.06%	86.30%	86.35%	0.20%	-0.54%	0.0247
chess	-0.44%	85.65%	86.10%	14.62%	-7.43%	$p < 0.0001$
kr_vs_kp	-0.64%	81.45%	82.10%	0.55%	-2.75%	$p < 0.0001$
ring	-0.72%	72.96%	73.68%	0.66%	-2.30%	$p < 0.0001$
tokyo1	-1.01%	92.50%	93.51%	0.91%	-4.00%	$p < 0.0001$
magic	-4.39%	59.39%	63.78%	-2.12%	-8.42%	$p < 0.0001$
letter	0.04%	29.95%	29.91%	1.00%	-1.35%	0.1421
mfeat_karhunen	0.04%	87.61%	87.57%	0.81%	-0.55%	0.3152
texture	0.03%	81.49%	81.46%	0.57%	-0.54%	0.0576
mfeat_factors	-0.08%	90.81%	90.88%	0.52%	-0.95%	0.1660
Hill_Valley_without_noise	-0.71%	56.74%	57.45%	14.18%	-31.48%	0.6076

in Section 5.4, with SVC, LR, and NB showing the smallest maximum differences in the performance estiThe k-means clustering analysis was repeated using only the worst-case experiments, and the results confirmed earlier observations: SMOTE is associated with substantial influence on performance inflation, and although all algorithms may be affected, NB, SVC, and LR do not appear in the *Large Influence* cluster (Appendix L), indicating that they are comparatively less susceptible under combined leakage conditions.

The k-means clustering analysis was repeated using only the worst-case experiments, and the results confirmed earlier observations: SMOTE is associated with substantial influence on performance inflation, and although all algorithms may be affected, NB, SVC, and LR do not appear in the *Large Influence* cluster (Appendix L), indicating that they are comparatively less susceptible under combined leakage conditions.

The k-means clustering analysis was repeated using only the worst-case experiments, and the results confirmed earlier observations: SMOTE is associated with substantial influence on performance inflation, and although all algorithms may be affected, NB, SVC, and LR do not appear in the *Large Influence* cluster (Appendix L), indicating that they are comparatively less susceptible under combined leakage conditions.

Additionally, the analysis of changes in best-algorithm selection revealed an even stronger impact in the worst-case scenario compared to any individual pipeline step. A *winner flip* occurred in 40.48% of datasets, meaning that in nearly half of the datasets model selection decision would lead to a different "best" algorithm when worst-case leakage was introduced.

Table 5.10: Algorithm-level analysis of F1-score differences between leakage-prone and leakage-free models under the worst-case scenario. Dark green row color indicate an average influence above 1%.

Algorithm	$\overline{\Delta F1}$	$\overline{F1leak}$	$\overline{F1correct}$	$Max\Delta F1$	$Min\Delta F1$	p-value
RF	4.59%	75.25%	70.65%	79.14%	-3.61%	$p < 0.0001$
XGB	4.55%	75.44%	70.89%	79.11%	-4.15%	$p < 0.0001$
DT	4.35%	72.96%	68.61%	79.25%	-5.82%	$p < 0.0001$
NN	4.30%	75.11%	70.82%	72.63%	-4.69%	$p < 0.0001$
SVC	4.09%	73.64%	69.54%	58.89%	-31.48%	$p < 0.0001$
KNN	3.73%	72.09%	68.37%	72.38%	-7.43%	$p < 0.0001$
LR	3.66%	71.11%	67.45%	49.66%	-8.42%	$p < 0.0001$
NB	3.37%	64.61%	61.24%	41.29%	-14.39%	$p < 0.0001$

Main takeaway. The worst-case scenario produced the largest overall impact of data leakage, yielding the highest $\overline{\Delta F1}$ among all experiments. More than half of the datasets showed average increases above 1%, with 10 datasets exceeding 5% and 4

datasets surpassing 10%. When leakage was introduced across all pipeline steps simultaneously, the top-performing algorithm changed in 40.48% of datasets, highlighting a substantial risk of misleading model selection under contaminated evaluation pipelines.

6 EVIDENCE-BASED GUIDELINES FOR AVOIDING DATA LEAKAGE

This chapter presents a set of practical guidelines for preventing data leakage in ML pipelines, derived from the experimental findings of this work. The recommendations are grounded in empirical evidence obtained from our controlled evaluation across multiple datasets, algorithms, and pre-training pipeline steps, and their applicability should be interpreted within this experimental context, namely structured datasets and classical ML pipelines.

6.1 Rationale and scope

The experimental results demonstrate that data leakage can substantially inflate performance estimates, often by several percentage points and, in extreme cases, by tens of percentage points. These findings highlight that avoiding data leakage is not merely a matter of following standard validation protocols, but requires careful attention to how data preparation steps are integrated into the ML pipeline. The guidelines proposed in this section aim to translate the empirical observations of this work into actionable recommendations that can assist practitioners and researchers in designing leakage-aware experimental protocols. we note that the scope of these guidelines is limited to structured datasets and classical ML pipelines, and thus, their applicability should be interpreted within this experimental context. – although we imagine that similar effects would be observed for other forms of data leakage, within other scopes (such as regression, deep learning, among others).

6.2 High-risk data preparation steps

The experimental analysis indicates that not all data preparation steps pose the same risk of inducing data leakage. Among the evaluated steps, class balancing using SMOTE exhibited the largest average impact on performance inflation, with an average increase of 3.97% in F1-score across the evaluated datasets. This result suggests that resampling techniques represent a high-risk source of data leakage when improperly applied. Even though SMOTE works by creating synthetic data, it can still cause a wrong impression of good generalization power due to data leakage. Although not tested in the

current work, other forms of data oversampling are probably also susceptible to similar effects. These results are aligned with the observations of Wen et al. (2025) that class balancing through SMOTE can lead to severe overestimation when not following the correct methodology.

Class balancing methods should therefore be treated as critical operations and strictly confined to the training data within each cross-validation fold. Applying resampling procedures prior to data splitting or across pooled training and validation sets can lead to substantial contamination and overoptimistic performance estimates.

Feature selection and hyperparameter tuning also consistently led to inflated performance estimates when data leakage was introduced, with average F1-score increases of 0.54% and 0.43%, respectively. Although these effects are smaller in magnitude than those observed for class balancing, they were consistently observed across a large number of datasets. Moreover, for specific cases, methodological flaws within these pipeline steps caused a performance estimate inflation of up to 42.27% for feature selection and 22.13% for hyperparameter tuning. This indicates that both feature selection and hyperparameter tuning constitute systematic sources of leakage when not properly nested within the training process.

6.3 Medium- and low-risk steps

In contrast to the steps discussed above, normalization and value imputation did not exhibit large mean differences when leakage was introduced in isolation; moreover, statistically significant performance inflation was observed only for normalization. These results suggest that, for the evaluated datasets and algorithms, the isolated impact of these preprocessing steps on performance estimates is limited.

However, this finding should not be interpreted as an indication that normalization and value imputation are inherently safe operations. Rather, it indicates that their individual contribution to performance inflation appears to be smaller when compared to other data preparation steps, but can still be significant in some cases. Thus, these operations should still be performed exclusively using training data statistics and applied consistently within each cross-validation fold, particularly when combined with higher-risk steps.

6.4 Worst-case scenarios and compounding effects

The worst-case scenario analysis revealed that the combination of multiple sources of data leakage can lead to severe performance inflation. In this analysis, specific datasets exhibited average F1-score increases of 65%, 16%, and 10% when leakage was simultaneously introduced across all evaluated data preparation steps.

These results demonstrate that data leakage effects are not merely additive. Even steps that individually exhibit moderate or negligible impact can jointly produce substantial distortions when combined within the same pipeline. Therefore, preventing data leakage should not rely on isolated safeguards targeting individual steps, but rather on the consistent application of leakage-aware practices across all stages of the ML pipeline.

6.5 Practical recommendations

Based on the empirical findings of this work, the following practical recommendations are proposed to mitigate the risk of data leakage in machine learning pipelines:

- Perform class balancing and resampling techniques strictly within the training data of each cross-validation fold, never before data splitting.
- Ensure that feature selection and hyperparameter tuning are fully nested within the training process and isolated from test and validation data.
- Avoid reusing validation data across multiple pipeline stages without proper nesting, particularly during model selection.
- Treat combinations of data preparation steps as high-risk scenarios, even when individual steps appear to have limited impact when evaluated in isolation.
- Be diligent with what data is being used in which steps of your pipeline can prevent every form of leakage we examined. Always ask yourself if the data you are providing for a model, processing step or preparation task is only the one it requires, nothing else.
- Prefer pipeline-based implementations that explicitly enforce data separation across all preparation steps, reducing the likelihood of inadvertent contamination.

Together, these guidelines emphasize that preventing data leakage requires a systematic and holistic approach to pipeline design. By prioritizing high-risk steps and ac-

counting for compounding effects across pipeline stages, practitioners can significantly reduce the likelihood of performance inflation and improve the reliability and reproducibility of machine learning results.

7 CONCLUSION

This work presented a systematic experimental assessment of the effects of data leakage on the performance estimates of ML classifiers. By evaluating leakage inserted at different stages of the pre-training pipeline—normalization, value imputation, feature selection, class balancing, and hyperparameter tuning—across multiple algorithms and a large collection of structured datasets, the study provides quantitative evidence on how the severity and consistency of leakage effects vary across practical scenarios.

Regarding the overall magnitude and consistency of leakage effects, our results indicate that performance inflation is frequent but highly variable. While most of the datasets (*i.e.,* 35 out of the 42 analyzes datasets) exhibited increases in estimated performance, the magnitude of these increases ranged widely, from negligible differences to important shifts in average F1-score, reaching up to 80% of performance overestimation in some cases. The observed variability shows that the effect of data leakage is not uniform, but it is a real risk during model development and evaluation, causing significant changes in performance estimates.

With respect to the pre-training pipeline, the results revealed clear differences among the investigated steps. Among the individual steps, class balancing using SMOTE had the strongest impact, with mean performance differences of 3.97% and maximum differences reaching 86.24%. This was followed by feature selection and hyperparameter tuning which, although exhibiting average differences below 1%, were capable of producing performance inflation of up to 42% and 22.13%, respectively, when incorrectly implemented. Leakage introduced through normalization and value imputation led to smaller average changes; however, in some cases, the performance overestimation caused by these steps exceeded 10%, indicating that their risks should not be overlooked. As expected, the worst-case condition, where leakage was simultaneously introduced through all steps, produced the highest levels of performance inflation.

Another important finding is that data leakage can alter experimental conclusions regarding the top-performing algorithm compared to leakage-free pipelines. Overall, 12.86% of experiments would lead to different model-selection decisions when using leakage-prone pipelines. In the worst-case scenario, this proportion increases to 40.48% of datasets, meaning that nearly half of them would identify unrealistic top-performing algorithms that are unlikely to generalize to real-world data. These findings highlight how strongly different pipeline steps shape the training data: tasks that restructure or generate

samples tend to produce larger distortions in performance estimates, as well as those that directly interfere in the configuration of model parameters (*e.g.,* hyperparameter tuning).

When comparing algorithms, the differences in susceptibility were relatively small. All classifiers experienced changes of similar magnitude, with a mean performance difference ranging from 0.73% to 1.26%, suggesting that algorithm choice alone does not substantially mitigate or amplify the effects of data leakage. While LR, SVC, and NB were found to be comparatively less sensitive to leakage in relation to the other algorithms, they are still affected, producing performance inflation that can surpass 50% in specific cases. The variation observed across datasets and pipeline steps was considerably larger than the variation observed across algorithms.

Concerning dataset characteristics, the analysis of instance hardness measures did not produce a consistent profile of datasets more susceptible to leakage. The same dataset could display minimal impact for one step and substantial impact for another. Some tendencies were observed, for instance, datasets with larger class imbalance are more susceptible to data leakage introduced with oversampling by SMOTE, as expected. Moreover, datasets with higher values of the hardness measures MV, DS, kDN, and N1 appeared more often among experiments with strongest effects perceived for data leakage However, these evidences were not strong enough to support reliable predictions. This reinforces the notion that leakage effects are context-dependent and are influenced by interactions between data, algorithms, and pipeline operations. In particular, we found that NN and DT are more susceptible to data leakage during hyperparameter tuning, while higher risks were found for kNN and data leakage during class-balancing.

Overall, the results of this study highlight not only the variability of data leakage effects but also their practical significance for real-world ML workflows. The quantitative evidences provided in this study show that even small methodological oversights can substantially distort model evaluation, alter model-selection decisions, and ultimately compromise the reliability of ML models. These findings emphasize that preventing data leakage by adopting careful methodological practices during model preparation is not merely a good practice, but a critical requirement for producing trustworthy and reproducible results. To assist practitioners, we provide a concise set of recommendations aimed at reducing the likelihood of introducing leakage in applied settings.

Despite the breadth of the experimental evaluation, this study has several limitations. First, the analysis focuses exclusively on leakage introduced during pre-training pipeline steps and does not consider other important sources of leakage, such as tempo-

ral leakage, label leakage, or nonindependence between training and test samples, which are issues often found in real-world systems. Second, all experiments were performed on classification tasks using structured data and traditional ML algorithms; therefore, the findings may not fully generalize to regression or for predictive models trained with other classes of algorithms, such as deep learning. Additionally, the study relies on publicly available tabular datasets, which, although diverse, may not capture the complexity, noise patterns, or data-generation processes present in more realistic industry or domain-specific datasets.

The study limitations highlight the need for future work exploring broader forms of data leakage, additional task types, and richer data sources to obtain a more complete understanding of the phenomenon. Additionally, exploring new meta-features or predictive indicators of dataset susceptibility could help develop more robust guidelines for avoiding leakage in diverse ML scenarios. While our results offer strong evidence of the risks posed by data leakage to performance estimation and model generalization, expanding this line of work is essential for building a more comprehensive and principled understanding of how leakage manifests across modern ML pipelines.

REFERENCES

ALPAYDIN, E. **Introduction to machine learning**. [S.l.]: MIT press, 2020.

APICELLA, A.; ISGRÒ, F.; PREVETE, R. Don't push the button! exploring data leakage risks in machine learning and transfer learning. **Artificial Intelligence Review**, Springer, v. 58, n. 11, p. 339, 2025.

BECKER, A. E.; RECAMONDE-MENDOZA, M. Mind the gap: Investigating the impact of data leakage on machine learning predictive models. In: SPRINGER. **Brazilian Conference on Intelligent Systems**. [S.l.], 2025. p. 166–180.

BERNETT, J. et al. Guiding questions to avoid data leakage in biological machine learning applications. **Nature Methods**, Nature Publishing Group US New York, v. 21, n. 8, p. 1444–1453, 2024.

BERRAR, D. Bayes' theorem and naive bayes classifier. **Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics**, Elsevier Science Publisher Amsterdam, The Netherlands, v. 403, p. 412, 2018.

BOUKE, M. A.; ABDULLAH, A. An empirical study of pattern leakage impact during data preprocessing on machine learning-based intrusion detection models reliability. **Expert Systems with Applications**, v. 230, p. 120715, 2023. ISSN 0957-4174. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0957417423012174>.

BROOKSHIRE, G. et al. Data leakage in deep learning studies of translational eeg. **Frontiers in Neuroscience**, Frontiers Media SA, v. 18, p. 1373515, 2024.

BURZYKOWSKI, T. et al. Validation of machine learning algorithms. **American journal of orthodontics and dentofacial orthopedics**, Elsevier, v. 164, n. 2, p. 295–297, 2023.

BUSSOLA, N. et al. Ai slipping on tiles: Data leakage in digital pathology. In: SPRINGER. **Pattern Recognition. ICPR International Workshops and Challenges: Virtual Event, January 10–15, 2021, Proceedings, Part I**. [S.l.], 2021. p. 167–182.

CAWLEY, G. C.; TALBOT, N. L. C. On over-fitting in model selection and subsequent selection bias in performance evaluation. **J. Mach. Learn. Res.**, v. 11, p. 2079–2107, 2010. Available from Internet: <https://dl.acm.org/doi/10.5555/1756006.1859921>.

CHAWLA, N. V. et al. Smote: synthetic minority over-sampling technique. **Journal of Artificial Intelligence Research**, v. 16, p. 321–357, 2002.

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. In: **Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. [S.l.: s.n.], 2016. p. 785–794.

CUNNINGHAM, P.; CORD, M.; DELANY, S. J. Supervised learning. In: CORD, M.; CUNNINGHAM, P. (Ed.). **Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 21–49. ISBN 978-3-540-75171-7. Available from Internet: <https://doi.org/10.1007/978-3-540-75171-7_2>.

CUTLER, A.; CUTLER, D. R.; STEVENS, J. R. Random forests. In: ZHANG, C.; MA, Y. (Ed.). **Ensemble Machine Learning: Methods and Applications**. Boston, MA: Springer US, 2012. p. 157–175. ISBN 978-1-4419-9326-7. Available from Internet: <https://doi.org/10.1007/978-1-4419-9326-7_5>.

DIXON, M. F.; HALPERIN, I.; BILOKON, P. **Machine learning in finance**. [S.l.]: Springer, 2020.

DOMINGOS, P. A few useful things to know about machine learning. **Communications of the ACM**, ACM New York, NY, USA, v. 55, n. 10, p. 78–87, 2012.

DROBNJAKOVIĆ, F.; SUBOTIĆ, P.; URBAN, C. An abstract interpretation-based data leakage static analysis. In: SPRINGER. **International Symposium on Theoretical Aspects of Software Engineering**. [S.l.], 2024. p. 109–126.

FARROW, E.; MOORE, J.; GAsEVIć, D. Analysing discussion forum data: A replication study avoiding data contamination. In: **Proceedings of the 9th International Conference on Learning Analytics & Knowledge**. New York, NY, USA: Association for Computing Machinery, 2019. (LAK19), p. 170–179. ISBN 9781450362566. Available from Internet: <https://doi.org/10.1145/3303772.3303779>.

GARCÍA, S.; LUENGO, J.; HERRERA, F. **Data preprocessing in data mining**. [S.l.]: Springer, 2015.

GREENE, D.; CUNNINGHAM, P.; MAYER, R. Unsupervised learning and clustering. In: CORD, M.; CUNNINGHAM, P. (Ed.). **Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 51–90. ISBN 978-3-540-75171-7. Available from Internet: <https://doi.org/10.1007/978-3-540-75171-7_3>.

GUPTA, M.; CHRISTOPHER, J.; RAMISETTY, K. Mitigation strategies for data leakage in machine learning system design. In: **2025 IEEE Region 10 Symposium (TENSYMP)**. [S.l.: s.n.], 2025. p. 1–8.

JURAFSKY, D. et al. **Speech and Language Processing**. Pearson Education, 2014. ISBN 9780133252934. Available from Internet: <https://books.google.com.br/books?id=Cq2gBwAAQBAJ>.

KANG, S. k-nearest neighbor learning with graph neural networks. **Mathematics**, v. 9, n. 8, 2021. ISSN 2227-7390. Available from Internet: <https://www.mdpi.com/2227-7390/9/8/830>.

KAPOOR, S.; NARAYANAN, A. Leakage and the reproducibility crisis in machine-learning-based science. **Patterns**, Elsevier, v. 4, n. 9, 2023.

KAUFMAN, S. et al. Leakage in data mining: Formulation, detection, and avoidance. **ACM Transactions on Knowledge Discovery from Data (TKDD)**, ACM New York, NY, USA, v. 6, n. 4, p. 1–21, 2012.

KRAMER, O. K-nearest neighbors. In: **Dimensionality Reduction with Unsupervised Nearest Neighbors**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 13–23. ISBN 978-3-642-38652-7. Available from Internet: <https://doi.org/10.1007/978-3-642-38652-7_2>.

KUNCHEVA, L. I.; RODRíGUEZ, J. J. On feature selection protocols for very low-sample-size data. **Pattern Recognition**, v. 81, p. 660–673, 2018. ISSN 0031-3203. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S003132031830102X>.

LEMAîTRE, G.; NOGUEIRA, F.; ARIDAS, C. K. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. **Journal of Machine Learning Research**, v. 18, n. 17, p. 1–5, 2017. Available from Internet: <http://jmlr.org/papers/v18/16-365.html>.

LUAN, H.; TSAI, C.-C. A review of using machine learning approaches for precision education. **Educational Technology & Society**, JSTOR, v. 24, n. 1, p. 250–266, 2021.

MAYER, I. et al. R-miss-tastic: a unified platform for missing values methods and workflows. **arXiv preprint arXiv:1908.04822**, 2019.

MITCHELL, T. **Machine Learning**. [S.l.]: McGraw-Hill Education, 1997.

NAQA, I. E.; MURPHY, M. J. What is machine learning? In: NAQA, I. E.; LI, R.; MURPHY, M. J. (Ed.). **Machine Learning in Radiation Oncology: Theory and Applications**. Cham: Springer International Publishing, 2015. p. 3–11. ISBN 978-3-319-18305-3. Available from Internet: <https://doi.org/10.1007/978-3-319-18305-3_1>.

NOBLE, W. S. What is a support vector machine? **Nature biotechnology**, Nature Publishing Group UK London, v. 24, n. 12, p. 1565–1567, 2006.

PAIVA, P. Y. A. et al. Relating instance hardness to classification performance in a dataset: a visual approach. **Machine Learning**, Springer, v. 111, n. 8, p. 3085–3123, 2022.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

RASCHKA, S. Model evaluation, model selection, and algorithm selection in machine learning. **CoRR**, abs/1811.12808, 2018. Available from Internet: <http://arxiv.org/abs/1811.12808>.

RINGNÉR, M. What is principal component analysis? **Nature Biotechnology**, Nature Publishing Group US New York, v. 26, n. 3, p. 303–304, 2008.

ROKACH, L.; MAIMON, O. Decision trees. In: MAIMON, O.; ROKACH, L. (Ed.). **Data Mining and Knowledge Discovery Handbook**. Boston, MA: Springer US, 2005. p. 165–192. ISBN 978-0-387-25465-4. Available from Internet: <https://doi.org/10.1007/0-387-25465-X_9>.

ROMANO, J. D. et al. Pmlb v1. 0: an open-source dataset collection for benchmarking machine learning methods. **Bioinformatics**, Oxford University Press, v. 38, n. 3, p. 878–880, 2022.

ROSENBLATT, M. et al. Data leakage inflates prediction performance in connectome-based machine learning models. **Nature Communications**, Nature Publishing Group UK London, v. 15, n. 1, p. 1829, 2024.

SAMALA, R. K. et al. Hazards of data leakage in machine learning: a study on classification of breast cancer using deep neural networks. In: SPIE. **Medical Imaging 2020: Computer-Aided Diagnosis**. [S.l.], 2020. v. 11314, p. 279–284.

SASSE, L. et al. Overview of leakage scenarios in supervised machine learning. **Journal of Big Data**, Springer, v. 12, n. 1, p. 135, 2025.

SHA, L. et al. Leveraging class balancing techniques to alleviate algorithmic bias for predictive tasks in education. **IEEE Transactions on Learning Technologies**, IEEE, v. 15, n. 4, p. 481–492, 2022.

SHIM, M.; LEE, S.-H.; HWANG, H.-J. Inflated prediction accuracy of neuropsychiatric biomarkers caused by data leakage in feature selection. **Scientific Reports**, Nature Publishing Group UK London, v. 11, n. 1, p. 7980, 2021.

SUTHAHARAN, S.; SUTHAHARAN, S. Support vector machine. **Machine learning models and algorithms for big data classification: thinking with examples for effective learning**, Springer, p. 207–235, 2016.

TAN, M. S. et al. A review on omics-based biomarkers discovery for alzheimer's disease from the bioinformatics perspectives: statistical approach vs machine learning approach. **Computers in Biology and Medicine**, Elsevier, v. 139, p. 104947, 2021.

ULLAH, Z. et al. Applications of artificial intelligence and machine learning in smart cities. **Computer Communications**, Elsevier, v. 154, p. 313–323, 2020.

VABALAS, A. et al. Machine learning algorithm validation with a limited sample size. **PloS one**, Public Library of Science San Francisco, CA USA, v. 14, n. 11, p. e0224365, 2019.

WANG, S.-C. Artificial neural network. In: **Interdisciplinary Computing in Java Programming**. [S.l.]: Springer, 2003. p. 81–100.

WEN, F. et al. Mitigating data leakage and class imbalance in explainable ai for stroke prediction. **Research Square preprint**, 2025.

WEN, J. et al. Convolutional neural networks for classification of alzheimer's disease: Overview and reproducible evaluation. **Medical Image Analysis**, Elsevier, v. 63, p. 101694, 2020.

YAGIS, E. et al. Effect of data leakage in brain mri classification using 2d convolutional neural networks. **Scientific reports**, Nature Publishing Group UK London, v. 11, n. 1, p. 22544, 2021.

YANG, C. et al. Data leakage in notebooks: Static detection and better processes. In: **Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering**. [S.l.: s.n.], 2022. p. 1–12.

YOZGATLIGIL, C. et al. Comparison of missing value imputation methods in time series: the case of turkish meteorological data. **Theoretical and Applied Climatology**, Springer, v. 112, p. 143–167, 2013.

APPENDIX A — RESUMO EXPANDIDO

Data Leakage no contexto de aprendizado de máquina é o nome dado a contaminação inadvertida dos dados de treinamento de um modelo com informações obre os dados de teste ou validação. Consequentemente, os modelos treinados possuem estimativas de performance não reproduzíveis no mundo real. Em um caso mais simples, essa contaminação pode acontecer simplesmente por não se evitar sobreposição entre os conjuntos de treino e teste. Entretanto, também é possível se inserir *data leakage* através erros metodológicos, como a utilização incorreta dos dados durante etapas de preparação do modelo: Seleção de *Feature*, Normalização, Imputação de Valores, Otimização de Hiperparametros e Balanceamento de Classes.

Alguns trabalhos recentes buscam entender esse problema, estudar formas de evitá-lo, e avaliar os seus efeitos em aplicações específicas. Porém, há uma falta de estudos que investiguem sistematicamente a variação dos efeitos de *data leakage* quando inseridos em diferentes etapas de preparação do modelo, em diferentes dataset e em diferentes algoritmos. Este trabalho busca preencher esse espaço, investigando os efeitos do *data leakage* quando inserido em cada uma das etapas mencionadas. Também é avaliado uma situação de "pior caso", onde o vazamento de dados é inserido em todas as etapas ao mesmo tempo.

Com esse objetivo, realizamos testes sistemáticos com treinamento de modelos pareados com e sem inserção de *data leakage*. Para inserir variabilidade na análise foram selecionados 42 datasets de classificação, utilizando uma seleção baseada em rodadas de *Principal Component Analysis* onde os maiores outliers eram selecionados. Foram extraídas medidas de complexidade dos dados, com objetivo de verificar se essas características dos datasets estão relacionadas ao efeito da contaminação por *data leakage*. Além disso, foram utilizados 8 diferentes algoritmos de classificação: K-nearest Neighbors, Naive Bayes, Suport Vector Classifiers, Arvores de Decisão, Florestas Aleatórias, Regressão Logistica, XGBoost e Redes Neurais.

A inserção do vazamento de dados foi feita de acordo com a tarefa em que foi inserido. Para tarefas em que não há o treinamento de múltiplos modelos, como Seleção de Features, Normalização, Imputação de Valores e Balanceamento de classes (feito através do algoritmo SMOTE) foi seguido o seguinte processo: O modelo com vazamento de dados teve a tarefa executada em todos os dados, antes da separação das partições de validação cruzada, enquanto o modelo sem leakage implementa a tarefa dentro da vali-

dação cruzada, utilizando as partições de treino para fazer o fit da tarefa e aplicando-a nas partições de treino e teste. Para o caso da otimização de hiperparametros a inserção de contaminação foi feita ao utilizar apenas um nível de validação cruzada para selecionar os parâmetros e estimar a performance. Para evitar o *data leakage*, foi utilizada validação cruzada aninhada, com a validação cruzada interna sendo utilizada para seleção de hiperparametros e a externa para estimativa de performance.

Tendo os resultados dos testes pareados, a análise dos resultados foi feita através de 3 abordagens diferentes: Análise estatística das distribuições das estimativas de performance, análise visual da distribuição das diferenças pareadas e clusterização baseada nas diferenças pareadas das estimativas de performance. Nossos resultados mostraram tendências interessantes que destacam a importância de se evitar a inserção de *data leakage*.

Considerando todos os casos testados, a média de aumento do F1-Score ($\overline{\Delta F1}$) foi de 0,99%. Foi percebido um aumento de estimativa superior a 1% em 13,48% dos testes pareados executados. O gráfico da distribuição das diferenças também mostra um número relevante de superestimações, incluindo diferenças superiores a 80%. Utilizando esses dados para realizar a clusterização, obtivemos 4 clusteres divididos pelo efeito observado nas estimativas de performance: 1 - Sem influência, 2- Baixa Influência, 3- Influência Moderata, 4 - Alta influência.

Ao olharmos para os resultados por datasets, fica claro que os efeitos da contaminação variam consideravelmente de acordo com o os dados utilizados para treinar o modelo. Das 42 base de dados utilizadas, 12 mostraram $\overline{\Delta F1}$ acima de 1%, 23 apresentaram $\overline{\Delta F1}$ maior que 0% e 3 apresentaram $\overline{\Delta F1}$ abaixo de 0%. Além disso 4 datasets tiveram $p > 0,05$, indicando não haver significância na diferença entre as distribuições das estimativas de performance. O dataset com maior aumento médio apresentou $\overline{\Delta F1}$ de 14.33%. Utilizando a clusterização, avaliamos as distribuições das medidas de complexidade dos datasets. Apesar de haver diferenças diferenças nas distribuições, não observamos nenhuma tendência clara de alguma métrica estar diretamente relacionada ao efeito observado.

Em relação aos efeitos de acordo com os diferentes algoritmos utilizados, não foram observadas diferenças relevantes entre o $\overline{\Delta F1}$ observado por cada um dos classificadores. Todos tiveram efeitos superiores a 0% e próximos de 1%. Na clusterização foi possível perceber que todos os algoritmos estavam presentes nos clusters de influência baixa e moderada, enquanto LR, SVC e NB não estavam presentes no cluster de maior

influência.

A tarefa de pré-processamento usada para inserir leakage mostrou ter grande impacto nos efeitos observados. O balanceamento de classe através de instâncias sintéticas se mostrou a tarefa com maior aumento da estimativa de performance, tendo um $\overline{\Delta F1}$ de 3.97%. Essa tarefa também causou uma troca no algoritmo melhor avaliado para tarefa em 23,81% dos experimentos. Em seguida, as tarefas de Seleção de Features e de Otimização de Hiperparametros aparacem om média de aumento de 0.54% e 0.43% respectivamente. A inserção de *data leakage* tarefa de Normalização teve uma média de aumento baixa de 0.02%, enquanto a inserção pela tarefa de Imputação de valores não mostrou significância na diferença entre as distribuições das estimativas com e sem vazamento de dados.

O teste de "pior caso", como esperado, mostrou aumento das estimativas de performance superior a todas as tarefas, com um $\overline{\Delta F1}$ de 4.08%. Notamos datasets com aumentos médios de 10%, 16% e surpreendentes 65% na estimativa média para essa análise. A alteração do algoritmo dentro desse subconjunto de testes também não mostrou grandes alterações aos efeitos nas estimativas de performance. Adcionamelmente, a inserção de leakage simultaneamente através de todas as tarefas causou uma troca no algoritmo que melhor performou (algoritmo vencedor) em 40.48% dos experimentos.

Analisamos também se combinações específicas entre tarefas, algoritmos e datasets apresentavam diferentes efeitos. Nossa análise deixou claro que os efeitos variam de acordo com essas combinações, com datasets apresentando aumentos médios acima de 5% para um algoritmo e abaixo de 1% para outros. Uma combinação interessante entre algoritmo e tarefa de pré-processamento foi observada para a Otimização de Hiper Parâmetro que teve o maior aumento médio quando pareado com o algoritmo de Redes Neurais. Isso pode ser explicado pela importâcia dos parâmetros em uma rede neural, uma vez que a alteração do tamanho ou número de camadas muda completamente a fronteira de decisão possível, por exemplo.

Esses resultados, apesar de representarem os datasets, algoritmos e tarefas selecionadas, ressaltam a importância de se evitar a introdução de contaminação no pipeline de treinamento de modelos de classificação. A alta variabilidade e imprevisibilidade dos efeitos atestam que os cuidados devem ser tomados em todos os casos. Com esse objetivo, esse trabalho apresenta uma lista sucinta de melhores práticas, que ajudam a evitar a introdução de *data leakage*.

APPENDIX B — FULL LIST OF HYPERPARAMETERS EVALUATED

B.1 Logistic Regression

- Solver: ['saga', 'lbfgs']
- C: [0.01, 0.1, 1, 10, 100]

B.2 Support Vector Classifier

- C: [0.01, 0.1, 1, 10, 100]
- Solver: ['linear', 'rbf', 'poly']/

B.3 K-Nearest Neighbors

- N Neighbors: [3, 5, 7, 9, 11]
- Metric: ['euclidean', 'manhattan']

B.4 Random Forest

- N Estimators: [100, 200, 500]
- Max Depth: [3, 5, 8, 10, None]

B.5 Decision Tree

- Max Depth: [3, 5, 8, 10, None]

B.6 Naive Bayes

- Var Smoothing: [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]

B.7 Neural Networks

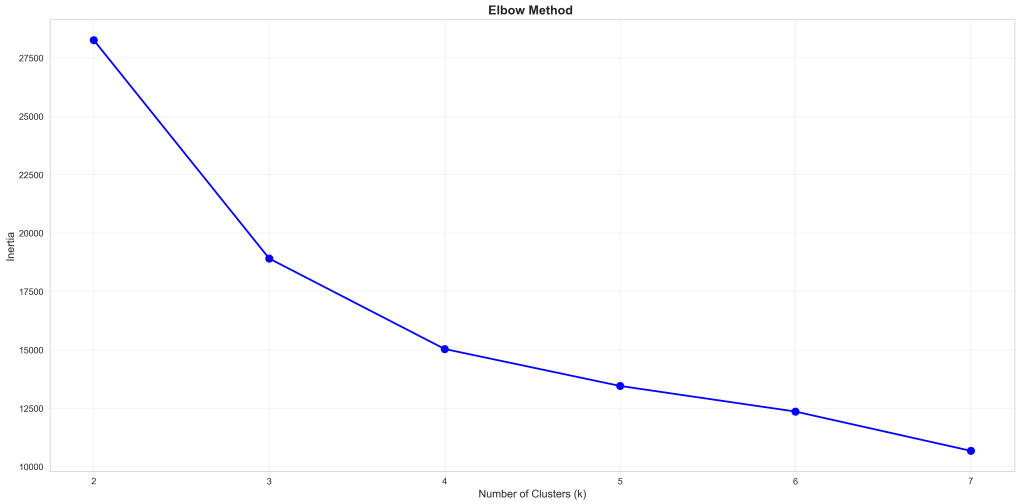
- hidden layer sizes: [(50,), (100,), (100, 50)],
- activation : [’relu’, ’tanh’],
- alpha: [0.0001, 0.001, 0.01],
- max iterations: [500, 1000]

B.8 XGBoost

- number of estimators: [100, 200, 300],
- max depth: [3, 5, 7],
- learning rate: [0.01, 0.1, 0.2],
- subsample: [0.8, 1.0]

APPENDIX C — METRICS USED TO SELECT NUMBER OF CLUSTERS K-MEANS

Figure C.1: Plots of the values used for selecting the best value of K

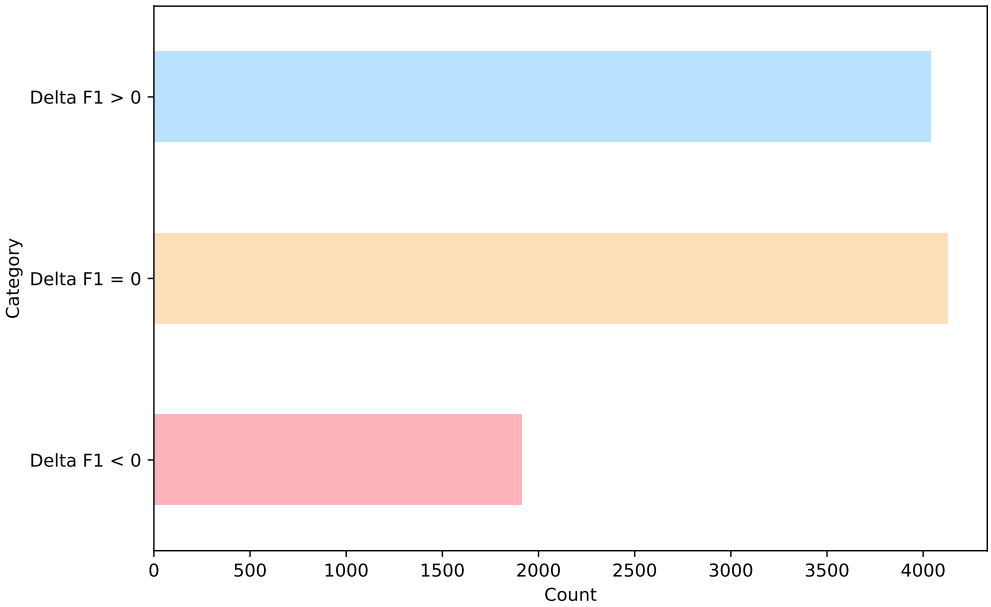


Source: The Author

APPENDIX D — COUNTS OF PAIRED EXPERIMENTS FOR $\Delta F1$ VALUES

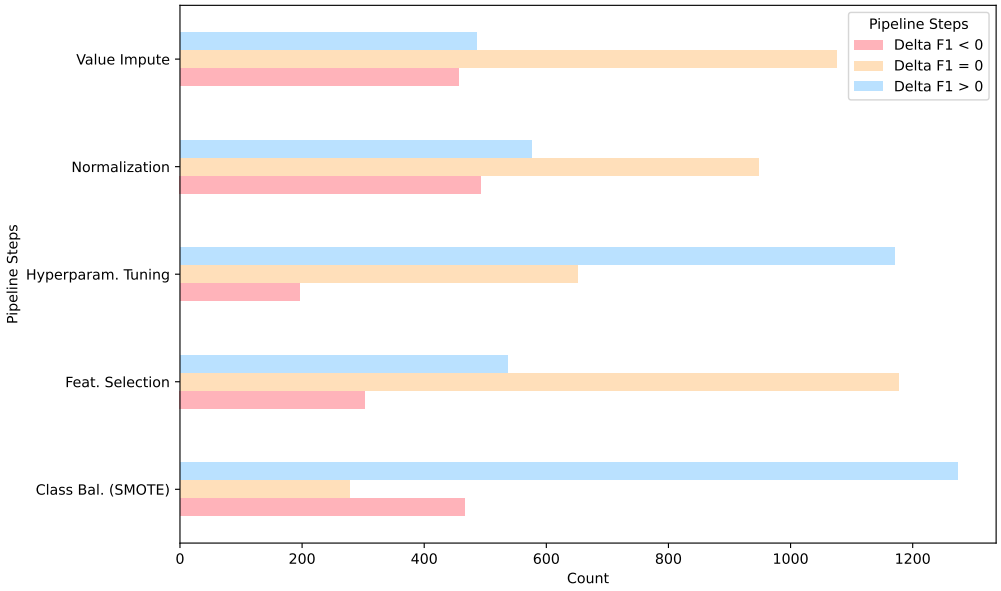
This appendix shows the plot of counts of experiment executions that had positive, negative, and neutral $\Delta F1$ values.

Figure D.1: Counts of experiment executions according to $\Delta F1$ values



Source: The Author

Figure D.2: Counts of experiment executions according to $\Delta F1$ values grouped by pipeline step used to insert data leakage



Source: The Author

APPENDIX E — ADDITIONAL ANALYSIS FOR MISSING-VALUE IMPUTATION AND FEATURE SELECTION STEPS

This appendix shows additional analysis for data leakage introduced in the missing-value imputation and feature selection steps, changing some of the hyperparameters involved in these tasks. Imputations with KNN was implemented with the `KNNImputer`, whereas imputations with the mean and median were applied with the the `SimpleImputer`. Feature selection was based on the `SelectPercentile` class using the `f_classif` scoring function.

Table E.1: Analysis of F1-scores for different percentages of selected features.

% of Features Selected	$\overline{\Delta F1}$	$\overline{F1leak}$	$\overline{F1correct}$	$Max\Delta F1$	$Min\Delta F1$	p-value
1	0.34%	49.23%	48.89%	51.05%	-15.28%	$p < 0.0001$
5	0.05%	59.39%	59.34%	27.45%	-22.25%	0.0532
10	0.54%	66.50%	65.96%	42.27%	-33.88%	$p < 0.0001$
20	0.30%	71.23%	70.93%	40.45%	-78.76%	$p < 0.0001$

Table E.2: Analysis of F1-scores for different percentages of missing values.

% Missing Values	$\overline{\Delta F1}$	$\overline{F1leak}$	$\overline{F1correct}$	$Max\Delta F1$	$Min\Delta F1$	p-value
5	0.00%	84.55%	84.55%	12.87%	-14.06%	0.3457
10	0.01%	84.41%	84.40%	12.08%	-18.03%	0.0036
20	0.01%	84.08%	84.07%	18.60%	-16.96%	0.0022
30	-0.00%	84.01%	84.02%	9.88%	-17.43%	0.0501

Table E.3: Analysis of F1-scores for different imputation strategies.

Imputer	$\overline{\Delta F1}$	$\overline{F1leak}$	$\overline{F1correct}$	$Max\Delta F1$	$Min\Delta F1$	p-value
`KNNImputer`	0.03%	84.41%	84.38%	18.60%	-18.03%	$p < 0.0001$
mean	-0.01%	84.24%	84.25%	12.87%	-11.14%	0.0363
median	-0.01%	84.14%	84.15%	10.83%	-17.43%	0.8476

Figure E.1: Distribution F1-Score differences for different percentages of missing values on the subset of 30 datasets

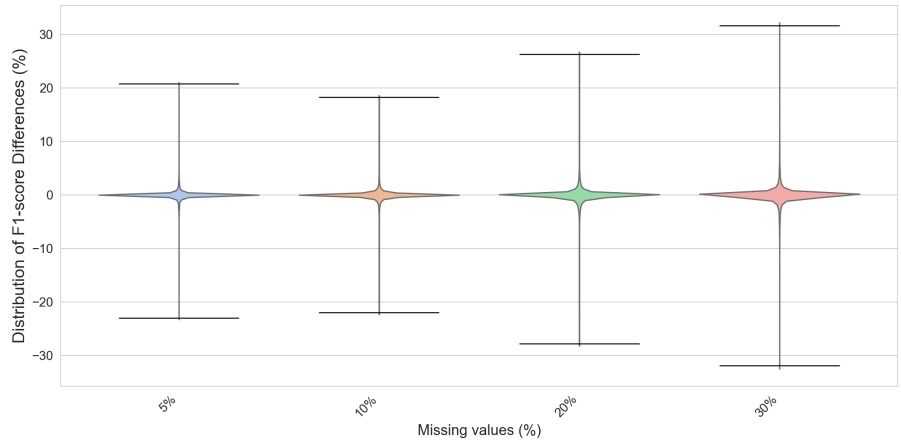


Figure E.2: Distribution F1-Score differences for different imputation strategies on the subset of 30 datasets

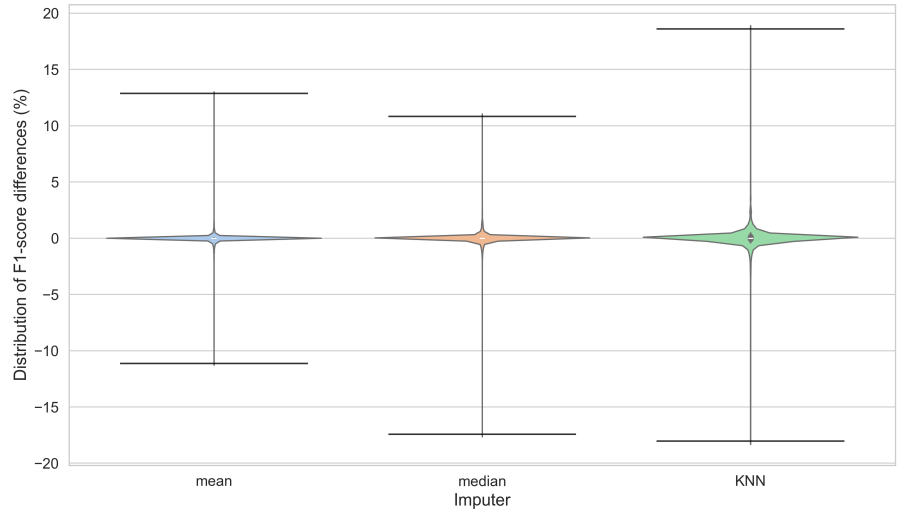
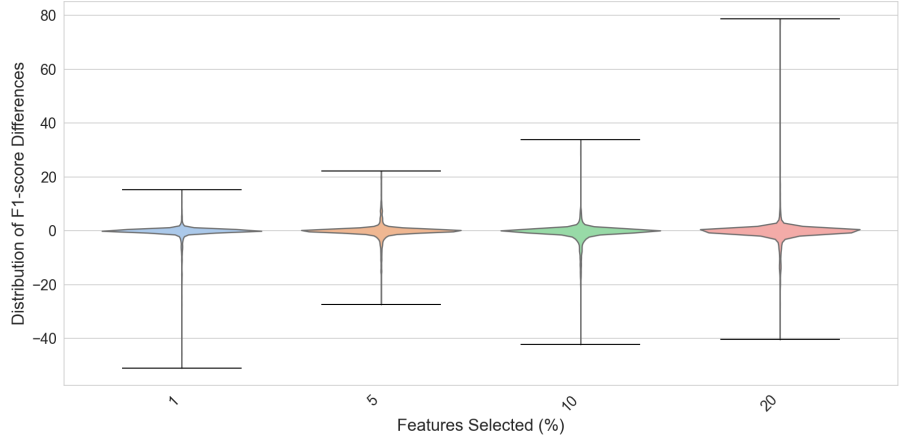


Figure E.3: Distribution F1-Score differences for different percentages of selected features



APPENDIX F — DISTRIBUTION OF PERFORMANCE DIFFERENCES ACROSS ALL DATASETS

Figure F.1: Performance difference distribution for datasets (Part 1)

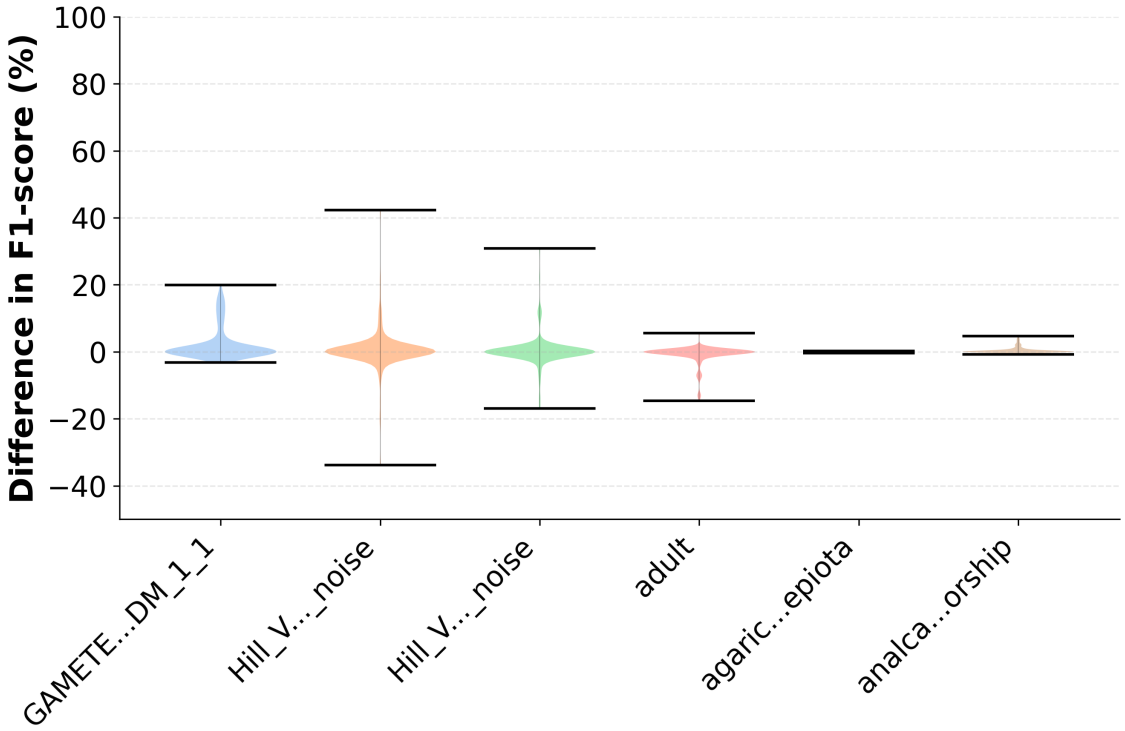


Figure F.2: Performance difference distribution for datasets (Part 2)

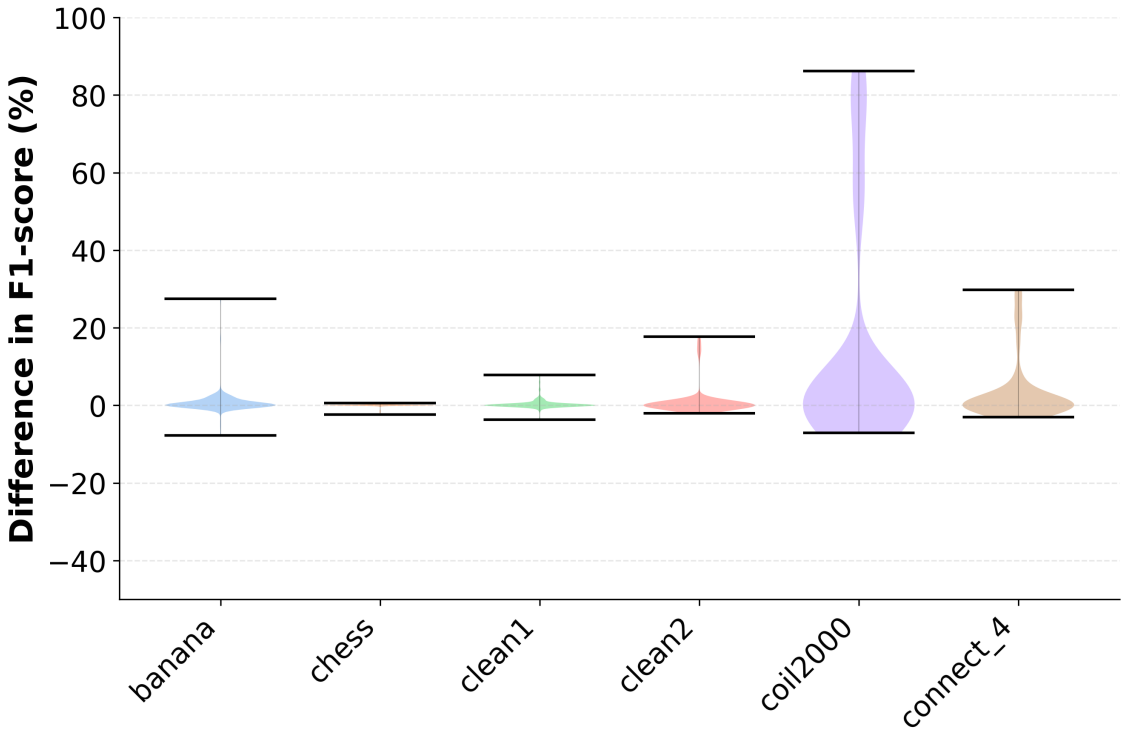


Figure F.3: Performance difference distribution for datasets (Part 3)

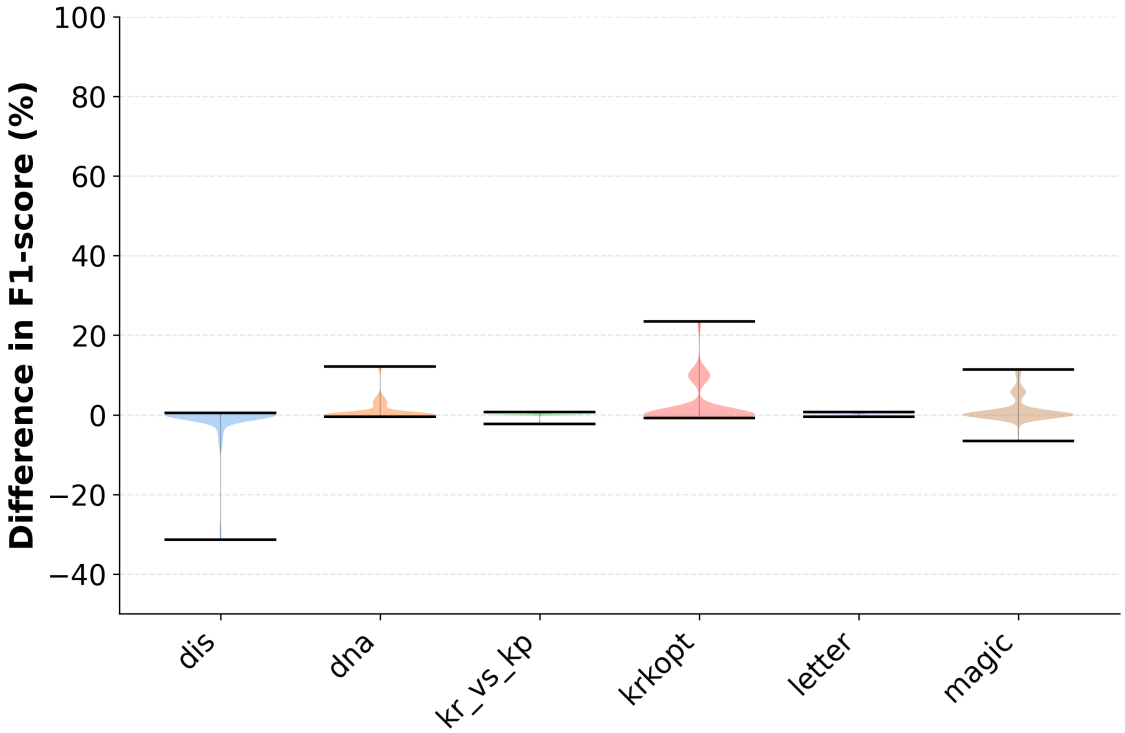


Figure F.4: Performance difference distribution for datasets (Part 4)

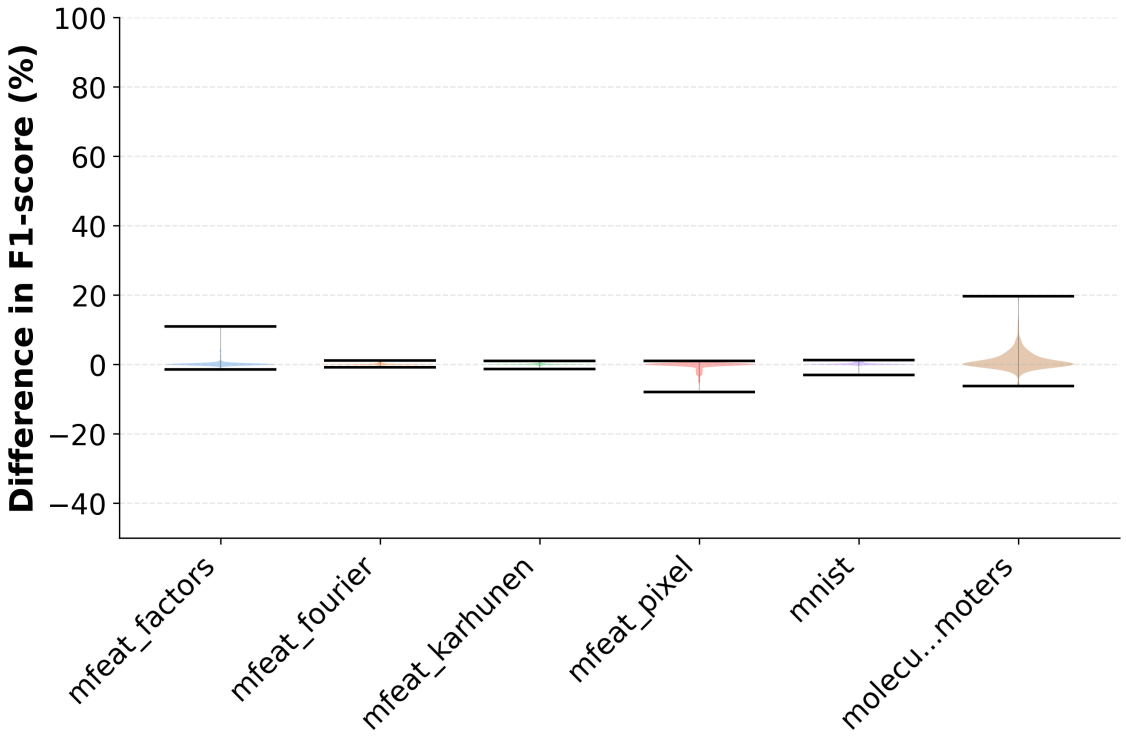


Figure F.5: Performance difference distribution for datasets (Part 5)

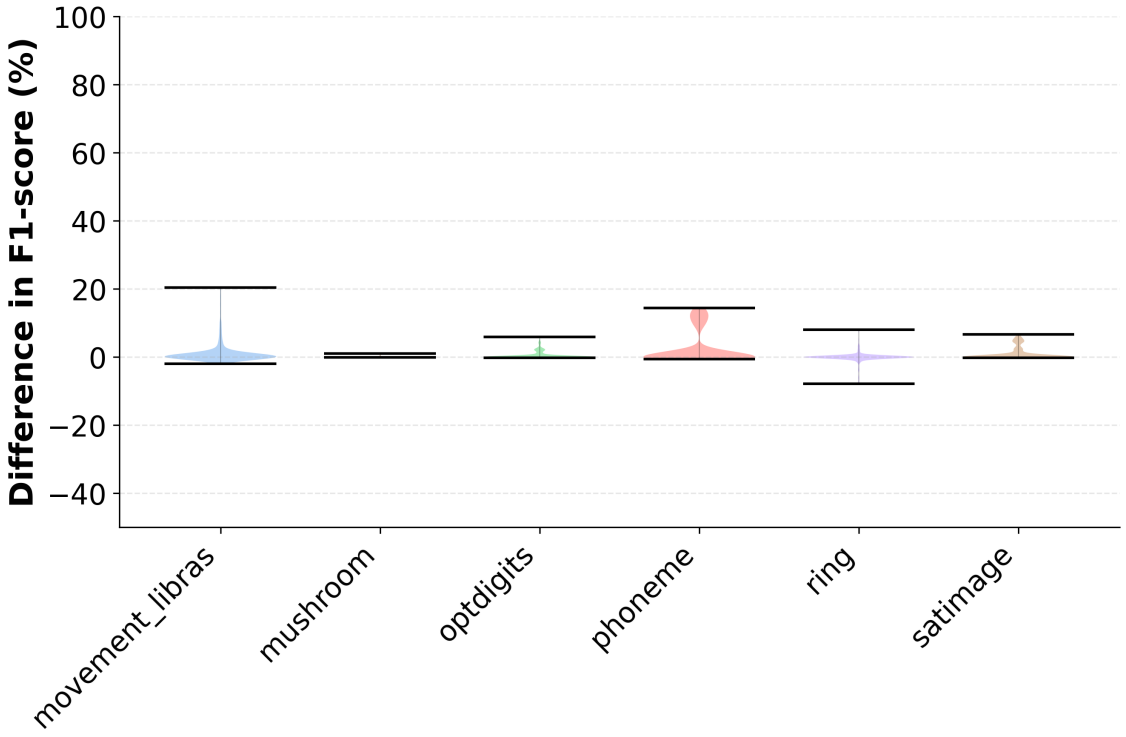


Figure F.6: Performance difference distribution for datasets (Part 6)

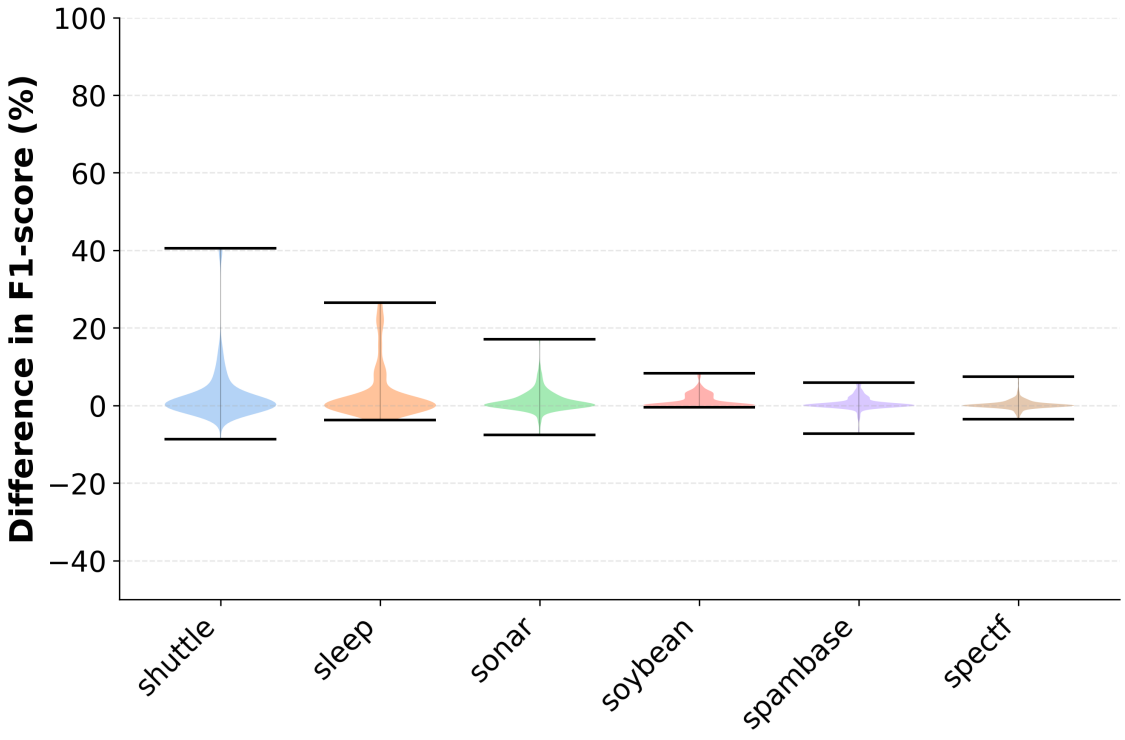
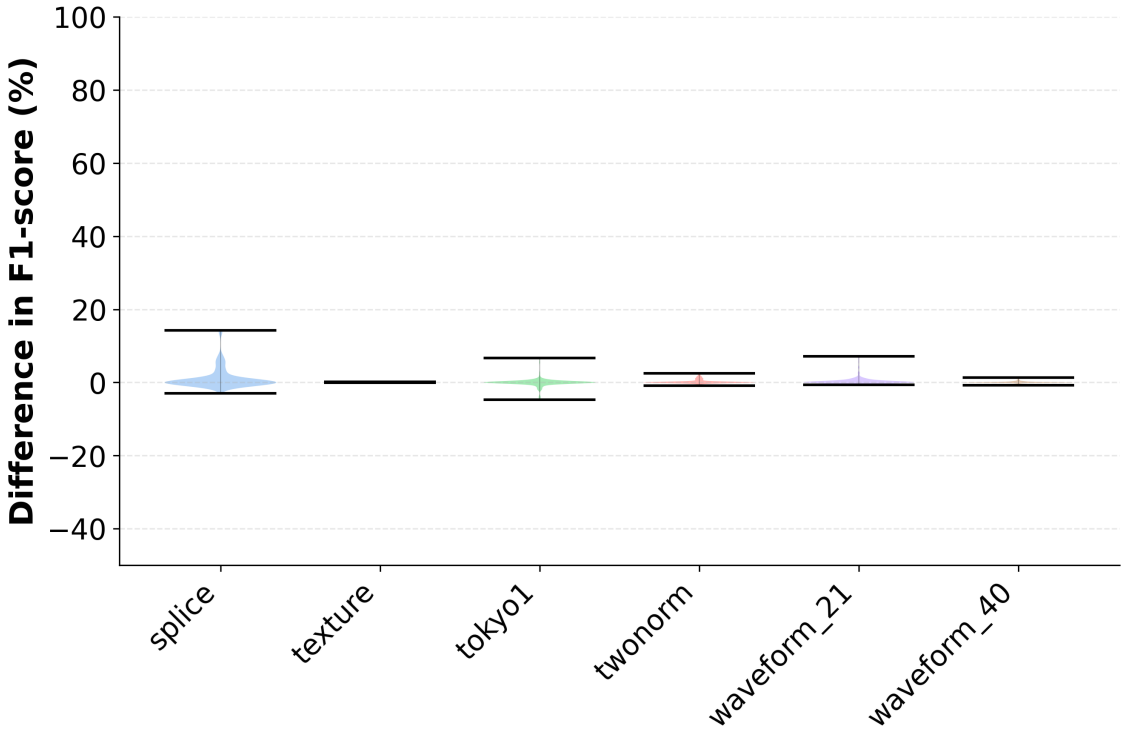


Figure F.7: Performance difference distribution for datasets (Part 7)



APPENDIX G — DETAILED STATISTICS RESULTS

Table G.1: Wilcoxon signed-rank analysis analysis of F1-scores in models with and without leakage across all executions.

Category	Statistic	p-value
Overall	4739657.0000	$p < 0.0001$

Table G.2: Wilcoxon signed-rank analysis of F1-scores for each dataset.

Category	**Statistic**	**p-value**
GAMETES_Epistasis_2_Way_1000atts_0.4H_EDM_1_EDM_1_1	5417.0000	$p < 0.0001$
Hill_Valley_with_noise	8026.0000	0.0135
Hill_Valley_without_noise	7960.5000	0.8917
adult	1720.0000	$p < 0.0001$
agaricus_lepiota	143.0000	0.0396
analcatdata_authorship	1179.0000	$p < 0.0001$
banana	2126.0000	$p < 0.0001$
chess	1289.0000	$p < 0.0001$
clean1	802.5000	$p < 0.0001$
clean2	743.0000	$p < 0.0001$
coil2000	1858.0000	$p < 0.0001$
connect_4	980.0000	$p < 0.0001$
dis	3142.0000	0.1194
dna	606.0000	$p < 0.0001$
kr_vs_kp	1858.5000	0.0006
krkopt	714.0000	$p < 0.0001$
letter	2081.0000	$p < 0.0001$
magic	2029.0000	$p < 0.0001$
mfeat_factors	3675.0000	$p < 0.0001$
mfeat_fourier	2631.0000	0.0042
mfeat_karhunen	3293.0000	0.0180
mfeat_pixel	7295.5000	0.1830
mnist	2425.0000	$p < 0.0001$
molecular_biology_promoters	2378.5000	$p < 0.0001$
movement_libras	2621.0000	$p < 0.0001$
mushroom	63.0000	$p < 0.0001$
optdigits	1878.0000	$p < 0.0001$
phoneme	2029.0000	$p < 0.0001$
ring	5567.0000	0.0112
satimage	1484.0000	$p < 0.0001$
shuttle	565.0000	$p < 0.0001$
sleep	510.0000	$p < 0.0001$
sonar	3253.0000	$p < 0.0001$
soybean	450.0000	$p < 0.0001$
spambase	3540.0000	$p < 0.0001$
spectf	5308.0000	0.0227
splice	6420.0000	$p < 0.0001$
texture	1715.0000	0.0360
tokyo1	4621.5000	0.7115
twonorm	3501.0000	$p < 0.0001$
waveform_21	3300.0000	$p < 0.0001$
waveform_40	3575.0000	$p < 0.0001$

Table G.3: Wilcoxon signed-rank analysis of F1-scores for each algorithm.

Category	Statistic	p-value
DT	80066.0000	$p < 0.0001$
KNN	115618.0000	$p < 0.0001$
LR	60839.0000	$p < 0.0001$
NB	35260.5000	$p < 0.0001$
NN	103247.0000	$p < 0.0001$
RF	97951.5000	$p < 0.0001$
SVC	82081.0000	$p < 0.0001$
XGB	38922.0000	$p < 0.0001$

Table G.4: Wilcoxon signed-rank analysis of F1-scores for each pipeline step.

Category	Statistic	p-value
Feature Selection	108415.0000	$p < 0.0001$
Hyperparameter Tuning	95957.5000	$p < 0.0001$
Value Imputation	218495.0000	0.7092
Normalization	261567.0000	0.0157
Class Bal. (SMOTE)	280342.5000	$p < 0.0001$

Table G.5: Wilcoxon signed-rank analysis of F1-scores in models with and without leakage for worst-case scenario.

Category	Statistic	p-value
Overall	313494.0000	$p < 0.0001$

APPENDIX H — CORRELATIONS OF DATASETS METADATA AND PERCEIVED F1-SCORE DIFFERENCE

Table H.1: Spearman correlation between dataset metadata and mean F1-score difference comparing leakage-prone and leakage-free models.

Dataset characteristic	Spearman correlation	p-value
Class imbalance	0.332	0.0319
Instances per class	0.116	0.4647
Number of instances	0.105	0.5098
Number of classes	0.080	0.6154
Features per class	-0.075	0.6384
Number of features	-0.033	0.8361

Figure H.1: Correlation between class imbalance and mean F1-Score difference for each of the 42 datasets.

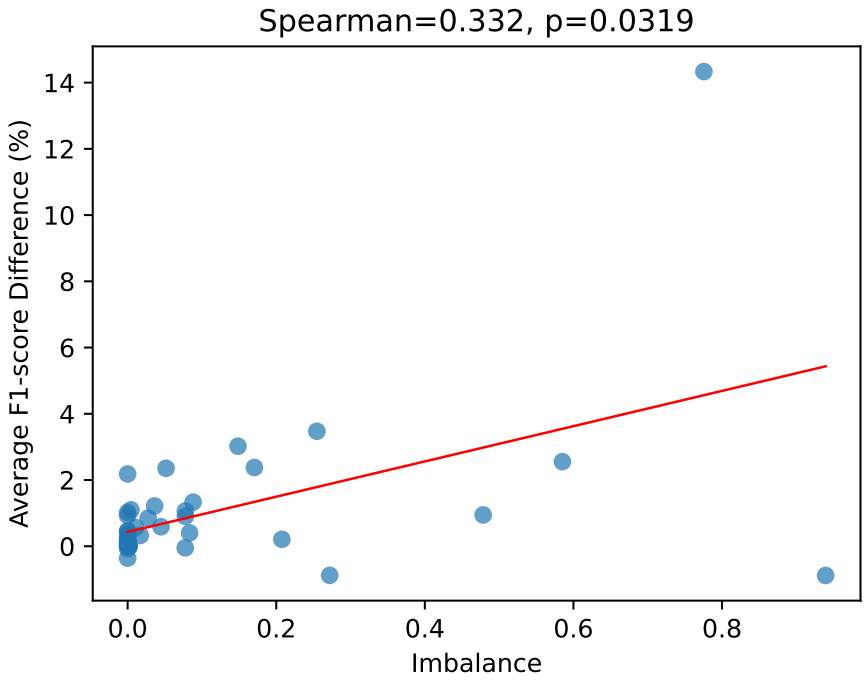


Figure H.2: Correlation between number of Classes and mean F1-score difference for each of the 42 datasets.

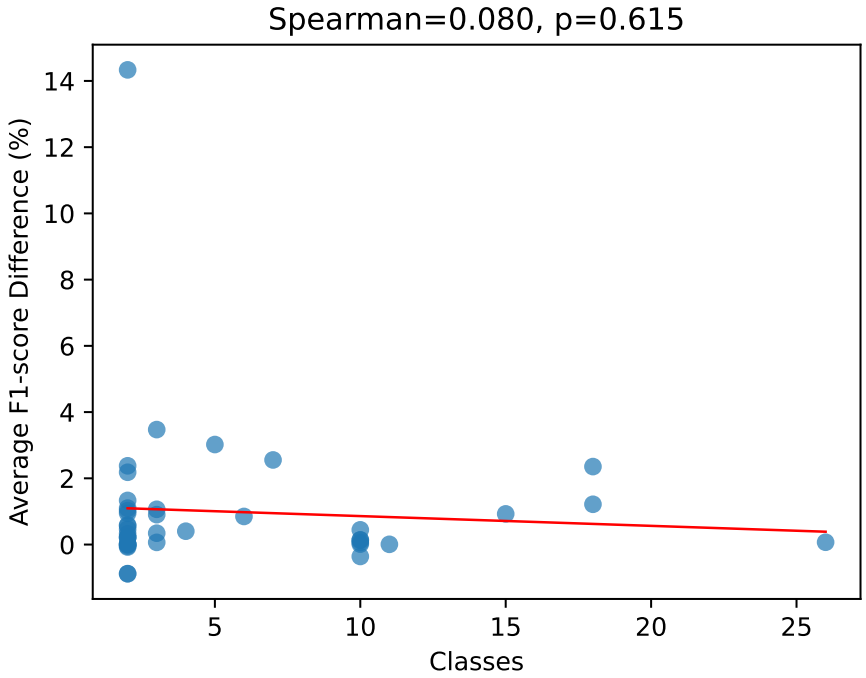


Figure H.3: Correlation between features per class and mean F1-score difference for each of the 42 datasets.

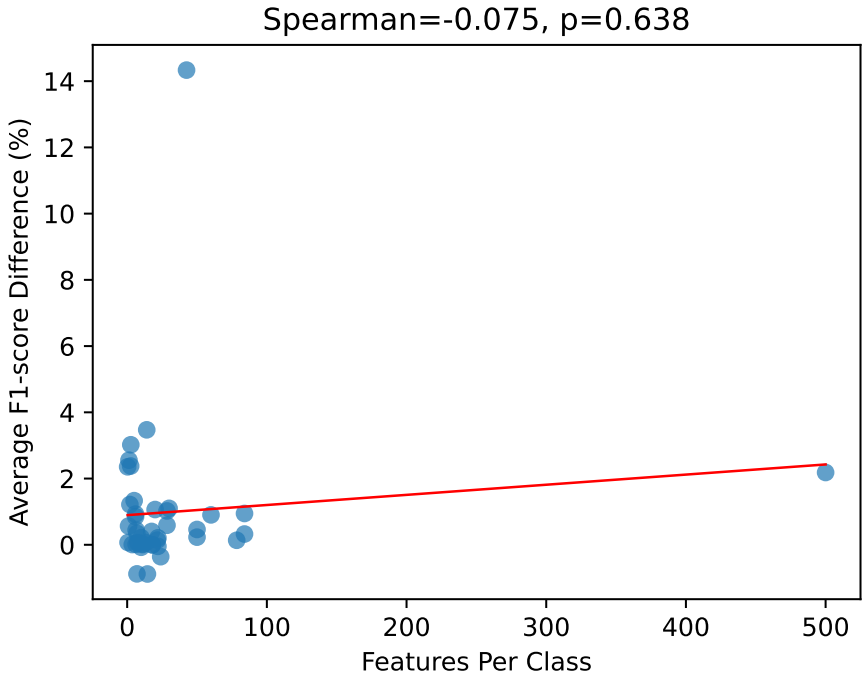


Figure H.4: Correlation between number of Features and mean F1-score difference for each of the 42 datasets.

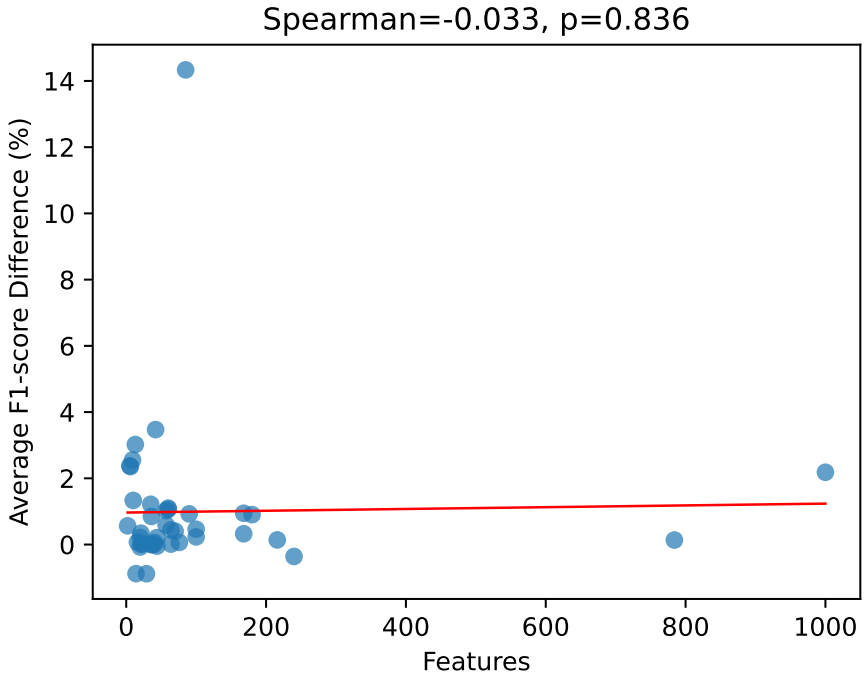


Figure H.5: Correlation between instances per class and mean F1-score difference for each of the 42 datasets.

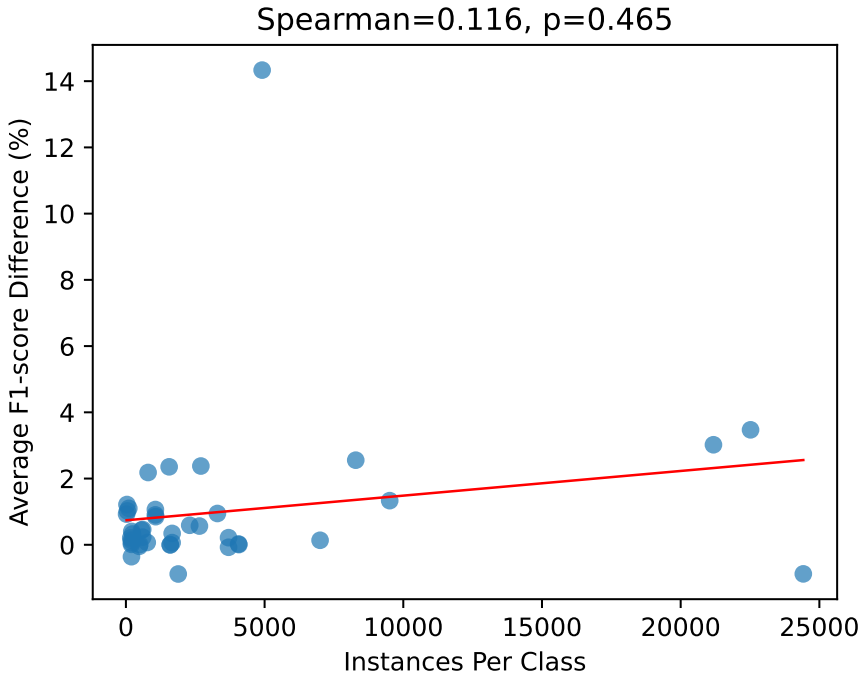
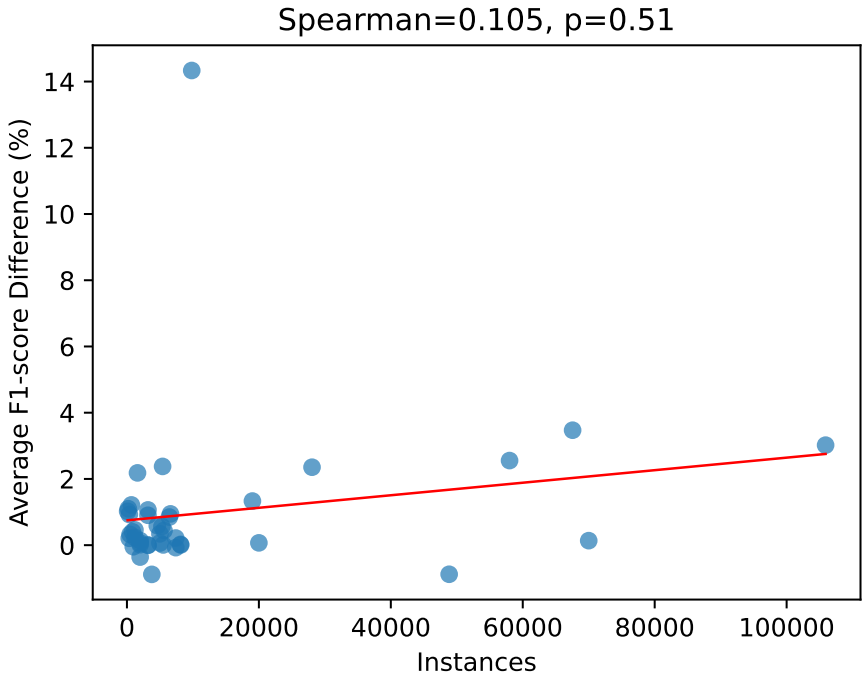
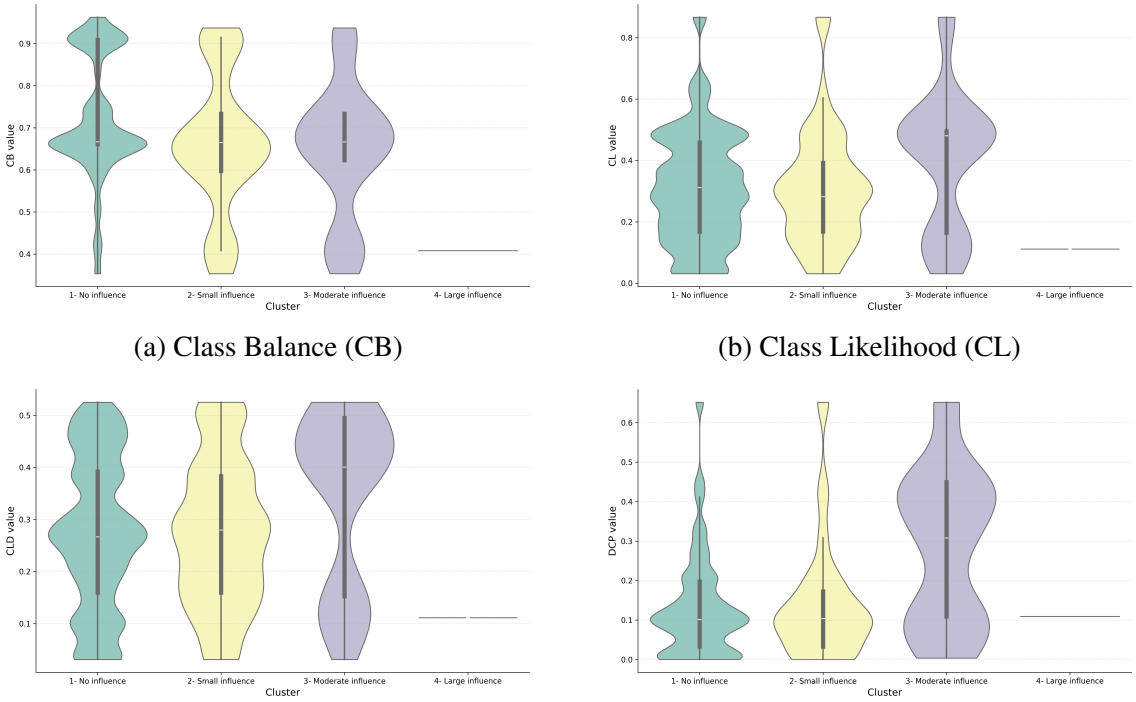


Figure H.6: Correlation between number of Instances and mean F1-score difference for each of the 42 datasets.



APPENDIX I — DISTRIBUTION OF HARDNESS MEASURES

Figure I.1: Distribution of complexity features across clusters (Part 1).



(a) Class Balance (CB)

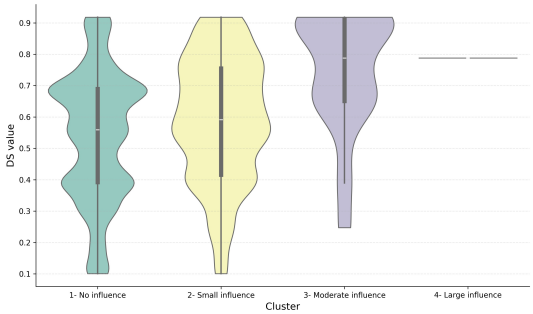
(b) Class Likelihood (CL)

(c) Class Likelihood Difference (CLD)

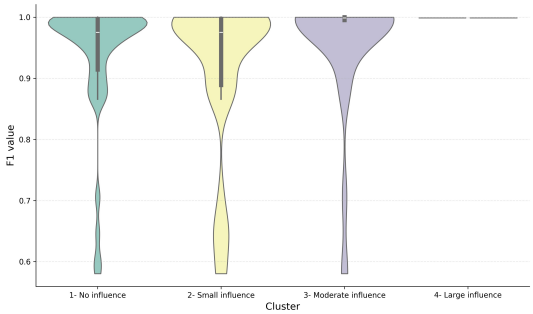
(d) Disjunct Class Percentage (DCP)

Source: The Author

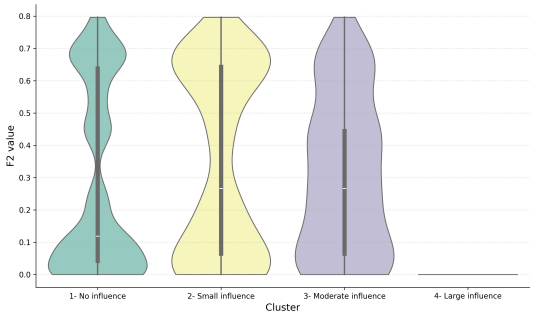
Figure I.2: Distribution of complexity features across clusters (Part 2).



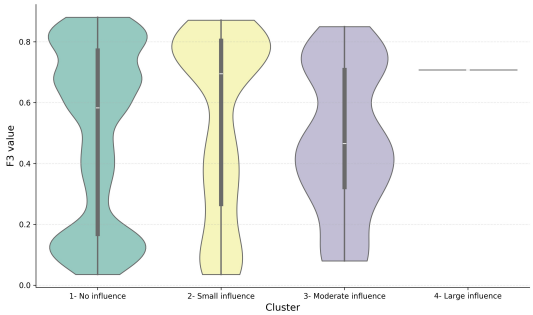
(a) Disjunct Size (DS)



(b) Overlapping Feature Fraction (F1)



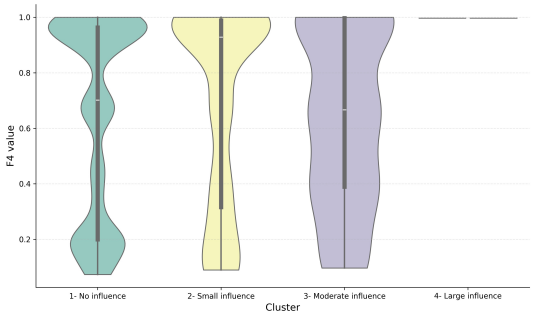
(c) Minimum Feature Overlap (F2)



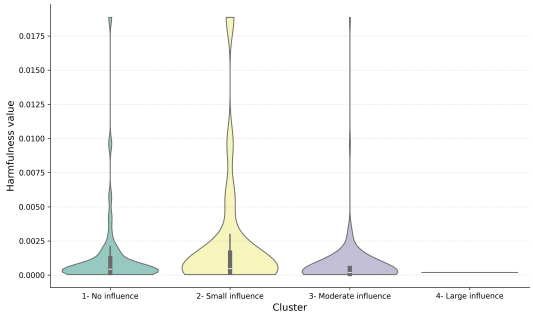
(d) Mean Feature Overlap (F3)

Source: The Author

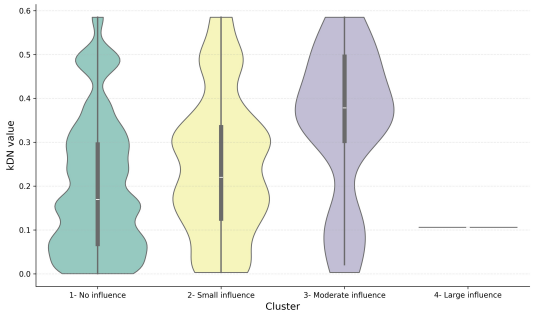
Figure I.3: Distribution of complexity features across clusters (Part 3).



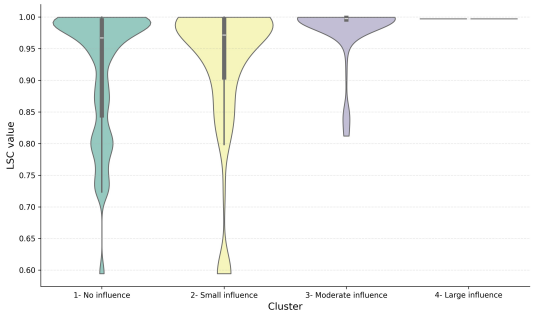
(a) Maximum Feature Overlap (F4)



(b) Harmfulness (H)



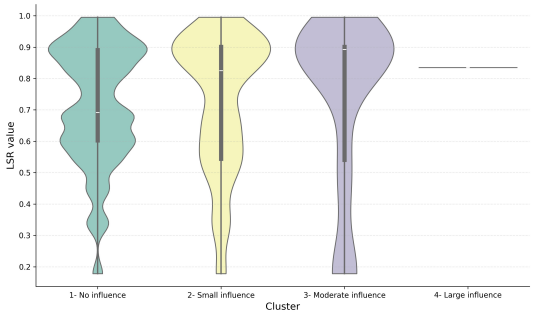
(c) k-Disagreeing Neighbors (kDN)



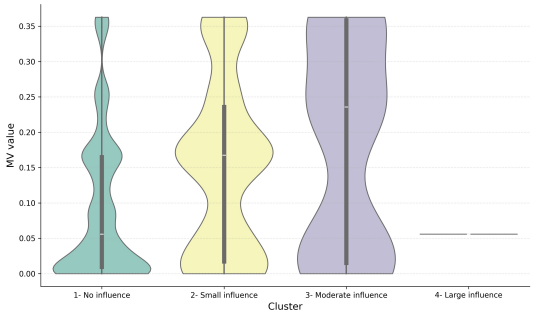
(d) Local Set Cardinality (LSC)

Source: The Author

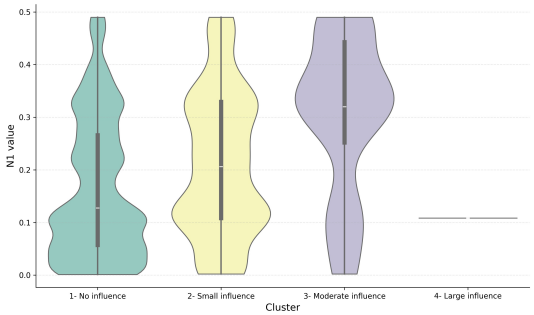
Figure I.4: Distribution of complexity features across clusters (Part 4).



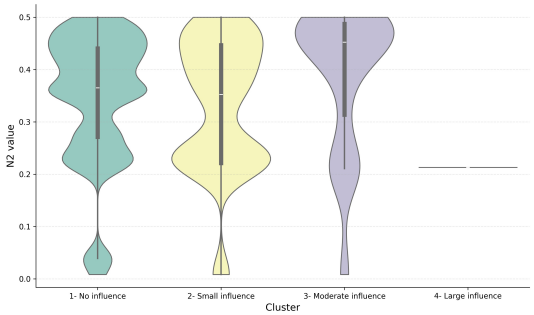
(a) Local Set Radius (LSR)



(b) Minority Value (MV)



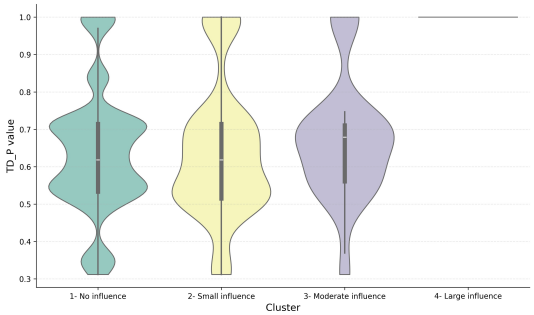
(c) Borderline Points (N1)



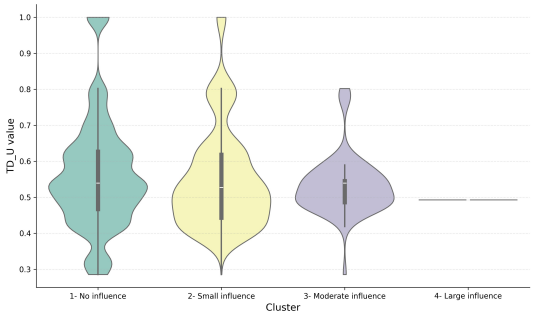
(d) Intra-Extra Ratio (N2)

Source: The Author

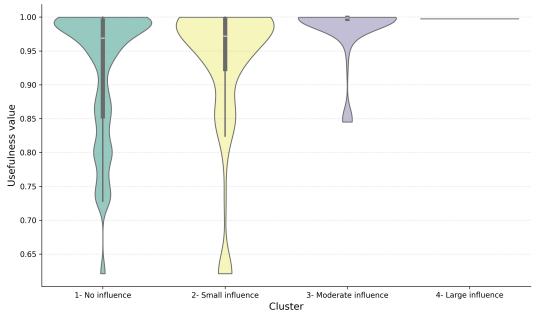
Figure I.5: Distribution of complexity features across clusters (Part 5).



(a) Tree Depth Pruned (TD_P)



(b) Tree Depth Unpruned (TD_U)



(c) Usefulness (U)

Source: The Author

APPENDIX J — DATASETS AND AVERAGE HARDNESS MEASURES

Table J.1: Datasets and their average value for the hardness measures (Part 1)

Dataset	kDN	DS	DCP	TD_P	TD_U	CL	CLD	MV	CB
adult	0.22	0.8381	0.2122	0.7191	0.4156	0.3145	0.3145	0.1647	0.5778
agaricus_lepiota	0.0006	0.3038	0	0.7013	0.7013	0.2405	0.2405	0.0354	0.6662
analcatdata_authorship	0.0119	0.3265	0.1013	0.846	0.7389	0.0581	0.0453	0.1708	0.75
banana	0.1321	0.6895	0.1412	0.7085	0.2854	0.4664	0.4664	0.084	0.6631
chess	0.1545	0.3993	0.0116	0.3452	0.3224	0.4389	0.4389	0.0407	0.666
clean1	0.2353	0.1006	0	1	1	0.317	0.317	0.1002	0.661
clean2	0.0678	0.1261	0	1	1	0.1984	0.1984	0.126	0.5072
coil2000	0.1061	0.7878	0.1093	1	0.4929	0.1113	0.1113	0.0559	0.4081
connect_4	0.3895	0.9101	0.4494	0.7106	0.5397	0.4833	0.4278	0.2359	0.6232
dis	0.0208	0.3921	0.0164	0.3119	0.4434	0.031	0.031	0.0151	0.3535
dna	0.3263	0.6463	0.1041	0.5012	0.4689	0.1087	0.1021	0.2582	0.7112
GAMETES_Epistasis...1	0.4965	0.6507	0.308	0.5608	0.5898	0.4948	0.4948	0	0.6667
Hill_Valley_with_noise	0.4927	0.606	0.1298	0.3692	0.4185	0.4998	0.4998	0	0.6667
Hill_Valley_without_noise	0.4872	0.682	0.0219	0.5333	0.4895	0.4995	0.4995	0.0097	0.6666
kr_vs_kp	0.1551	0.3993	0.0116	0.3452	0.3224	0.4584	0.4584	0.0407	0.666
krkopt	0.5851	0.8785	0.6517	0.6307	0.5452	0.8666	0.5252	0.3627	0.9016
letter	0.1747	0.7073	0.336	0.5597	0.4522	0.528	0.3674	0.082	0.9629
magic	0.2154	0.7655	0.2161	0.6498	0.5388	0.3831	0.3831	0.1612	0.637
mfeat_factors	0.065	0.3002	0.0691	0.5156	0.4996	0.1388	0.106	0	0.9091
mfeat_fourier	0.255	0.5591	0.2588	0.5954	0.5285	0.3691	0.2802	0	0.9091
mfeat_karhunen	0.0883	0.4636	0.1166	0.6107	0.6166	0.1089	0.0838	0	0.9091
mfeat_pixel	0.0476	0.3623	0.0474	0.735	0.6405	0.136	0.1019	0	0.9091
mnist	0.0854	0.6816	0.2739	0.7177	0.5499	0.6509	0.4611	0.0874	0.9089
molecular_biology_promoters	0.2953	0.4157	0.1047	0.7099	0.6399	0.1821	0.1821	0	0.6667
movement_libras	0.4683	0.5298	0.033	0.5319	0.5428	0.6034	0.3753	0	0.9375
mushroom	0.0003	0.1857	0	0.5962	0.5962	0.211	0.211	0.0334	0.6662
optdigits	0.0337	0.5297	0.0962	0.5456	0.5684	0.344	0.2666	0.0174	0.9091
phoneme	0.1758	0.7545	0.1722	0.4759	0.4135	0.3116	0.3116	0.1716	0.6098
ring	0.3369	0.6311	0.1216	0.5625	0.5019	0.0311	0.0311	0.0095	0.6666
satimage	0.1259	0.5677	0.1595	0.5226	0.5275	0.3532	0.2799	0.2038	0.8374
shuttle	0.0031	0.2473	0.0037	0.7468	0.7806	0.1682	0.1595	0.1784	0.4326
sleep	0.3347	0.918	0.4114	0.6792	0.4861	0.4803	0.4009	0.3583	0.7334
sonar	0.2841	0.3894	0.0796	0.97	0.8022	0.3922	0.3922	0.0588	0.6652
soybean	0.1698	0.5581	0.0765	0.5149	0.3964	0.2814	0.2285	0.3411	0.9149
spambase	0.1498	0.591	0.096	0.6521	0.5185	0.2581	0.2581	0.1378	0.6517
spectf	0.3006	0.5052	0	0.6182	0.6182	0.2816	0.2816	0.1704	0.5975
splice	0.3785	0.6889	0.0996	0.5636	0.4674	0.1645	0.1526	0.2582	0.7112
texture	0.0399	0.4553	0.078	0.5515	0.43	0.3791	0.2903	0	0.9167
tokyo1	0.0949	0.3702	0.0998	0.7143	0.6626	0.2104	0.2104	0.1571	0.6408
twonorm	0.0641	0.676	0.1462	0.8314	0.6909	0.0333	0.0333	0.0008	0.6667
waveform_21	0.2408	0.7225	0.2084	0.6641	0.6265	0.268	0.2552	0.0171	0.75
waveform_40	0.2851	0.6914	0.1975	0.6777	0.5935	0.2784	0.2641	0.0149	0.75

Table J.2: Datasets and their average value for the hardness measures (Part 2)

Dataset	N1	N2	LSC	LSR	Harmfulness	Usefulness	F1	F2	F3	F4
adult	0.2177	0.3523	0.9959	0.9175	0.0001	0.996	0.9974	0.6646	0.7715	0.969
agaricus_ lepiota	0.0015	0.0414	0.7678	0.6747	0.0002	0.7681	0.9687	0	0.6653	0.9453
analcatdata_ authorship	0.0078	0.4345	0.7234	0.3409	0.0018	0.7281	0.9064	0.2281	0.3887	0.4843
banana	0.1304	0.2199	0.9712	0.955	0.0004	0.9716	0.9961	0.7968	0.8499	0.903
chess	0.0994	0.3493	0.9928	0.8801	0.0006	0.9934	0.9944	0	0.5999	0.9301
clean1	0.1284	0.3695	0.9649	0.7682	0.0042	0.9691	0.9595	0.0778	0.8045	0.998
clean2	0.0573	0.2378	0.9852	0.8254	0.0003	0.9855	0.9752	0.0852	0.8148	0.9984
coil2000	0.1083	0.213	0.9971	0.8348	0.0002	0.9973	0.999	0	0.7077	0.9968
connect_ 4	0.3569	0.4713	0.9996	0.8931	8.22e-05	0.9997	1	0.4444	0.4656	0.6667
dis	0.0213	0.2109	0.9669	0.8609	0.0005	0.9674	0.9648	0	0.3738	0.9672
dna	0.2899	0.4119	0.995	0.4551	0.0005	0.9959	0.9995	0.1687	0.4429	0.4444
GAMETES_ Epistasis_.. 1_ 1	0.4899	0.4999	0.9975	0.1781	0.0013	0.9987	1	0.6547	0.7782	1
Hill_ Valley_ with_ noise	0.3981	0.4896	0.9965	0.9928	0.0017	0.9981	0.9973	0.6886	0.6991	0.7088
Hill_ Valley_ without_ noise	0.3695	0.4467	0.996	0.9953	0.0017	0.9977	0.9972	0.688	0.6949	0.7015
kr_ vs_ kp	0.1622	0.3493	0.9927	0.8801	0.0006	0.9933	0.9944	0	0.5999	0.9301
krkopt	0.4575	0.4873	0.9962	0.9473	0.0001	0.998	0.8905	0.0645	0.0798	0.0967
letter	0.0864	0.2814	0.9436	0.8123	0.0001	0.9462	0.9145	0.0471	0.0621	0.0733
magic	0.2063	0.4147	0.9949	0.9155	0.0002	0.9951	0.9966	0.6926	0.8359	0.9749
mfeat_ factors	0.0554	0.365	0.799	0.6055	0.0006	0.804	0.8658	0.1018	0.1595	0.195
mfeat_ fourier	0.2128	0.4402	0.8912	0.5341	0.0006	0.8962	0.9325	0.1174	0.1646	0.1942
mfeat_ karhunen	0.0509	0.4132	0.8672	0.4409	0.0006	0.8722	0.9446	0.119	0.1687	0.1947
mfeat_ pixel	0.0391	0.3631	0.7867	0.6204	0.0006	0.7917	0.9896	0.0422	0.1389	0.2
mnist	0.0577	0.3932	0.955	0.6881	4.69e-05	0.9558	0.9962	0	0.0858	0.1981
molecular_ biology_ promoters	0.2437	0.4643	0.9046	0.3369	0.0189	0.9235	0.9975	0.6431	0.7539	0.9286
movement_ libras	0.277	0.3136	0.8119	0.6832	0.003	0.8536	0.7057	0.0655	0.0951	0.1226
mushroom	0.0012	0.0392	0.7478	0.6682	0.0002	0.748	0.9857	0	0.6612	0.9635
optdigits	0.0207	0.3497	0.7984	0.6026	0.0002	0.8002	0.9799	0	0.1162	0.1965
phoneme	0.1181	0.2463	0.9716	0.8937	0.0004	0.9719	0.973	0.7433	0.8561	0.9512
ring	0.321	0.4562	0.9722	0.6036	0.0003	0.9725	0.9602	0.6923	0.871	0.9883
satimage	0.1001	0.3645	0.8697	0.811	0.0002	0.8707	0.871	0.2106	0.2668	0.3162
shuttle	0.0022	0.0084	0.8446	0.934	7.53e-05	0.845	0.5804	0.1176	0.1796	0.2287
sleep	0.3198	0.4523	0.9988	0.9008	4.32e-05	0.999	0.9954	0.2663	0.3229	0.39
sonar	0.1757	0.4133	0.9314	0.6127	0.0096	0.941	0.9777	0.638	0.8034	0.9902
soybean	0.1278	0.2213	0.5945	0.5441	0.0016	0.6211	0.6469	0.0044	0.0352	0.0897
spambase	0.1024	0.2714	0.9886	0.8907	0.0004	0.989	0.9903	0.5593	0.6769	0.8618
spectf	0.3312	0.2464	0.9398	0.6835	0.0057	0.9456	0.889	0.5531	0.8073	0.9943
splice	0.3293	0.3864	0.9954	0.5171	0.0005	0.9964	0.996	0.4159	0.5191	0.6666
texture	0.018	0.2789	0.7308	0.7469	0.0002	0.7328	0.6015	0.0875	0.1212	0.1513
tokyo1	0.0973	0.2273	0.8221	0.8401	0.0021	0.8242	0.8699	0	0.6754	0.9255
twonorm	0.0578	0.439	0.9794	0.582	0.0003	0.9796	0.9948	0.7208	0.8806	0.9892
waveform_ 21	0.2335	0.4716	0.9911	0.6919	0.0003	0.9917	0.9546	0.4667	0.5729	0.6489
waveform_ 40	0.2658	0.4868	0.9962	0.5569	0.0003	0.9968	0.973	0.4596	0.5831	0.6592

APPENDIX K — DISTRIBUTION OF F1-SCORE DIFFERENCES FOR HIGHEST IMPACTED DATASETS IN EACH PIPELINE STEP

Figure K.1: Distribution of F1-Score differences for datasets with highest $\overline{\Delta F1}$ for the hyperparameter tuning step.

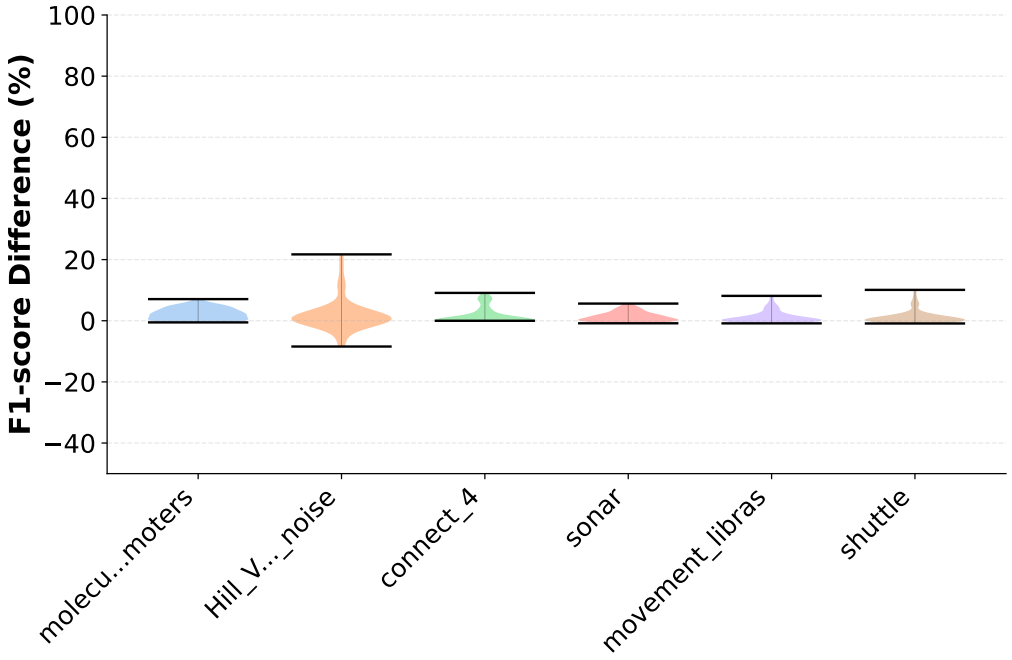


Figure K.2: Distribution of F1-Score differences for datasets with highest $\overline{\Delta F1}$ for the feature selection step.

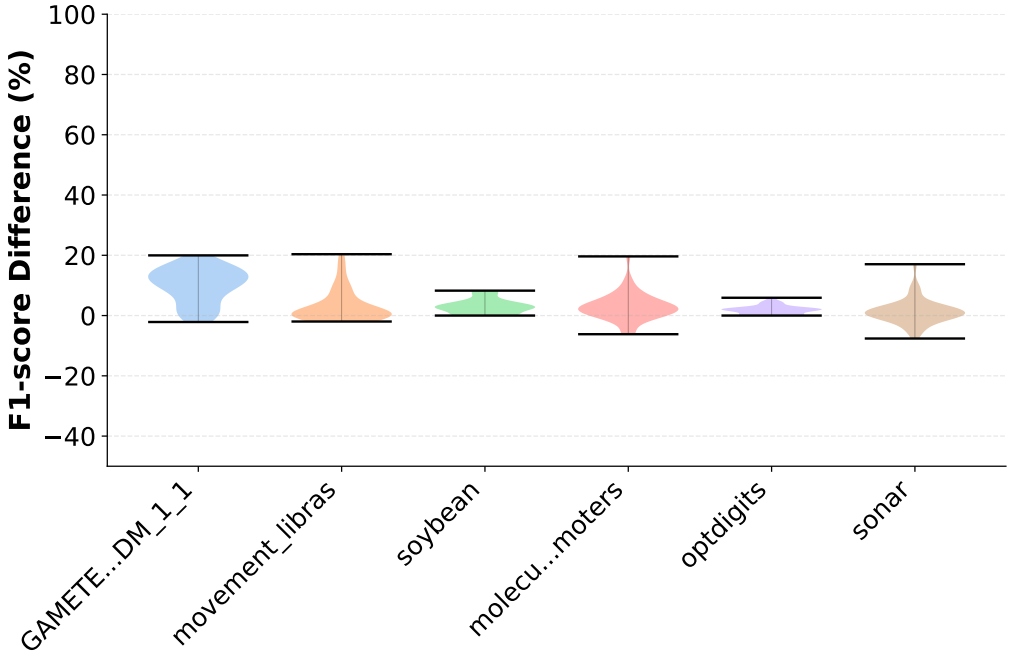


Figure K.3: Distribution of F1-Score differences for datasets with highest $\overline{\Delta F1}$ for the missing-value imputation step.

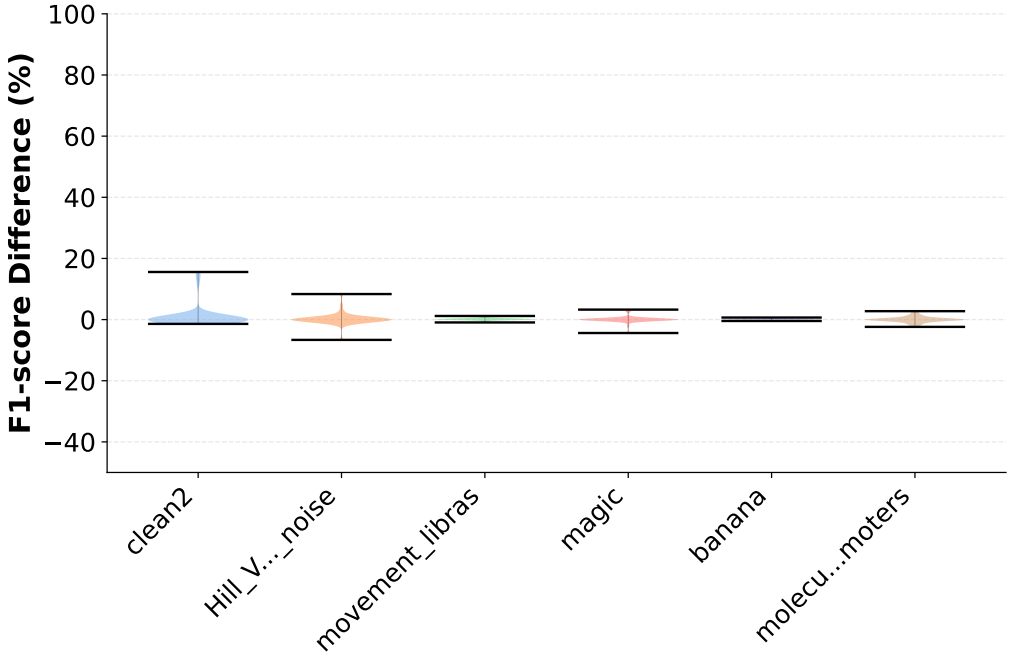


Figure K.4: Distribution of F1-Score differences for datasets with highest $\overline{\Delta F1}$ for the normalization step.

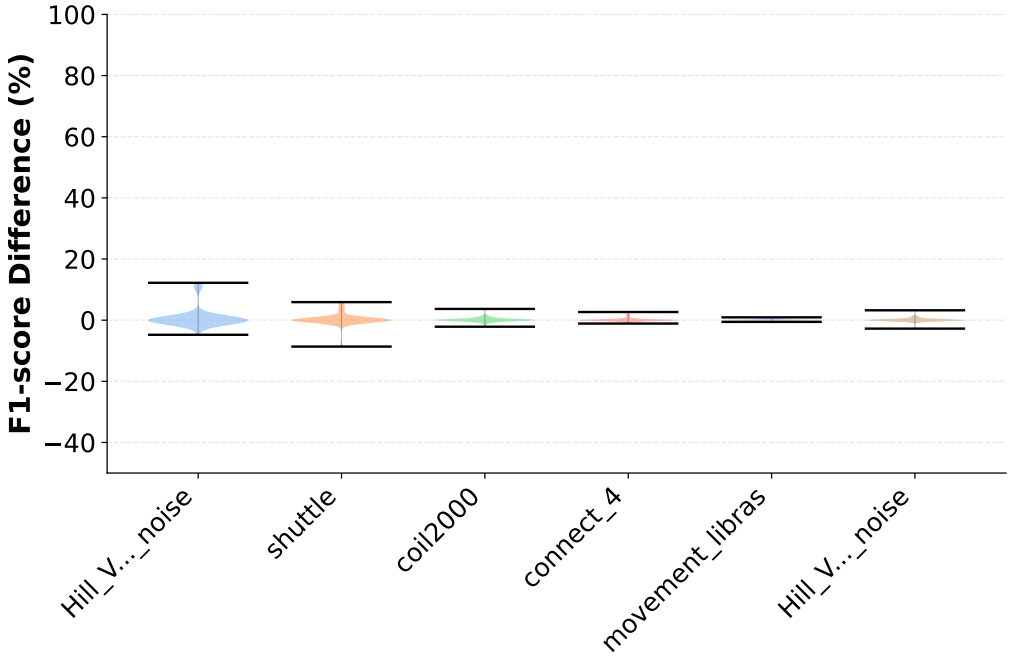
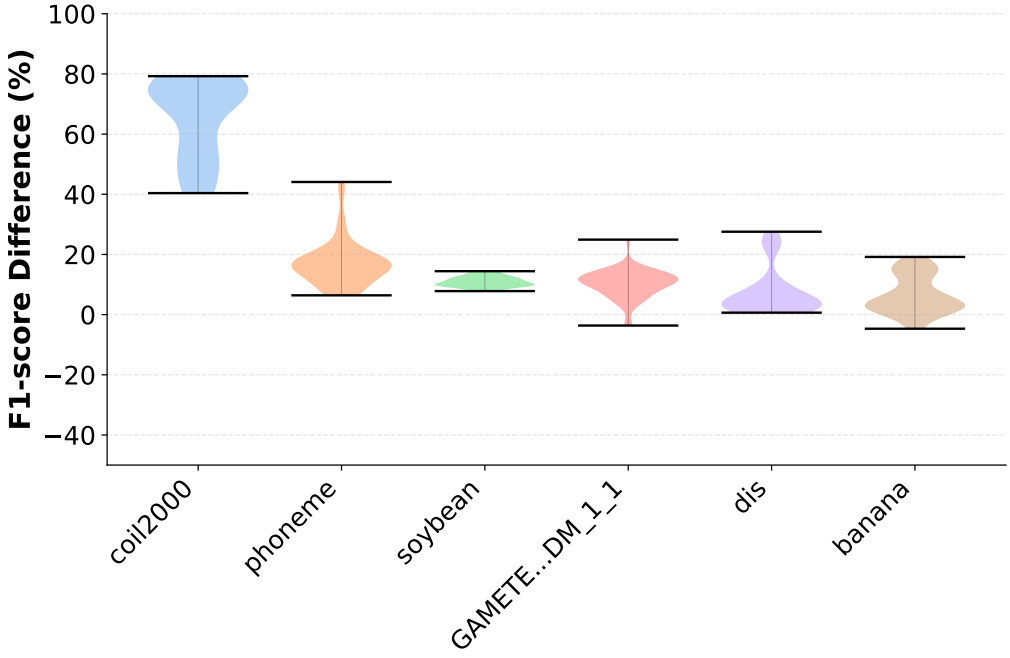
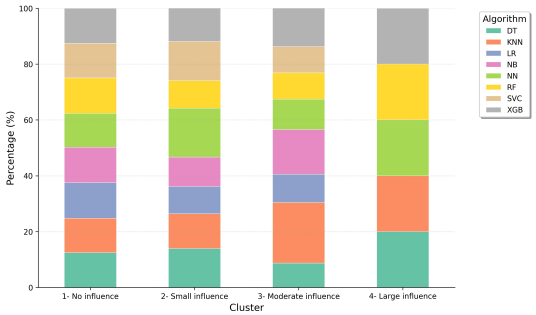


Figure K.5: Distribution of F1-Score differences for datasets with highest $\overline{\Delta F1}$ for the worst-case scenario.

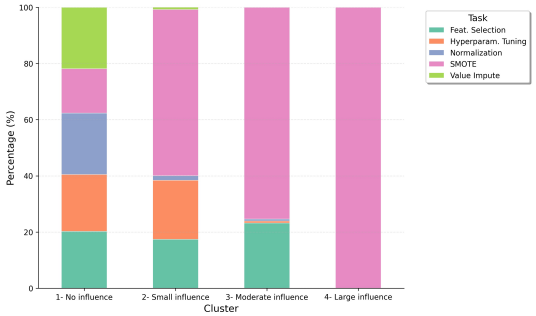


APPENDIX L — CLUSTER ANALYSIS CONSIDERING THE WORST-CASE SCENARIO

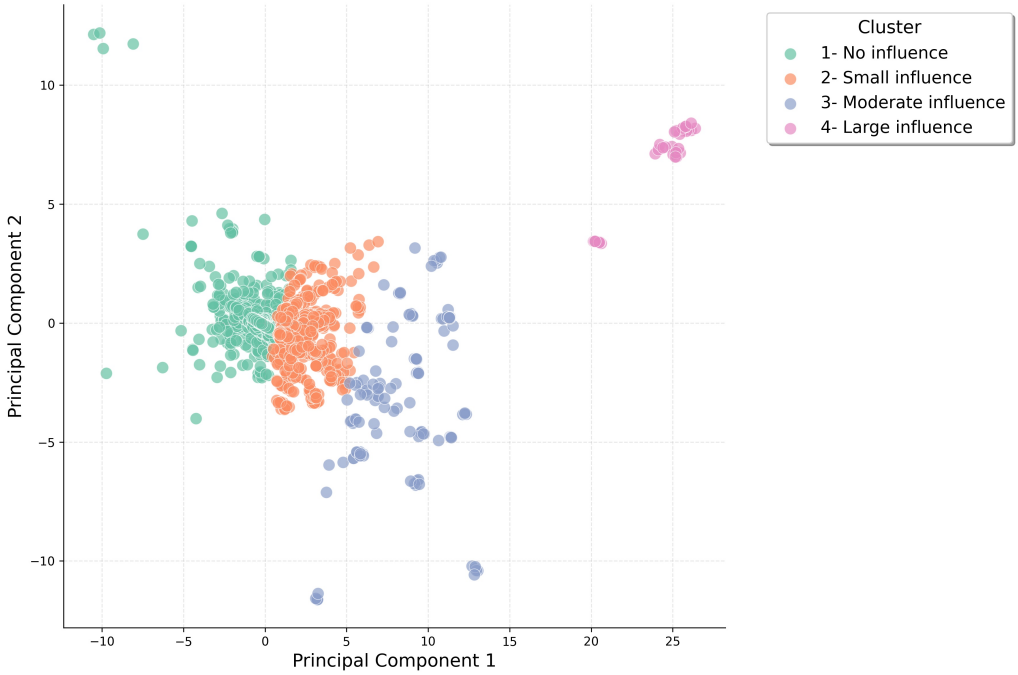
Figure L.1: Cluster analysis overview showing distribution of algorithms, pipeline steps, and PCA visualization. These results are obtained considering the experiments for the worst-case scenario.



(a) Algorithm distribution by cluster



(b) Task distribution by cluster



(c) PCA visualization of clusters with performance metrics

Source: The Author