UNIVERSITAT DE BARCELONA

FUNDAMENTAL PRINCIPLES OF DATA SCIENCE MASTER'S THESIS

Machine Learning Copies as a Means for Black Box Model Evaluation

Author:
Dr. Muriel ROVIRA-ESTEVA

Supervisors:
Dr. David ZEBER
Dr. Oriol PUJOL

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamental Principles of Data Science

in the

Facultat de Matemàtiques i Informàtica

September 2, 2021

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Machine Learning Copies as a Means for Black Box Model Evaluation

by Dr. Muriel ROVIRA-ESTEVA

The use of propietary black-box machine learning models and APIs in the form of Machine Learning as a Service, makes it very difficult to control and mitigate their potential harmful effects (such as lack of transparency, privacy safeguards, robustness, reusability or fairness). The state-of-the-art technique of Machine Learning Classifier Copying allows us to build a new model that replicates the decision behaviour of an existing one without the need of knowing its architecture nor having access to the original training data.

PRESC (Performance and Robustness Evaluation for Statistical Classifiers) is an existing free software tool for the evaluation of machine learning classifiers, which is maintained by Mozilla's Data Science team. It aims to provide a toolkit to analyze model performance beyond the standard accuracy-based methods and into areas which tend to be underexplored in data science practice.

Among the multiple applications of Machine Learning Classifier Copying, a systematic construction and examination of model copies has the potential to be an universally accessible and inexpensive approach to study and evaluate a rich variety of original models, and to help understand its behavior. In this work, an implementation of Machine Learning Classifier Copying has been contributed to the PRESC project, so that this tool becomes readily accessible to researchers and practitioners, and its applicability and performance in a synthetic problem has been explored to understand the copying process. The solution provides a model agnostic sampling strategy and an automated copy process for a number of fundamentally different hypothesis spaces, so that the set of achievable copy-model-fidelity measures can be used as a diagnostic measure of the original model characteristics.

Acknowledgements

Many thanks to my two supervisors for their kind help and friendly guidance. Their experience and insight was key to provide me with the best of two worlds, the non-profit free software development community and academia, that often have the same goals but very different approaches.

Also very special thanks to those who have fed and encouraged me through the developing and writing process. As always, your love is essential for any success.

Contents

Chapter 1

Introduction

1.1 Motivation

Data science has experienced a boom in recent years which has made it widely popular and even fashionable. Companies, governments, media, and even the general public... everyone wants to do data science now. Such popularity has been followed by an explosion of available teaching materials, resources and tools which have democratized its practice by individuals or organisations with limited resources and has facilitated a large pool of self-taught professionals.

However, it has been suggested theoretically that "the reliability of findings published in the scientific literature decreases with the popularity of a research field", and a number of authors have provided with empirical evidence supporting this claim [1–3]. Data science is not an exception to this problem, neither in the academic nor the corporate environments, something which has been reflected by growing concerns in the specialized literature [4].

Proper model evaluation is an essential step that is often skipped through, especially by more inexperienced practitioners or those lacking a solid base in the discipline, but surprisingly also in other environments. As long as the model provides an answer, that is enough. However, accountability is a very important aspect of data science practice and, even assuming no wrongdoing, a lack of thorough understanding of the model's behaviour can lead to extremely damaging social, economical or personal consequences. Lack of transparency, privacy safeguards, robustness, reusability or fairness, just to name a few, are omnipresent problems in the application of machine learning solutions, and there is still a limited availability of resources that allow to mitigate such shortcomings. Hence, developing methodologies and tools which allow for advanced model evaluation and mitigation strategies is clearly of great interest in the field.

One of such methodologies recently developed is Machine Learning Classifiers Copying, which allows us to build a new model that replicates the decision behaviour of an existing one without the need of knowing its architecture nor having access to the original training data [5–9]. Among the myriad of potential applications of Machine Learning Classifier Copying, a systematic construction and examination of model copies has the potential to be an universally accessible and inexpensive approach to study and evaluate a rich variety of original models, and to help understand their behavior. A suitable copy allows to audit the already deployed model, mitigate its shortcomings, and even introduce improvements, without the need to build a new model from scratch, which would require access to the original data, often too expensive or simply impossible. However, Machine Learning Classifier Copying is currently still under active research efforts and has not permeated yet from the academic domain into the general public.

The Mozilla Foundation has focused its activism in the recent years on campaigning for trustworthy AI, a broad term that also includes Machine Learning. Part of these efforts are the publication of a white paper on how to build Trustworthy AI, in order to encourage best practices in the industry [10], and another part has been the development of free software tools based in AI technologies and that comply with certain ethical standards. One of the toolkits being developed, PRESC or Performance and Robustness Evaluation for Statistical Classifiers, aims to provide evaluation tools for machine learning classifiers beyond the standard accuracy-based methods. Developing those tools as free software is not inconsequential. Apart from the enhanced security and accountability, it facilitates a universal access to advanced methodologies that would be otherwise reserved to a few.

1.2 Objectives

One of the main goals of this work is to contribute to the technology transfer of Machine Learning Classifier Copying, which is still being researched and used only in an academic environment, to the general public, and help make this technology accessible and universally available not only to other researchers but also to all data science practitioners.

Another important goal is to explore the application of the implementation of this technique in order to better understand the copying process from an academic point of view, and to determine how different characteristics both of the original problem and of the copying procedure affect the fidelity and the performance of the copy.

1.3 Contributions

In this work, an implementation of Machine Learning Classifier Copying has been contributed to the PRESC project, covering the whole pipeline in a modular way and with a number of possible options, so that this tool becomes readily accessible to researchers and practitioners as is but can also be easily extended. The solution provides a model agnostic sampling strategy and an automated copy process for a number of fundamentally different hypothesis spaces, so that the set of achievable copy-model-fidelity measures can be used as a diagnostic measure of the original model characteristics. Among other basic samplers, a spherical sampler which ensures balanced datasets for the copies and not previously mentioned in the literature was also implemented.

To ensure the long-term availability and sustainability of this work, it has been carried out following the free software development methodology and coding practices, and complying with certain code quality standards. This includes an emphasis on code readability and testing to facilitate reviewing and maintenance, in the form of thorough documentation and a collection of unit tests that accompany the code.

Additionally, in order to better understand the copying process, a systematic exploration of the effect on the performance and fidelity of the copy for different problems and copy parameters has been carried out (class separation and dimensionality of the problem, type of original model, choice of model for the copy, choice of sampler, and number of synthetic samples generated). A complex synthetic problem generator with tunable parameters has been implemented and used to prepare the datasets for the analysis, so that different degrees and also flavours of complexity could be easily adjusted before performing the copies.

1.4 Layout

The structure of this report is as follows: First, it begins with an introduction, where the motivation of this work, the objectives, and the contributions are explained. Then, a summary of the theoretical background to Machine Learning Classifiers Copying is provided, in order to have a solid foundation to develop the rest of the work. Afterwards, a proposal chapter to explain the methodologies that have been followed is included, with special emphasis on the Machine Learning Classifier Copying implementation and the contribution methodology in free software. This is followed by an experimental chapter detailing the exploration of all the intricacies of a tunable synthetic problem carried out using this implementation, and the results of that exploration. Then, there is a summary of the conclusions. And, finally, an appendix explaining where to find the source code and supplementary materials, and the bibliography.

Chapter 2

Background

2.1 Machine Learning Classifier Copies

Machine learning classifier copying allows us to build a new model that replicates the decision behaviour of an existing one while including the possibility of adding new features, such as interpretability, online learning or equity, under the assumptions that the original training data is unknown or lost, access to the model is limited to a membership query interface (i.e., its internal architecture is not known), and that the query interface only produces hard predictions. This corresponds to a black box model scenario where distillation is conducted without the original data [5].

Simultaneous optimization of the synthetic data and the copy parameters requires online updating of the hypothesis space, among other properties. Hence, only the simplest optimization approach, the single-pass copy, is discussed by Unceta, Nin, and Pujol [5]. In this simpler case, the steps to carry out the classifier copying can be split in two main groups, generating the synthetic data and building the copy:

1. Define the range of interest of the features (it is necessary to have a notion of the dynamic range of all features).
2. Generate random data in the feature space. The distribution will depend on the particular problem and the sampling approach.
3. Label this synthetic data using the prediction results from queries to the original model.
4. Choose a hypothesis space for the copy model according to the goals of each particular case and its desired capacity.

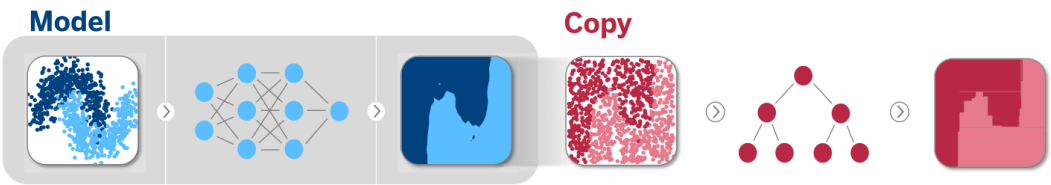

FIGURE 2.1: Example of a single-pass copy. The general assumption is that we have no access to the original data nor model characteristics (grey box), but we can generate new unlabelled data and use the original model to label it through a query interface. The generated data can then be used to train a copy model that reproduces the decision boundary of the original, albeit with additional properties of interest or incorporating some improvement. From Ref. [5].

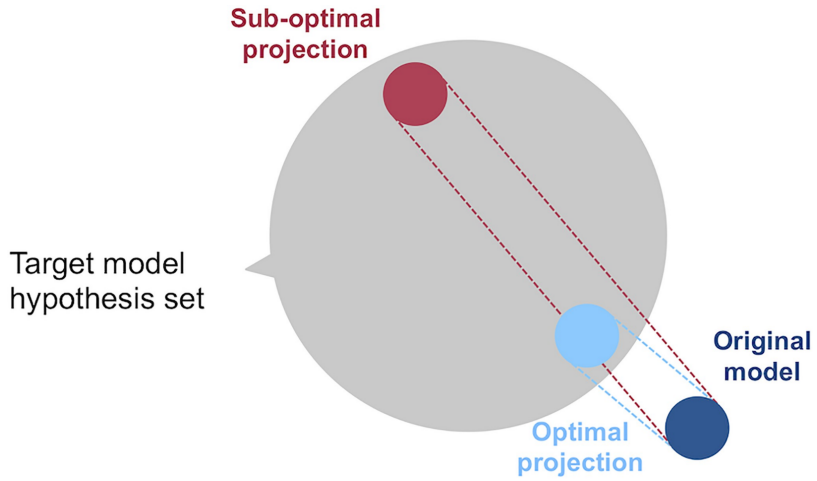

FIGURE 2.2: Original model and projection in a new hypothesis space. The projection closest to the original model is considered to be the optimal. From Ref. [8].

5. Train the copy model with the generated data.

2.2 Theoretical principles

The process of copying can be interpreted as projecting the original decision function $f_{\mathcal{O}}$ from its hypothesis space $\mathcal{H}_{\mathcal{O}}$ into a chosen new hypothesis space $\mathcal{H}_{\mathcal{C}}$ that the copy decision function $f_{\mathcal{C}}$ belongs to (see Figure 2.2) [5].

We want to obtain a copy $f_{\mathcal{C}}$ with the same predictive distribution for new data $P\left(y^*|f_{\mathcal{O}}, x^*\right)$ as the original decision function $f_{\mathcal{O}}$, where x^* is the new data points input, and y^* their label.

To determine the predictive distribution of this decision function we marginalise with respect to the model copy parameters θ, i.e., we add the individual probability contributions of each possible value of the parameters:

$$P\left(y^*|f_{\mathcal{O}}, x^*\right) = \int_{\theta \in \mathcal{H}_{\mathcal{C}}} P\left(y^*|\theta, f_{\mathcal{O}}, x^*\right) P\left(\theta|f_{\mathcal{O}}, x^*\right) d\theta \tag{2.1}$$

Then we make two basic assumptions:

- $P\left(\theta|f_{\mathcal{O}}, x^*\right) = P\left(\theta|f_{\mathcal{O}}\right)$, because data about the unseen point x^* is not available.
- $P\left(y^*|\theta, f_{\mathcal{O}}, x^*\right) = P\left(y^*|\theta, x^*\right)$, because interaction with the classifier $f_{\mathcal{O}}$ is no longer required once the copy is built and the θ values are fixed.

If we introduce these two assumptions in Equation 2.1 and force the posterior to have the form $P\left(\theta|f_{\mathcal{O}}\right) = \delta(\theta - \theta^*)$, i.e., that all probability mass is due to the θ^* contribution, corresponding to the optimal values of the model parameters, we obtain:

$$P\left(y^*|f_{\mathcal{O}}, x^*\right) = P\left(y^*|\theta^*, x^*\right) \tag{2.2}$$

Hence, to make a classifier copy, the optimal model parameters that maximize the posterior probability have to be found:

$$\theta^* = \arg\max_{\theta} P\left(\theta|f_{\mathcal{O}}\right) \tag{2.3}$$

Unfortunately, this maximization cannot be conditioned to $f_{\mathcal{O}}$, but we can introduce a set of synthetic data z in a domain $\mathcal{Z}$ and then maximize:

$$\int_{z \in \mathcal{Z}} P(\theta|z)\, dz \tag{2.4}$$

However, the usual trainig algorithms are not the most adequate to carry out classifier copies, because they normally have built-in strategies to avoid overfitting and, in the case of classifier copying, the problem will be separable by construction so we may in fact be trying to induce certain type of overfitting, and these mechanisms might prevent us from obtaining the identical boundary that we seek.

2.3 Evaluation

The fidelity error captures the loss of copying, and in its general form is the probability that the copy resembles the model. It evaluates the disagreement between $f_{\mathcal{O}}$ and $f_{\mathcal{C}}$ through the percentage error of the copy over a given set of data points, taking the predictions of the original model as ground truth [7]. In the case of the $0/1$ loss over the generated dataset $\mathcal{Z}$, the empirical fidelity error can be written as

$$R_{\text{emp}}^{\mathcal{F},\mathcal{Z}} = \frac{1}{N} \sum_{j=1}^{N} \mathbb{I}\left[f_{\mathcal{O}}(z_j) \neq f_{\mathcal{C}}(z_j)\right] \tag{2.5}$$

where N is the number of samples and z_j are the generated data points.

Since the synthetic dataset is always separable, it is always possible to achieve zero empirical error, given we choose a copy model with enough capacity. And since it is theoretically possible to generate an infinite amount of synthetic data, the generalization error can asymptotically also be reduced to zero. Hence, copying can be in theory carried out without any loss. In practice, however, the generated dataset is invariably finite.

A low empirical fidelity error does not guarantee a good copy. In addition to that, the generated dataset must ensure a good coverage of the input space, and any volume imbalance effect needs to be accounted for as well.

The copy accuracy metric is introduced to evaluate the generalization performance of the copy. In the ideal case, the fidelity error is zero and the accuracy of the copy is the same as that of the original classifier. The copy will be of higher quality when it mimics the orginal model exactly, including its misclassifications. When performing a copy, we do not aim for an improvement of the original model performace, but to obtain the exact same behavior.

The choice of copy hypothesis can have a major influence on the performance, if there is a mismatch between the capacity of the original model and the capacity of the copy, this may yield a large capacity error and poor performance results [5].

2.4 Sampling

Since the general assumption is that the original training data is not accessible, new data must be generated in order to solve Equation 2.3 and find the parameters of the copy that maximize the posterior probability [5].

In the general case, the distribution of this generated data does not have to match the original training data, and the choice of an adequate distribution can be challenging.

In the particular case that we do have access to the distribution of the original training data, it is desirable to generate the new data with the same distribution, to ensure that the copy replicates the behavior of the original classifier, especially where the training data lies.

Since our goal is to replicate the boundaries of the original classifier it is good to have a good coverage of those regions. However, it is common that the boundary between the classes lies far from the maximum peaks of their distributions, in minima between them, which is an area that with the original distribution would have little amount of samples.

Another challenge when generating the synthetic dataset is volume imbalance between the regions occupied by each class, because the empirical fidelity error depends on the fraction occupied by each volume. This problem can be mitigated through an appropriate choice of the generated data distribution or imposing a balanced result to the final dataset.

Unceta et al. [7] have tackled the sampling problem in the context of machine learning classifier copies, where they explored and evaluated the strengths and weaknesses of several sampling strategies:

- **General random distribution**
- **Boundary exploration**: Uniform exploration until a point from a different class is found, then exploration of the line between the last two samples to find the boundary, which stops when points from the two classes are closer than a certain tolerance, then select a random direction to explore at constant steps, and repeat the process. Half the samples were generated using random sampling. Performs well for linear problems. It is also a good compromise between time and accuracy for problems with many dimensions.
- **Bayesian-based optimizer**: The optimization function corresponds to a random process with a term that tends to explore areas with a larger variance (most unexplored areas), and another term that tends to explore areas with class transitions (near boundaries). This method has a high computational cost due to the estimation of the mean and the covariance matrix. Hence, to overcome this problem, a faster approach was proposed which limits the number of samples used in this calculation to b, although it is also less accurate. This method yielded the best performance for fewer points. If the computational cost is not a determining factor, then Bayesian sampling yields the most reliable exploration of the space.
- **Modified version of the Jacobian-based Dataset Augmentation approach** proposed by Papernot et al. [11]: This method generates linear structures with a large number of samples and, hence, it has a larger uncertainty.
- **Random sampling**: The main advantage is that it samples across the space with equal probability but, for the same reason, it is indifferent to the structures of interest (i.e., the shape of the decision boundaries). It is also quite fast due to its low computational cost. This approach works well for decision trees. It also works best for a majority of problems when a large number of points are sampled. It does not work so well for problems with a large number of dimensions where the space can not be sampled exhaustively.

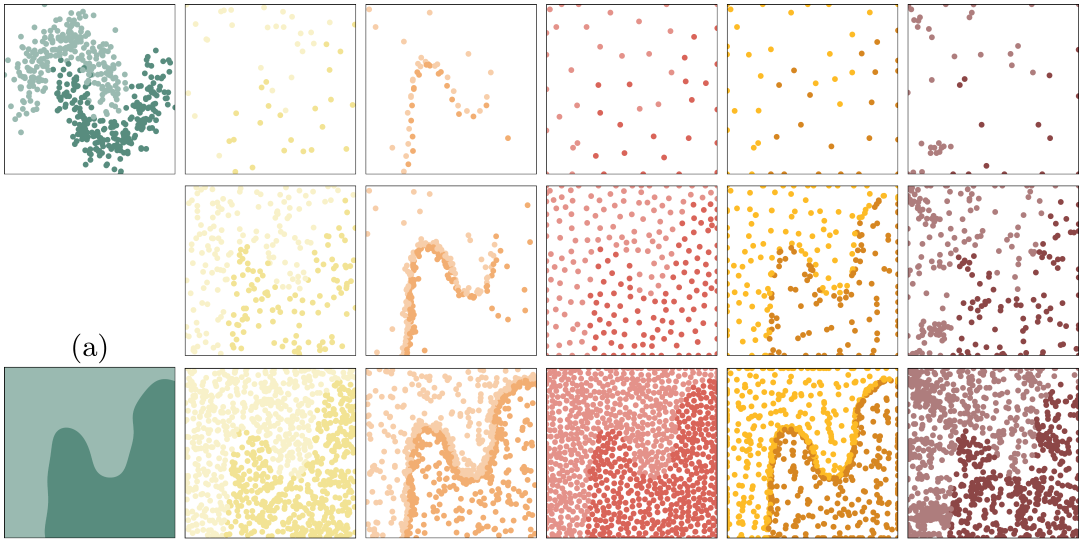

FIGURE 2.3: (a) Training dataset and (b) decision boundary learned by an SVM with a radial basis function kernel. Synthetic datasets of sizes 50, 250 and 1000 generated using (c) Random sampling, (d) Boundary sampling, (e) Fast Bayesian sampling, (f) reoptimized Bayesian sampling and (g) adapted Jacobian sampling. From Ref. [7].

Chapter 3

Methods and Proposal

3.1 Implementation of Machine Learning Classifier Copies

3.1.1 The PRESC model evaluation package

PRESC (Performance and Robustness Evaluation for Statistical Classifiers) is an already existing free and open-source software tool intended to be used by engineers for the evaluation of machine learning classifiers when developing or updating the models. It aims to provide additional performance insights beyond the standard accuracy-based measures, in particular about its generalizability to previously unseen data, sensitivity to error and small changes in the methodology, performance for subsets of the feature space, misclassification analysis and distribution [12, 13].

The package has been implemented in Python and is maintained by the Data Science team at Mozilla, although it receives contributions from the free software community worldwide. It has a free software Mozilla Public License, which means that the package can be distributed, modified, used commercially, and used for patenting or private uses, as long as the source is disclosed, a license and copyright notice are included, and the same license is used for any derivative material.

3.1.2 Outline of the contribution

Figure 3.1 shows a scheme of the Machine Learning Classifier Copying package that was contributed to Mozilla's free software project PRESC. Apart from implementing the copying pipeline and a number of basic samplers and options, there are two implemented features of the package that must be specially mentioned: the Spherical Balancer Sampler and the Multidimensional Multiclass Gaussian Problem Generator algorithm (see Figure 3.1).

Copying pipeline

To carry out the copy, the ClassifierCopy class needs two inputs: the original classifier to copy and, depending on the sampler, a dictionary with basic descriptors of the features. Right now the package assumes that we have the classifier saved as a sklearn-type model. The original data is not necessary to perform the copy but, if available, the "dynamic_range" function can conveniently extract the basic descriptors of its features into a dictionary. In this case, the data should be available as a pandas DataFrame. Otherwise, the dictionary with the basic feature descriptors can always be built manually. Even if we don't have access to the original data or detailed information of the features, we need at least to be able to make a guess or some assumptions about them.

When instantiating the ClassifierCopy class, an instance of a sklearn-type model to build the copy must also be specified, as well as the choice of sampling function

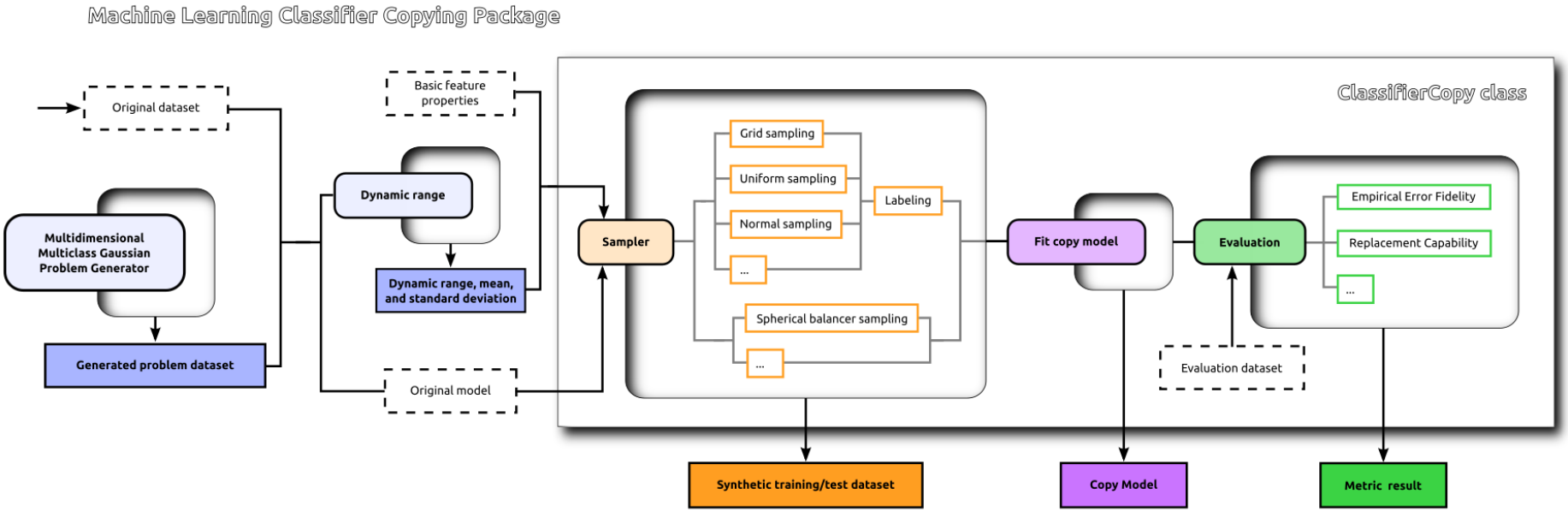

FIGURE 3.1: Scheme of the Machine Learning Classifier Copying Package.

and its options. The necessary feature descriptors for the sampler can be the maximum and minimum values that the features can take, like in the case of the grid and uniform samplers, the mean and standard deviation of each feature, such as in the normal sampler, or an overall minimum and maximum single value common to all features, as in the case of the spherical balancer sampler.

The "copy_classifier" method will generate synthetic data in the feature space using the sampling function and options specified on instantiation, will label it using the original classifier, and then will use it to train the desired copy model. The generated synthetic training data can be saved in this step if needed but it can also be recovered later using the "generate_synthetic_data" method simply using the same random seed.

After the copy has been obtained, an evaluation of the copy can be carried out using the "compute_fidelity_error" and the "replacement_capability" methods. The evaluation methods need data with which to perform the evaluation, so an unlabeled array-like parameter should be specified when calling them. If original test data is available, it can be used as a test for the copy evaluation. Otherwise, synthetic test data can be generated with the "generate_synthetic_data" method simply using another random seed. However, interpretation of the results will of course have a different meaning than with the original test data.

The package has been built in a modular manner so that it allows users or other developers to expand it further with their own sampling or evaluation functions. However, further analysis of the copy model can always be performed outside of the package simply by querying the copy model.

Spherical Balancer Sampling

The Spherical Balancer Sampling generates synthetic random samples within a sphere (or a spherical shell) centered at the origin of the feature space. It labels them using the original classifier, and then checks whether the number of samples for each class has achieved a certain predefined objective. If that objective hasn't been met for all the classes, it generates another batch of additional synthetic samples recursively until it succeeds, or until it reaches the allowed number of maximum iterations.

Therefore, this sampler has the advantage that it yields balanced synthetic datasets, which may be critical to train a successful copy for certain problems or combinations of original and copy models. Another advantage of this sampling function is that almost no information about the features must be provided. Only two single values for the inner and the outer radius to be sampled need to be specified. However this sampling only makes sense for standardized datasets where all the features have been rescaled and concentrate near the origin.

The Multidimensional Multiclass Gaussian Distribution Problem

In order to investigate the copying process and how it gets affected when different classes overlap or when the number of features is increased, a synthetic problem generator was implemented. In this problem each class is represented by a Gaussian distribution in a multidimensional feature space. The randomized distance between the centers of the Gaussian distributions of the classes, and the number of features of the dataset can be increased or decreased at will.

How is the problem generated?

1. A multidimensional Gaussian distribution is generated for the first class centered at the origin and with the standard deviation between the two scaling parameters.
2. For each additional class, a unit vector is generated in a random direction, and then it is multiplied by a random value between the two center parameters. This determines where the mean of the Gaussian distribution is located for that class.
3. A Gaussian distribution is generated with the mean at that point for the second class and then scaled to a random standard deviation between the two scaling parameters.
4. The same procedure is repeated to generate all the additional classes in the problem.

3.1.3 Functions and classes in the ML Classifier Copies package

presc.copies.examples.multiclass_gaussians

This function generates a multidimensional Gaussian dataset with multiple classes that can be used as a standardized synthetic classification problem with a specific chosen complexity. It first generates a multidimensional normal distribution centered at the origin with standard deviation one for class zero, and then adds an additional Gaussian distribution per class, centered at a random distance within a chosen interval, and with random standard deviation also within a chosen interval. The number of features/dimensions, the number of classes, and the number of data points per class are function parameters.

presc.copies.sampling.dynamical_range

This function returns the minimum, maximum, mean, and standard deviation of each feature in the dataset in a specific dictionary format that can be used as parameters across the different sampling functions.

presc.copies.sampling.grid_sampling

This sampling function generates synthetic samples with a regular grid-like distribution within the feature space described by the minimum and maximum values of the dynamic ranges of the features. It computes the grid spacing so that all features have the same number of different values.

presc.copies.sampling.uniform_sampling

This sampling function generates synthetic samples with a random uniform distribution within the feature space described by the minimum and maximum values of the dynamic ranges of the features.

presc.copies.sampling.normal_sampling

This sampling function generates synthetic samples with a normal distribution according to the feature space described by the provided mean and sigma values of the features. Features are assumed to be independent (that is, not correlated).

presc.copies.sampling.spherical_balancer_sampling

This sampling function generates synthetic samples with a spherical (shell) distribution between a minimum and a maximum radius values and then labels them using the original classifier. This function will attempt to obtain a balanced dataset by randomly generating samples but keeping only the necessary ones until it has the same number of samples specified for all classes, unless it stops earlier due to reaching the maximum number of iterations.

presc.copies.sampling.labeling

This function labels the samples from an unlabeled dataset which only contains the features, by querying an already trained classifier.

presc.copies.evaluations.empirical_fidelity_error

Computes the empirical fidelity error of a classifier copy, which quantifies the resemblance of the copy to the original classifier. The goal is that the copy mimics the original as closely as possible, including misclassifications. When the copy makes exactly the same predictions than the original classifier we have a perfect copy and this value is zero.

presc.copies.copying.ClassifierCopy

This class represents a classifier copy and its associated sampling method of choice. Each instance wraps the original ML classifier with a ML classifier copy and the specified sampling method to carry out the copy, including both the sampling function and any number of parameters.

presc.copies.copying.ClassifierCopy.copy_classifier

This ClassifierCopy class method allow to carry out the copy the original classifier by using data generated with the original model. It generates synthetic data using only basic information of the features (dynamic range, or mean and sigma), labels the data using the original model, and trains the copy model with this synthetic data.

presc.copies.copying.ClassifierCopy.generate_synthetic_data

This ClassifierCopy class method allows to generate additional synthetic data from the original model using the specified sampling method on instantiation and basic information of the features (dynamic range, or mean and sigma), including the train or test data for subsequent classifier copies.

presc.copies.copying.ClassifierCopy.compute_fidelity_error

This ClassifierCopy class method computes the empirical fidelity error of the classifier copy to evaluate the quality of the copy, quantifying the resemblance of the copy to the original classifier with respect to any dataset. This value is zero when the copy makes exactly the same predictions than the original classifier (including misclassifications).

3.1.4 Minimal example

Let's assume we have a labeled dataset in the PRESC format describing a certain classification problem, and that we use it at some point to train an original classifier model from it. This is in principle an independent process that has been carried out previously to the copying:

```
from sklearn.svm import SVC

# Instantiate and fit SVC classifier
original_classifier = SVC(kernel='linear')
original_classifier.fit(dataset.features, dataset.labels)
```

CODE 3.1: Fit original model.

Depending on the sampling that we use for the copy we may need to make some assumptions on the data distribution. We can introduce those assumptions manually in the "feature_parameters" options or, for convenience, if we have access to the original dataset, we can extract them in the appropriate format with the function "dynamical_range":

```
from presc.copies.sampling import dynamical_range

# Extract maximum and minimum values of the features
feature_range = dynamical_range(dataset.features)
```

CODE 3.2: Extract feature parameters from the distribution.

Assuming we have a trained classifier, and know the options to introduce to the sampling function, we can simply copy the model by instantiating the "ClassifierCopy" class with all the necessary parameters, including the type of model to use as a copy, and then running the "copy_classifier" method, such as in the code shown below:

```
from sklearn.tree import DecisionTreeClassifier
from presc.copies.copying import ClassifierCopy
from presc.copies.sampling import grid_sampling

# Instantiate and copy original model to a decision tree classifier
classifier_copy = DecisionTreeClassifier()
copy_grid = ClassifierCopy(original_classifier, classifier_copy,
                           grid_sampling, nsamples=200, random_state=42,
                           feature_parameters=feature_range)
copy_grid.copy_classifier()
```

CODE 3.3: Classifier copying

3.1.5 Coding standards

Coding standards are especially important to maintain and develop open-source projects due to their highly collaborative nature. It is common to have random collaborators and users with whom the maintainers of the project may never communicate, so everything must be extremely clear and self explanatory. Additionally, it

must be prevented that a meaningless difference in coding style between two versions of the code such a space or an empty line triggers an unnecessary review, so formatting must be homogenized as much as possible.

Code submitted to the PRESC project must adhere to certain style and standards, before it is considered for incorporation to the main branch of the code. Additionally, code must also be globally optimized to minimize dependencies, and reuse of already incorporated code must be prioritized before other solutions, as much as possible.

Code style and formatting

Maintaining a unified code style and formatting is very important for collaborative projects to ensure code clarity and readability, and facilitate its long term maintenance. This is especially pertinent in open-source software projects, which may have multiple contributors from different backgrounds.

Contributions to the PRESC code must adhere to the Python Black style [14] and Flake8 [15], which is in fact a wrapper for three lint enforcers: PyFlakes, pycodestyle, and Ned Batchelder's McCabe script.

Adherence to Black and Flake8 is checked locally in each developer's computer in an automated way before every new commit, so that new contributions that do not follow the code style cannot even be commited. Black reformats and changes the code so that it complies with the style. Flake8 carries out a check and points out at discrepancies, but the code needs to be modified manually.

Docstrings

In general all code should have proper documentation in the form of embedded inline comments, as well as higher level general documentation. However, a structured and specifically formatted comment block has the advantage of being easily parsed, so the text can be manipulated and combined together on a desired template in a straight-forward manner. This functionality is built into the Python language in the form of docstrings.

All code entities (functions, classes, etc) in the PRESC code must include their own docstring, which is a documentation string associated to the entity. Incorporating the basic documentation embedded in the code allows to automate a large part of the code documentation and generate it automatically after every version update, which helps prevent mismatches that otherwise occur frequently in other projects between the latest code version and outdated documentation that gets updated in a separate step.

The docstrings must contain:

1. A short description or summary in a single line.
2. If necessary, one or more paragraphs with a longer description or other explanations.
3. A list describing the parameters (in the case of functions) or attributes (in the case of classes), with their type, usage, etc.
4. A list describing the return values.

These docstrings are automatically converted to formatted documentation pages using Sphinx [16].

Testing

The development policy in PRESC also complies with unit testing by means of Pytest [17], so all entities (functions, classes, etc) submitted in the code must also have their associated unit test which checks its basic functionalities and expected outcomes.

This collection of tests is kept together with the code and is run in full automatically for every commit and pull request. Such testing is carried out using Continouous Integration (CI) linked to PRESC's public repository, so that there are at least some assurances that new code additions do not break previously established functionality. That the code passes the unit test collection is verified before the code is even considered for manual revision by other developers.

Usage examples

PRESC contribution guidelines also recommend to provide a Jupyter Notebook example demonstrating the usage with a real world dataset of any new code.

3.1.6 Contribution mechanics

Like most free software developed collaboratively, PRESC has a public repository with a distributed version control system (in this case, Git [18]) that allows to coordinate work among programmers [13]. The project's tasks and code discussions are also handled there through Github's issue tracker. As usual in free software projects, in order to contribute developers must follow a specific methodology and protocol:

1. Fork the original repository to work in their own version (this was indeed the case here, where the author forked the PRESC repository [19]).
2. Set up the local environment with the required dependencies.
3. Create a local branch to develop each specific feature or set of modifications, and add the desired code there.
4. Make sure the contributed code complies with the coding standards, such as style and formatting, and that it includes the required docstrings and that they have been updated as necessary.
5. Add or update unit tests to check that basic features of the code work as expected.
6. Add or update usage examples of the code in the form of Jupyter notebooks.
7. When the modifications are ready, a pull request explaining all the changes must be submitted against the desired branch of the main software repository. Normally, new features are not added to the main development branch of the original repository, but are incorporated into a separated specific branch instead until the next version release of the software. This is the case here, where each set of code modifications has been developed in a different branch of the author's fork of the PRESC repository, and then a pull request was submitted against the `model-copying` branch of the original repository, where the Classifier Copies Package is being developed.

8. Next, a number of automated processes are run in the main repository to ensure that the pull request complies with code style and that it passes all existing unit tests.

9. One or more human reviewers examines the proposed code and suggests changes if necessary.

10. Depending on the complexity of the code and the suggested changes, this can be an iterative process of back and forth discussions between the developer and the repository maintainers until the result is satisfactory.

11. The code is merged into the development branch of the feature and, eventually, when this is finished, the feature branch is merged into the main branch of the original repository and the code becomes part of the stable version of the software.

3.2 Master Thesis Project Funding

Although free software is often built by volunteer developers donating their time, it is sometimes necessary to obtain funding in order to ensure specific parts get developed. Public funds from the EU, individual countries, and local governments, or from organisations and NGOs are allocated to free software development, due to its contribution to the common good (see campaign "Public Money, Public Code" [20]). But many big and small private companies are also interested in investing their resources in free software development.

Academic research projects also require participation in competitive processes with proposal submission in order to secure the necessary funding. Therefore, funding for the development of this work was also sought by this master student as part of the Master Thesis efforts. A project proposal was submitted to the NLNet Foundation, and funding was successfully secured within the context of the Next Generation Internet Zero Discovery Fund [21].

3.3 Analysis of the copying process with a tunable problem

Once the copying pipeline has been implemented, it is interesting to explore the application of the technique, and to delve into exploring the copying process to better understand how the different parameters both of the original problem and of the copying process affect the quality of the copy.

Some interesting questions that emerge and that we would like to answer are: How is the quality of the copy affected as we increase the overlap between the classes in the original problem? And if we increase the dimensionality of the problem? What role does the size of the synthetic dataset play? What is the effect of different samplings? In particular, samplings that yield balanced synthetic datasets to train the copy model? And how does the choice of the original and copy model family affect the quality of the copy?

For this exploration we want to use a synthetic problem generator with tunable parameters so that we have a set of controlled experiments. As mentioned in section 3.1.2, an algorithm to generate multiclass multidimensional Gaussian distributions has also been implemented as part of the package ("multiclass_gaussians"). We have used this generator to prepare two collections of datasets for analysis so that the

different degrees and also flavours of complexity could be easily adjusted before performing the copies.

In the collection of multiclass separation experiments generated to **study the effect of the class overlap**, we have 5 features and 20 classes, where each class is a multidimensional Gaussian distribution with a standard deviation of 1 and with the center located in a random direction and distance from the center. For all problems the random distance to the center is taken from an interval that has a minimum value of 2, but the maximum distance takes these 11 different values: 2.0, 2.8, 3.6, 4.4, 5.2, 6.0, 6.8, 7.6, 8.4, 9.2, and 10.0 (see Code 3.4. As we increase the maximum distance the classes are more dispersed and become more easily separable (see Figure 3.2).

```
from presc.copies.examples import multiclass_gaussians

nproblems=11
original_datasets = [None]*nproblems
for problem in range(nproblems):
    maximum_distance = 2+(8*problem/(nproblems-1))
    original_datasets[problem] = multiclass_gaussians(nsamples=20000,
                                    nfeatures=5, nclasses=20, center_low=2,
                                    center_high=maximum_distance,
                                    scale_low=1, scale_high=1)
```

CODE 3.4: Generate class overlap problem series.

Figure 3.2 shows data obtained with the Multidimensional Multiclass Gaussian Distribution Problem Generator using as options 5 features, 20 classes with standard deviation 1, and an interval for the centers of the additional distributions between 2 and 10, 2 and 6, and 2 and 2, respectively. As can be seen, classes are reasonably separated in the feature projections for the largest maximum distance but reducing this interval progressively forces the centers of the Gaussian distributions to be closer to the origin and, hence, they become more overlapped.

In the collection of multiclass separation experiments generated to **study the effect of increasing the dimensionality of the problem**, the number of features take the 11 different values 5, 15, 25, 35, 45, 55, 65, 75, 85, 95, and 105. For all problems we have 20 classes and each class has a standard deviation of 1 and has the center located in a random direction and distance from the center with a minimum of 2 and a maximum of 10 (see Code 3.5). Since the maximum distance to the center of the feature space is constant, as we increase the dimensions there are contributions from more features and in average each feature contributes less. Hence, the classes are more overlapped in the individual feature projections as dimensions increase (see Figure 3.3).

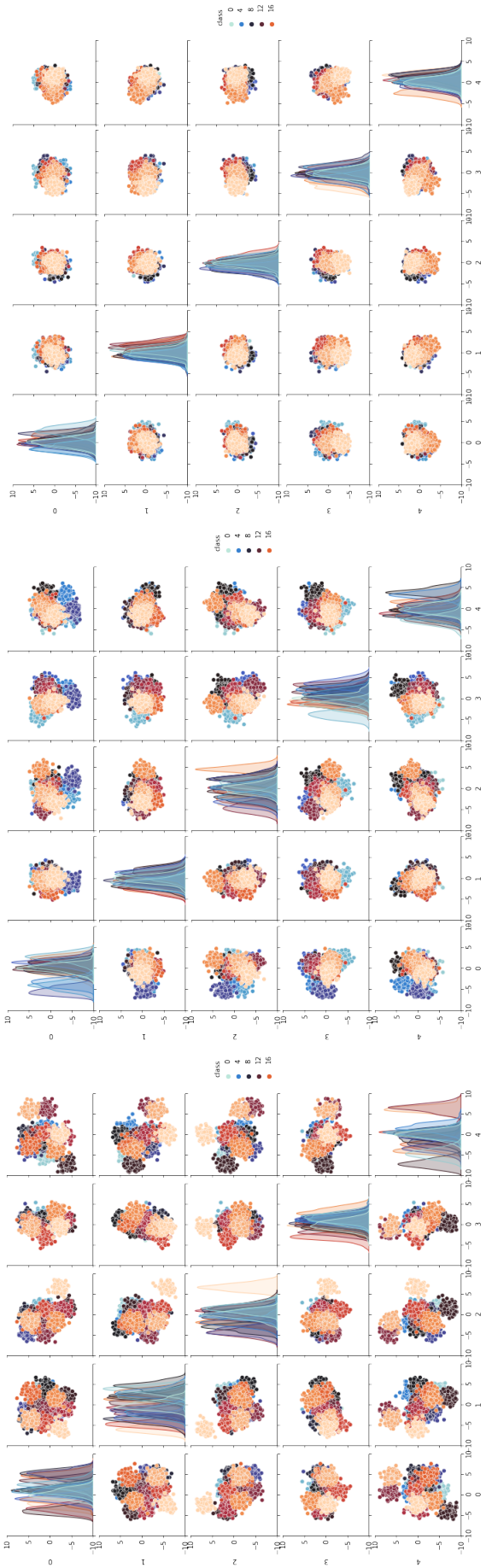

FIGURE 3.2: Data obtained with the Multidimensional Multiclass Gaussian Distribution Problem Generator using as options 5 features, 20 classes with standard deviation 1, and an interval for the centers of the additional distributions between 2 and 10, 2 and 6, and 2 and 2, respectively. As can be seen, classes are reasonably separated in the feature projections for the largest maximum distance but get progressively overlapped for the smallest one.

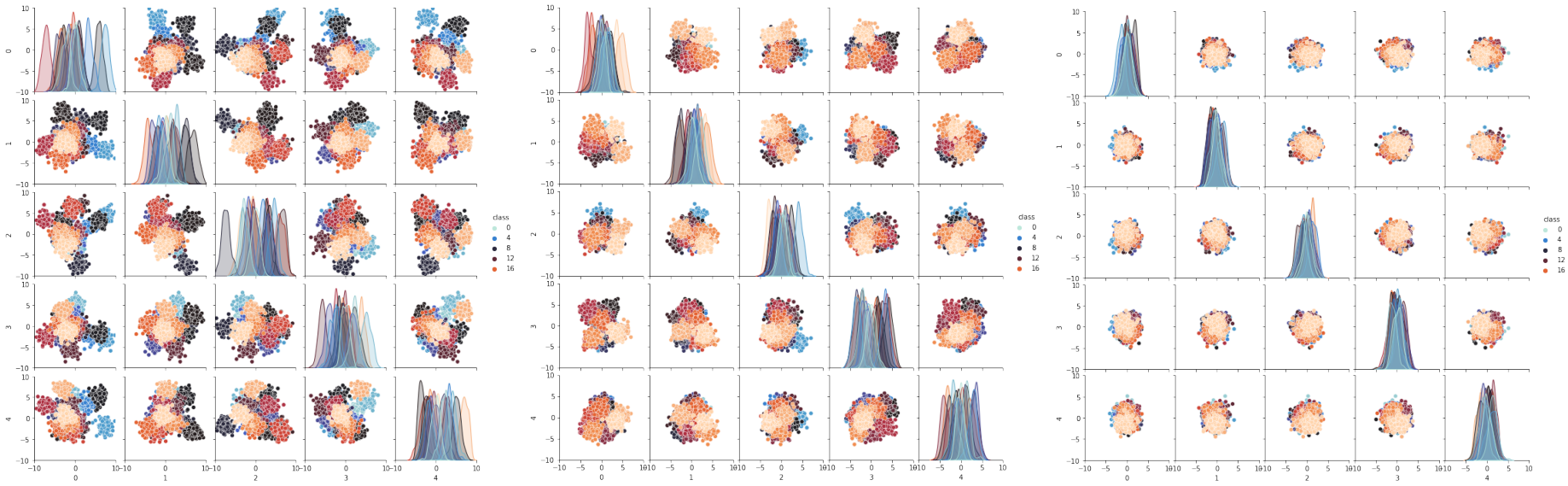

FIGURE 3.3: Data obtained with the Multidimensional Multiclass Gaussian Distribution Problem Generator using as options 5, 12 and 50 features, respectively, although in all cases only the first five have been represented. The rest of the problem parameters are 20 classes with standard deviation 1, and an interval for the centers of the additional distributions between 2 and 10. As can be seen, classes are here reasonably separated in the feature projections for lower dimensions but get progressively clumped together as the dimensions increase.

```
from presc.copies.examples import multiclass_gaussians

nproblems=11
original_datasets = [None]*nproblems
for problem in range(nproblems):
    nfeatures = 5+int(100*(problem/(nproblems-1)))
    original_datasets[problem] = multiclass_gaussians(nsamples=20000,
                                                      nfeatures=nfeatures, nclasses=20,
                                                      center_low=2, center_high=10,
                                                      scale_low=1, scale_high=1)
```

CODE 3.5: Generate dimensional problem series.

With each one of these 22 problems of 20.000 samples we have performed a large variety of classifier copies to explore the effect of different copy parameters:

Model combinations We have chosen three combinations of original/copy model pairs: from linear SVC to linear SVC, to study the case when the destination model is from the same family as the original model, and from a linear SVC to a decision tree classifier and vice versa, to study the case when the two models are from very different families and also whether this process has a certain symmetry. We have used the scikit-learn models "sklearn.svm.SVC" and "sklearn.tree.DecisionTreeClassifier" with one set of default parameters for all the models from the Linear SVC family and another set of parameters for the models from the Decision Tree Classifier family (see Code 3.6 and 3.7).

```
IN: original_SVC_classifiers[0].get_params()
OUT: {'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovo',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'linear',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

CODE 3.6: Parameters of the Linear SVC models.

```
IN: original_tree_classifiers[0].get_params()
OUT: {'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

CODE 3.7: Parameters of the Decision Tree Classifier models.

Sampling function We have made the classifier copies using two different methods to generate the training synthetic datasets: the normal sampler and the spherical balancer sampler. The normal sampler assumes a Gaussian distribution for each feature with the mean and standard deviation specified in the feature descriptors, not correlated among them, and generates values from a multi-dimensional Gaussian distribution. The spherical balancer sampler generates random samples within a sphere centered at the origin of the feature space (see 3.1.2 for more details). The main difference is that the latter generates a balanced training synthetic dataset.

Number of samples For the copies carried out with the normal sampler we have generated 1.000, 10.000 and 100.000 synthetic samples, and for the spherical balancer sampler we have generated 1.000, 2.000, and 3.000 samples. Although there is no theoretical hard limit on the number of samples that can be generated for the copy, there may be a practical limitation depending on the efficiency of the sampler. The spherical balancer sampler was more computationally expensive than the normal sampler and is the reason why we generated a lower volume of samples for the latter. Therefore, it is interesting to analyse the smallest synthetic dataset size that allows to obtain a good quality copy. We expect this to also depend on the model pair.

Therefore, we have prepared the quantity of $11 \cdot 2 \cdot 2 = 44$ original models and carried out a total of $(11 + 11) \cdot 3 \cdot 2 \cdot 3 = 396$ classifier copies.

We have analysed the accuracy of the original models as well as the accuracy of all these copies, to check for the performance of both and see how it changes. We also also analyze the Empirical Fidelity Error (see Equation 2.5), in order to quantify the goodness of the copies. The copy will be of higher quality when it mimics the original model exactly, including its misclassifications. When performing a copy, we do not aim for an improvement of the original model performance, but to obtain the exact same behavior.

Another very useful metric, that we can call Replacement Capability, is simply to take the ratio between the copy model accuracy with respect to the original model accuracy. This allows to evaluate the performance of the copy with respect to the original model, which can have a high value even if the performance of the original model is not very good. And it can even take values larger than one if the copy model is better than the original.

Chapter 4

Experimental Results and Discussion

All the code corresponding to the calculation of the results shown in this chapter can be found in the two Jupyter notebooks in the author's repository [19].

4.1 Original Models

Each of the 22 generated problems was fitted with two original models: one from the Linear SVC family and another from the Decision Tree Classifier family. More details of the model parametrization are shown in Codes 3.6 and 3.7. Looking at Figures 3.2 and 3.3 it is clear that for lower maximum distances and higher dimensions the different classes get more overlapped in the feature projections. However, the accuracy of the original Linear SVC models is quite high in all cases regardless of the dimensionality.

Taking a look at the accuracy of the original models it is clear that this has a larger effect in the maximum distance problems than in the dimensional ones (see Figures 4.3 and 4.5). In the former, accuracy is reduced by a 53% for the Linear SVC models and by a 68% for the Decision Tree Classifier models when reducing the maximum distance from 10 to 2, due to the classes becoming more overlapped and harder to separate, while in the latter this reduction is an almost unnoticeable 1,5% for the Linear SVC models and a 41% for the Decision Tree Models when increasing the dimensions from 5 to 105. Thus, it appears that, although the information is scattered through more dimensions and there is a higher overlap within individual feature projections, this may be compensated on the Linear SVC models, which are much better suited to handle the increase of dimensions, by the increase of features.

Apart from the overall variations in accuracy of the models, we have also studied the performance of the models in the 11 maximum distance problems with each class by plotting their precision as a function of the distance to the origin (see Figure 4.1). It can be clearly seen that the original classifier models for each problem yield a poorer performance for classes that are closer to the origin, that is because the density and, hence, the class overlap, increases dramatically. Model performance with respect to classes closer to the origin also improves (lighter colors) for the problems where the distance interval is larger, due to the lower densities. It is also clear that for classes farther than a certain minimum threshold around 7, they become completely separable reaching a precision of one, and the performance cannot improve anymore.

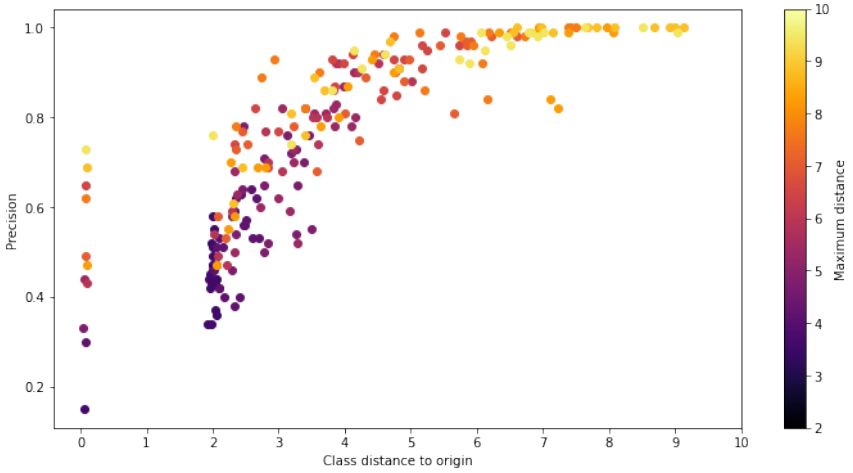

FIGURE 4.1: Precision of the prediction for each class as a function of the distance to the origin of that particular class (only for the Linear SVC models). Colors indicate the belonging of each class to one of the particular 11 problems all with 5 features and 20 classes each, but with different distance intervals.

4.2 Copies as a function of class overlap (distance between classes)

As can be seen in Figure 4.2, the quality of the classifier copies does not change as a function of the distance interval, except for the case of the Linear SVC copies from a Decision Tree Classifier, which are better when the classes are less overlapped. This is also the case for the copies from a Linear SVC to a Decision Tree Classifier but only when using the spherical balancing sampler.

In the original problem, classes become easier or harder to separate by the classifier when changing the interval, but the classifier copy does not see the original problem, it only sees the original classifier, which may have similar characteristics (and complexity) in most cases. Hence, the copying problem does not become more complex and shows a similar Empirical Fidelity Error.

On the other hand, the performance of the copies increases as the maximum distance between the classes increase and they are less overlapped, until a certain separation where there is a plateau and performance does not improve anymore (see Figure 4.3).

4.3 Copies as a function of dimensionality

The behavior of the goodness of the copies with changing dimensionality is complex and highly dependent on the original and copy models, as well as the sampler used to generate the synthetic training data.

As can be seen in Figure 4.4, where the empirical fidelity error of the copies is depicted, the quality of the copies in the problems generated here generally decreases with increasing the number of features. Especially when copying crossed model families, where this phenomenon happens already at low dimensionality and, in the particular case from Linear SVC to Decision Tree Classifier, quite abruptly.

However, when copying models within the same model family, a step-wise behaviour was observed when increasing the dimensionality: very good copies were

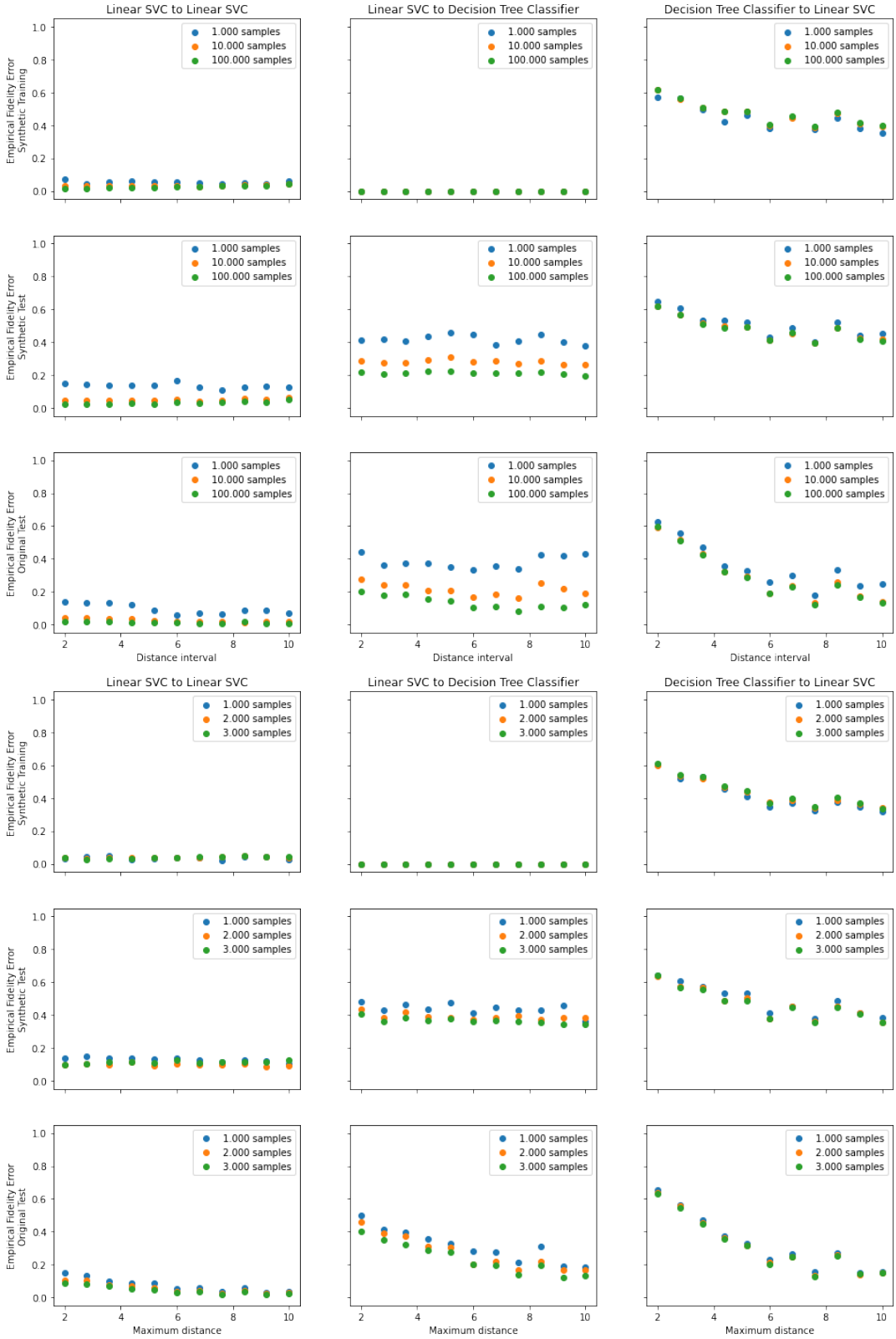

FIGURE 4.2: Empirical Fidelity Error for the 198 copies of the maximum distance problems (5 features, 20 classes, and class distance from a minimum fixed at 2 and a maximum ranging from 2 to 10) performed with normal sampling (rows 1-3), and with spherical balancer sampling (rows 4-6).

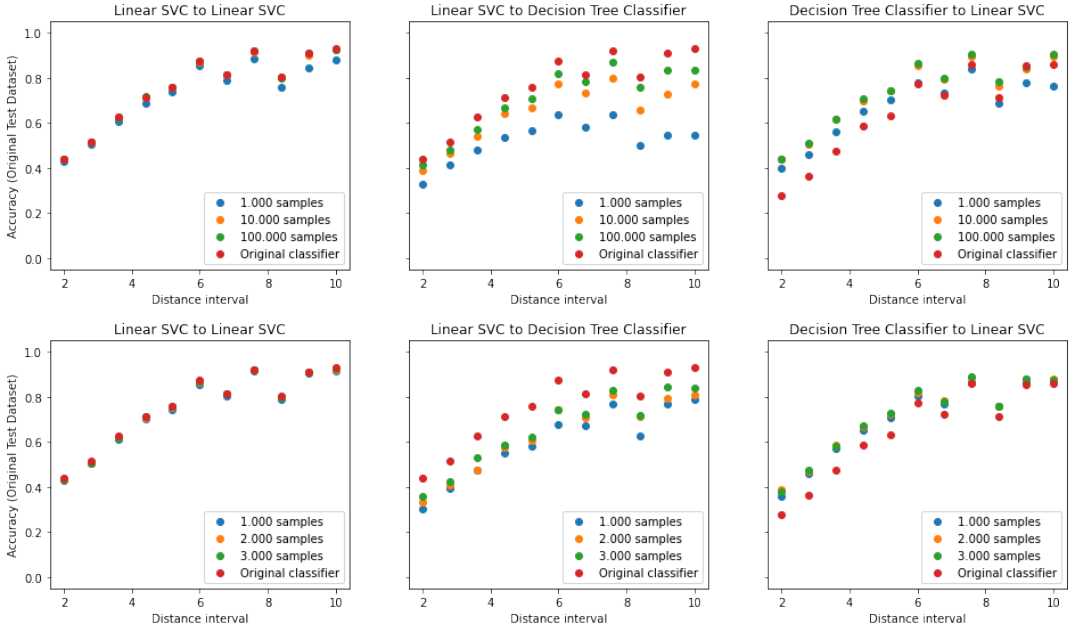

FIGURE 4.3: Accuracy for the 22 original models of the maximum distance problems (5 features, 20 classes, and class distance from a minimum fixed at 2 and a maximum ranging from 2 to 10) and also their 99 copies performed with normal sampling (top row), and 99 copies performed with the spherical balancer sampling (bottom row).

obtained up to a certain point and then for higher dimensionalities the copies became of bad quality. The threshold for this behavior seems to move to higher dimensions as the number of generated synthetic samples is increased, and the general quality of the copies in higher dimensions also increases.

As shown in Figure 4.5, the performance of the copies follows the same behavior as their quality: a general decrease for higher dimensions, more or less abrupt depending on the original/copy model pair. And not as relevant when copying to the same family as the original model. There, even for higher dimensions, the copy becomes reasonably good if we increase the number of samples.

4.4 Copies as a function of the number of samples

As expected, the quality of the copies and their accuracy generally improve with the number of samples. However, it is still interesting to explore what is the minimum number of samples that allows to have a good copy. And the answer is highly dependent on the problem and sampler that has been used.

For instance, in the maximum distance problem, the copies from Linear SVC to a Linear SVC or a Decision Tree Classifier with normal sampler experience a significant increase when going from 1.000 to 10.000 samples, but from 10.000 to 100.000 the improvement is much smaller. And for the case of the Decision Tree Classifier to a Linear SVC there is almost no change whatsoever in the quality of the copy between any of those training dataset volumes.

The behavior of the copy with the number of samples also gives a sense of the sampling efficiency, by seeing how many synthetic datapoints are needed to recover the performance of the original classifier, compared to the original dataset size.

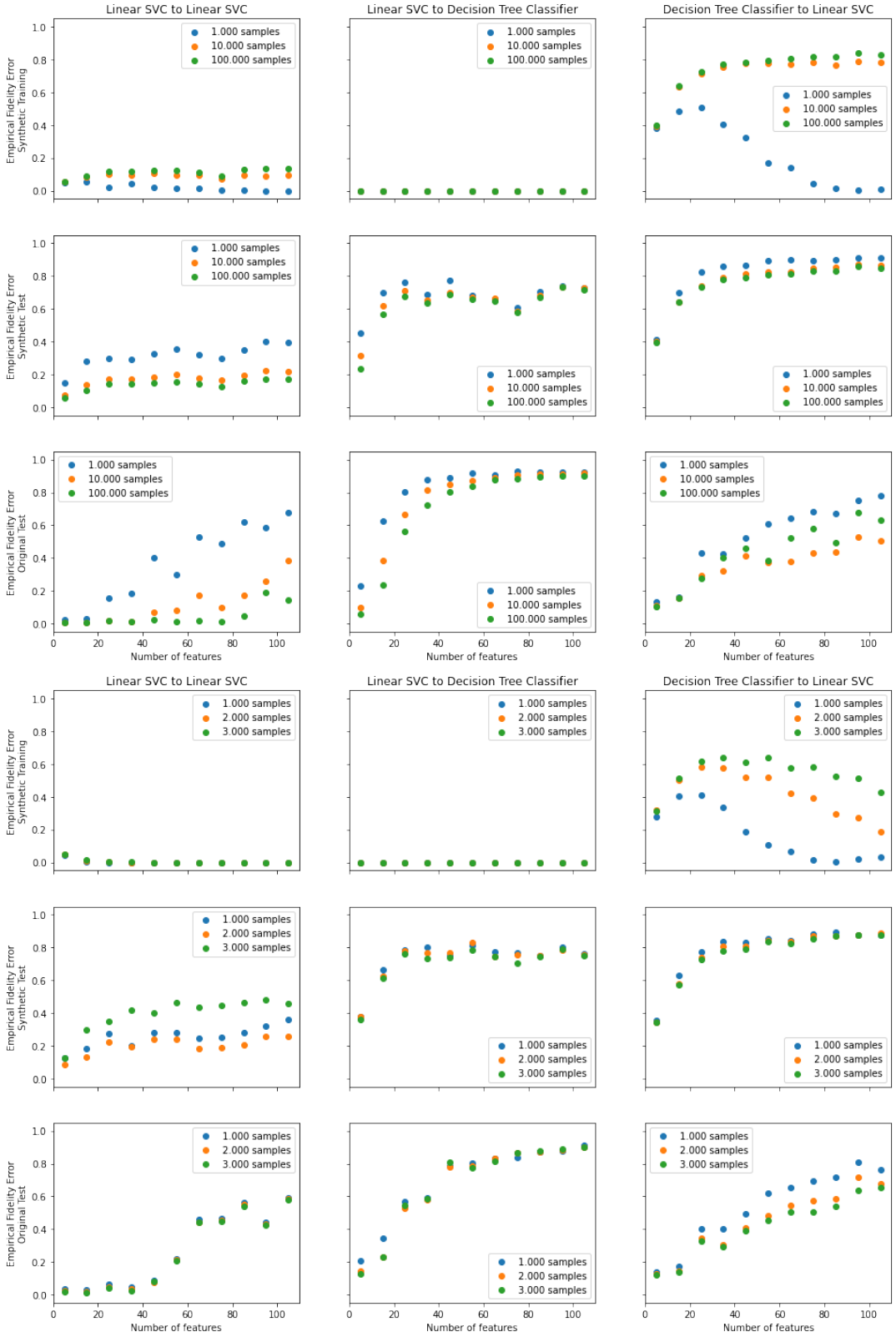

FIGURE 4.4: Empirical Fidelity Error for the 198 copies of the dimensional problems (5-105 features, 20 classes, and fixed class distance interval from 2 to 10) performed with normal sampling (rows 1-3), and with spherical balancer sampling (rows 4-6).

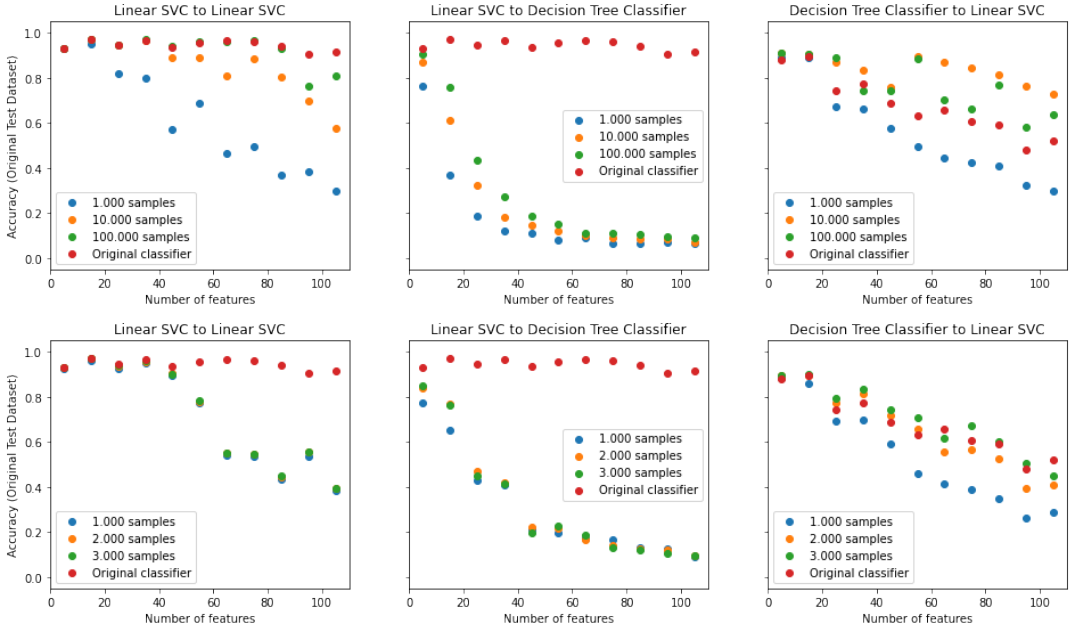

FIGURE 4.5: Accuracy for the 22 original models of the dimensional problems (5-105 features, 20 classes, and fixed class distance interval from 2 to 10) and also their 99 copies performed with normal sampling (top row), and 99 copies performed with spherical balancer sampling (bottom row).

4.5 Copies as a function of model combinations

Different families of classifiers represent class boundaries in characteristic ways. On one hand, Linear SVCs find the optimal hyperplanes of the feature space that separate the samples and, on the other, Decision Tree Classifiers create step-wise boundaries that are parallel to the axes (see Figure 4.6).

The copying problem is defined by the boundaries of the original model, regardless of the "natural" boundaries of the original problem: our goal is to mimic the model, not the original data. Hence, increasing the difficulty of the problem, not always results in a more difficult copying problem if at the end we have to copy an original model from the same family and with a similar complexity. When fewer samples are tested, it is equivalent to having a low resolution of the boundaries and, hence, a slanted and a step-wise boundary may not be distinguishable from one another, and the copy may show to have a better empirical fidelity error than when more samples are used and the boundary differences are inescapable.

Here, copies from original Linear SVC classifiers onto the same model family were carried out, which serve as the copying problem reference. Additionally, crossed model family copies from Linear SVC classifiers onto Decision Tree Classifiers and vice versa were also carried out.

When copying from an original Linear SVC model onto a model of the same family, the copies are always very good for the maximum distance problems and they show a low empirical fidelity error. For the dimensional problems the copies are good up to a certain threshold and then worsen. This threshold shifts to higher dimensions when increasing the number of samples, and its value also depends on the sampler. As expected, the accuracy of the copies is very similar to the original models when the copy is good, so it increases with the maximum distance for the

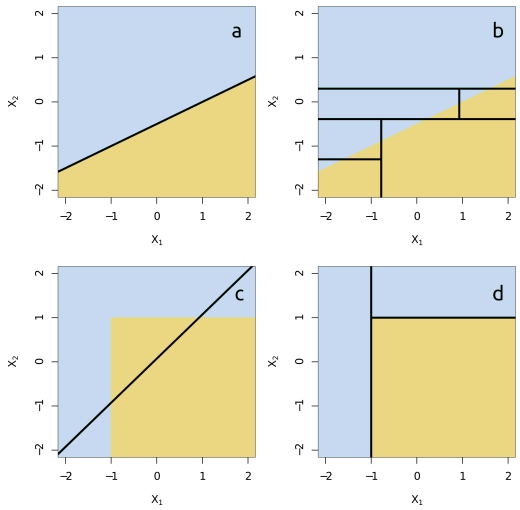

FIGURE 4.6: Difficulties when copying between model families that represent boundaries differently. Top row: If the original decision boundary is linear, a linear copy boundary (left) will outperform a decision tree copy that performs splits parallel to the axes (right). Bottom Row: A linear copy model is unable to capture the original non-linear decision boundary to make a good copy (left), whereas a decision tree copy is successful (right). Figure from Reference [22].

class overlapping problems, and it decreases with the number of features for the dimensional problems.

When the original Linear SVC models are copied into a Decision Tree Classifier model, the quality of the copies is not very affected by the maximum distance between the classes, but shows a rather abrupt decay with the number of dimensions. This may simply be an indication that the Decision Tree Classifier is not handling very well having to deal with an increasingly large number of dimensions. In fact, this seems to be confirmed by the fact that the accuracy of the **original** Decision Tree Classifier models also shows a continuous decrease as a function of the dimension.

The Linear SVC copies of the Decision Tree Classifiers are increasingly bad with the number of dimensions, but their accuracy are reasonably close to the accuracy of the original models. A noteworthy phenomenon that can be observed in Figures 4.2, 4.3, 4.4, and 4.5 is the fact that both for the maximum distance and the dimensional problems we obtain model copies with better accuracies than the original models.

One reason for this can be that the original problem is better behaved with linear boundaries, but that they have been approximated by a step-wise original model (in this case, a model from the Decision Tree Classifier family). When copying the original model to a linear model (in this case, a model from the Linear SVC family), an interpolation of the step-wise boundary is in effect taking place, which approximates the boundaries of the original data much better. That is why we do not succeed in obtaining an exact copy but we obtain a copy with a better performance than the original model, even when we are generating more and more synthetic data from the original model.

To be clear, this was not our goal here, however, it helps us understand and diagnose the inadequacy of the original model to the problem at hand, which is another application of the Machine Learning Classifier Copying technique.

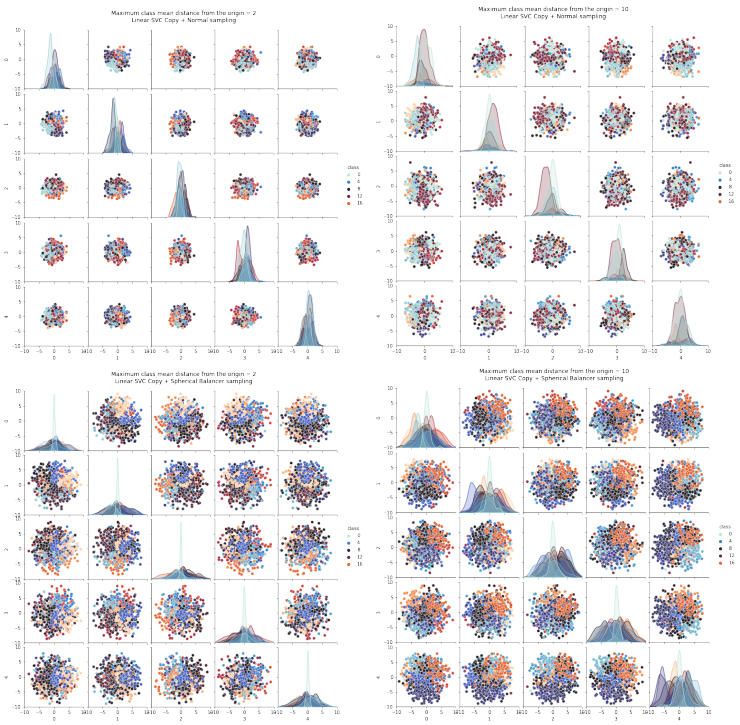

FIGURE 4.7: Synthetic training samples generated by the normal and spherical balancer samplers (top and bottom rows, respectively) for the maximum distance problem of 2 and 10 (left and right columns, respectively) when copying from a Linear SVC original model.

4.6 Copies as a function of the sampling strategy

4.6.1 Differences between the sampling strategies

In order to understand the differences between the sampling strategies of the normal and spherical balancer samplers we examine the synthetic training datasets they generated. Figures 4.7 and 4.8 show the the distribution of the synthetic training samples generated by the normal and spherical balancing samplers for the edge cases of the maximum distance and the dimensional problems, respectively.

For the maximum distance problem it can be seen that the normal sampling (top row) probes a smaller volume for the maximum distance of 2 than 10, because it uses the feature information (means and standard deviations) to sample the feature space, so it takes into account that in the first case the original samples are nearer the origin than in the second case. Conversely, the spherical balancing sampler (bottom row) always samples the same volume of the feature space because it does not make any assumptions for the individual features. However, it has a larger concentration of samples near the center for the maximum distance of 2 and a more homogeneous distribution for the maximum distance of 10. This is caused by the fact that the classes themselves are more concentrated near the origin in the first case so the sampler is forced to generate more samples near the origin in order to get a balanced dataset.

For the dimensionality problem it can be seen that the normal sampling (top row) probes a larger fraction of each feature for the smaller number of dimensions than for the large ones, again because the distributions of the classes are themselves more concentrated near the origin for higher dimensions, and the normal sampler uses that information. The spherical balancing sampler (bottom row), however, probes in

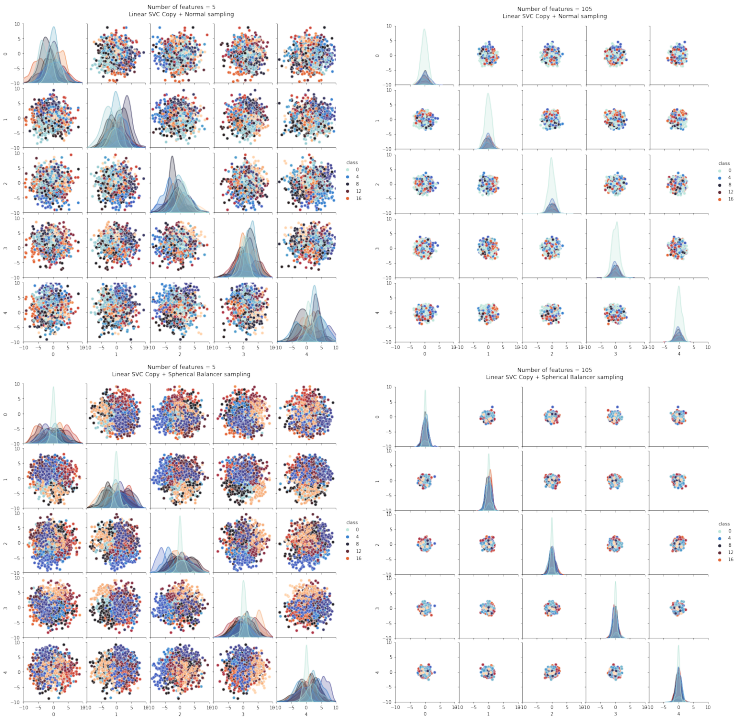

FIGURE 4.8: Synthetic training samples generated by the normal and spherical balancer samplers (top and bottom rows, respectively) for the dimensional problem with 5 and 105 features (left and right, respectively) when copying from a Linear SVC original model. However, only the first 5 features have been represented in the latter case.

this case a larger fraction of the feature projections for small number of dimensions than for large number of dimensions, because the region that this sampler probes is defined as a fixed minimum and maximum radius from the origin and, as dimensions increase, each feature contributes a smaller fraction of that distance, hence, the samples are closer to the origin.

Another practical difference between the normal and the spherical balancing samplers implemented here is that the latter is more computationally expensive than the first. This is why we were able to generate training datasets with a much larger volume of synthetic data for the normal sampler. It is not a determinant issue because the reason for that is that this implementation achieves balanced synthetic datasets by brute force. It is an easy to understand algorithm but not a very smart one, and there is clearly room for significant improvement in this aspect in future work.

4.6.2 Differences between the copies with different samplers

The comparison between the two samplers can only be sensibly carried out by looking at the copies carried out with the same number of samples (1.000) (which correspond to the blue dots in all the Accuracy and Empirical Fidelity Error Figures).

For the maximum distance problem, the empirical fidelity error of the copies performed with both samplers are almost the same, perhaps only slightly better for the spherical balancing sampler. There is an exception to that for the copies from a Linear SVC to a Decision Tree Classifier, where the spherical sampler yields sensibly

better quality copies for larger maximum distances. In all cases the spherical balancing sampler yields copies with a better accuracy than the normal sampler, a few of them significantly better, which means that the spherical balancing sampler can achieve a better result more efficiently with less samples than the normal sampler.

For the dimensional problem, the differential behavior of the two samplers is strongly linked to the choice of original/copy model pair. When copying to the same model family (from Linear SVC to Linear SVC), the empirical fidelity error for the normal sampler becomes steadily poor very fast as dimensions are increased, but with the spherical balancing sampler the quality stays very good until a certain dimension (45 features) which then drops abruptly to similar levels as the normal balancer. When copying from Linear SVC to Decision Tree Classifier models, with the normal sampler the quality of the copies sinks abruptly already at lower dimensions as we increase them, while the copies with the spherical balancing sampler show a more progressive decline instead. Lastly, the Decision Tree Classifier to Linear SVC copies do not show any difference at all between the two samplers. The accuracy follows a very similar behavior as the empirical fidelity error: for copies from Linear SVC to the same type of model, a much better performance can be observed for the copies with spherical balancing sampler up to 45 dimensions, and then an abrupt worsening of the performance to the same levels as the copies with normal sampler, for copies from Linear SVC to Decision Tree Classifiers there is a fast and much more abrupt worsening of the performance for the copies with the normal sampler, and for the Decision Tree Classifier to Linear SVC copies there is no difference.

In general, the copies obtained with the spherical balancing sampler have smaller empirical fidelity errors than the ones obtained with the normal sampler. And they quickly improve with the original classifier as the problem simplifies. The performance of the copies carried out with the spherical balancing sampler also follows the performance of their original classifiers much more closely than the copies carried out with the normal sampler.

However, the difference between the quality and performance of the copies becomes smaller as more samples are generated. Therefore, it is a matter of sampling efficiency and having a more representative synthetic dataset already with less samples, which helps obtain a better copy. The imbalance between the most populated and the least populated classes of the synthetic training datasets can be very large and the fact that the spherical sampler yields a balanced training dataset has shown here to be key for achieving better results.

Chapter 5

Conclusions

Machine Learning Classifier Copying is a powerful technique that allows to replicate the decision boundary of a classifier model without the need of knowing its architecture or accessing its original data.

The implementation that has been provided here as a package of Mozilla's free software PRESC toolkit has proven to be practical for large-scale analyses, extendable with more advanced options as the discipline matures, and is now readily accessible to researchers and data science practitioners.

Experiments show that plenty of information can be extracted regarding the original model with an approach of systematic series of model copies, and confirm the potential of this technique for original model evaluation.

Results show that:

- More difficult problems with classes that are less separable does not necessarily result in worse copies if the original model is from the same family and has the same complexity.
- Copies with the spherical balancing sampler generally perform better, specially when generating smaller volumes of synthetic training samples.
- Generating a balanced synthetic training dataset can be key in order to obtain a good copy with fewer data.
- Differences between model idiosyncrasies play a large role on the quality of the copy. Different families of models represent boundaries in different ways and that may give rise to unexpected behaviors.
- Although it is not our goal when doing the copies, sometimes they perform better than the original model, and then it is an indication that the original model is not the most adequate for the problem. This can be used as a diagnostic tool that can help evaluate the original model.
- Copies tend to improve when the synthetic training datasets have larger volumes but only up to a point, then the improvement is negligible. This sweet spot depends on the specifics of the problem, the original/copy model pair, and the chosen sampler.

Similar systematic experiments as the ones performed here need to be carried out in future work with real datasets. Although convenient for exploring the behaviour of the copying process in a controlled manner, all data used in this work has been simulated, which has clear limitations. More work is needed in order to see how well these conclusions apply in general.

More work is needed to develop smarter balancing samplers that allow to generate balanced synthetic training data more efficiently. One approach is to use seeded

sampling, which can sample the space surrounding known samples of the scarcer classes. Developing samplers that focus on sampling the boundaries would also be very useful.

Further systematic exploration is also quite necessary to determine the minimum number of samples required to obtain a good copy, and the influence of the differences on the type of boundary of the original and copy models in the goodness of the copy, in order to develop strategies to overcome this challenge.

It would also be quite interesting to research how the mismatch between the complexity of the original and copy models affects the quality of the copy. This would allow to quantify in a standardized manner the complexity of unknown original classifiers, or perhaps to use this knowledge to obtain simpler versions of complex models.

Appendix A

Source Code

Mozilla's public official repository for the PRESC (Performance and Robustness Evaluation for Statistical Classifiers) project is available at Github and can be accessed through the following link:

https://github.com/mozilla/PRESC/

In particular, the Machine Learning Classifier Copies package is currently being developed in a feature branch of Mozilla's repository ("model-copying"), where a version of the PRESC toolkit which includes all the contributed code that has already been accepted and passed the code reviews can be found. This branch will be merged with the main branch of the program once the package is finished and the code has passed all requirements and tests to be production ready:

https://github.com/mozilla/PRESC/tree/model-copying

As is customary in free software development, the code regarding the implementation of this Master Thesis has been first developed in a PRESC public fork created by the author, where the most updated version of the Machine Learning Classifier Copies package, the Master Thesis report, and two extensive Jupyter notebooks with the code for the calculations and figures shown in the exploratory chapter can currently be found:

https://github.com/alberginia/PRESC/tree/master_thesis

The PRESC package is built using the standard Python libraries for scientific computing and machine learning, including Pandas, Numpy and Scikit-learn. All of the computations described in this work have been executed using Scikit-learn version 0.23.1.

Bibliography

[1] John P. A. Ioannidis. "Why Most Published Research Findings Are False". In: *PLOS Medicine* 2.8 (Aug. 2005), null. DOI: 10.1371/journal.pmed.0020124. URL: https://doi.org/10.1371/journal.pmed.0020124.

[2] Thomas Pfeiffer and Robert Hoffmann. "Large-Scale Assessment of the Effect of Popularity on the Reliability of Research". In: *PLOS ONE* 4.6 (June 2009), pp. 1–4. DOI: 10.1371/journal.pone.0005996. URL: https://doi.org/10.1371/journal.pone.0005996.

[3] S N Goodman and R Royall. "Evidence and scientific research." In: *American Journal of Public Health* 78.12 (1988). PMID: 3189634, pp. 1568–1574. DOI: 10.2105/AJPH.78.12.1568. eprint: https://doi.org/10.2105/AJPH.78.12.1568. URL: https://doi.org/10.2105/AJPH.78.12.1568.

[4] Matt Crane. "Questionable Answers in Question Answering Research: Reproducibility and Variability of Published Results". In: *Transactions of the Association for Computational Linguistics* 6 (2018), pp. 241–252. DOI: 10.1162/tacl_a_00018. URL: https://aclanthology.org/Q18-1018.

[5] I. Unceta, J. Nin, and O. Pujol. "Copying Machine Learning Classifiers". In: *IEEE Access* 8 (2020), pp. 160268–160284. DOI: 10.1109/ACCESS.2020.3020638. URL: https://ieeexplore.ieee.org/document/9181566.

[6] Irene Unceta, Jordi Nin, and Oriol Pujol. "From Batch to Online Learning Using Copies". In: *Artificial Intelligence Research and Development - Proceedings of the 22nd International Conference of the Catalan Association for Artificial Intelligence, CCIA 2019, Mallorca, Spain, 23-25 October 2019*. Ed. by Jordi Sabater-Mir et al. Vol. 319. Frontiers in Artificial Intelligence and Applications. IOS Press, 2019, pp. 125–134. DOI: 10.3233/FAIA190115. URL: https://doi.org/10.3233/FAIA190115.

[7] Irene Unceta et al. "Sampling Unknown Decision Functions to Build Classifier Copies". In: *Modeling Decisions for Artificial Intelligence*. Ed. by Vicenç Torra et al. Cham: Springer International Publishing, 2020, pp. 192–204. ISBN: 978-3-030-57524-3. URL: https://link.springer.com/chapter/10.1007/978-3-030-57524-3_16.

[8] Irene Unceta, Jordi Nin, and Oriol Pujol. "Risk mitigation in algorithmic accountability: The role of machine learning copies". In: *PLOS ONE* 15.11 (Nov. 2020), pp. 1–26. DOI: 10.1371/journal.pone.0241286. URL: https://doi.org/10.1371/journal.pone.0241286.

[9] Irene Unceta, Jordi Nin, and Oriol Pujol. "Environmental Adaptation and Differential Replication in Machine Learning". In: *Entropy* 22.10 (Oct. 2020). ISSN: 1099-4300. DOI: 10.3390/e22101122. URL: https://www.mdpi.com/1099-4300/22/10/1122.

[10] Becca Ricks et al. *Creating Trustworthy AI: a Mozilla white paper on challenges and opportunities in the AI era*. 2020. URL: https://foundation.mozilla.org/en/insights/trustworthy-ai-whitepaper/.

[11] Nicolas Papernot et al. "Practical Black-Box Attacks against Machine Learning". In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ASIA CCS '17. Abu Dhabi, United Arab Emirates: Association for Computing Machinery, 2017, pp. 506–519. ISBN: 9781450349444. DOI: 10.1145/3052973.3053009. URL: https://doi.org/10.1145/3052973.3053009.

[12] Mozilla Foundation. *PRESC's Documentation*. https://mozilla.github.io/PRESC/index.html. Accessed: 2021-08-21.

[13] Mozilla Foundation. *PRESC: Performance and Robustness Evaluation for Statistical Classifiers - Github Repository*. https://github.com/mozilla/PRESC. Accessed: 2021-08-21.

[14] Łukasz Langa and others. *Black - The uncompromising code formatter*. Accessed: 2021-08-21. URL: https://black.readthedocs.io/en/stable/.

[15] Ian Cordasco and Tarek Ziade. *Flake8: Your Tool For Style Guide Enforcement*. Accessed: 2021-08-21. URL: https://flake8.pycqa.org/en/latest/.

[16] Georg Brandl, Armin Ronacher, et al. *Sphinx - Python Documentation Generator*. Accessed: 2021-08-21. URL: https://www.sphinx-doc.org/.

[17] Holger Krekel and others. *pytest: helps you write better programs*. Accessed: 2021-08-21. URL: https://docs.pytest.org/.

[18] Linus Torvalds, Junio Hamano, et al. *git - everything is local*. Accessed: 2021-08-21. URL: https://git-scm.com/.

[19] Muriel Rovira-Esteva. *PRESC: Performance and Robustness Evaluation for Statistical Classifiers - Github Repository*. Accessed: 2021-09-02. URL: https://github.com/alberginia/PRESC/tree/master_thesis.

[20] Free Software Foundation Europe. *"Public Money, Public Code" Campaign*. Accessed: 2021-08-21. URL: https://publiccode.eu/.

[21] NLNet Foundation. *PRESC Classifier Copies Package - Implementing Machine Learning Copies as a Means for Black Box Model Evaluation and Remediation*. Accessed: 2021-08-21. URL: https://nlnet.nl/project/PRESC/.

[22] Gareth James et al. *An Introduction to Statistical Learning with Applications in R*. Springer, 2021. URL: https://www.statlearning.com/.