UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

FELIPE FÜHR DOS REIS

Measuring Drift Impact: A Customizable Synthetic Data Generator

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Engineering

Advisor: Prof. Dr. Anderson Rocha Tavares

Porto Alegre
September 2023

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos André Bulhões Mendes
Vice-Reitora: Profª. Patrícia Helena Lucas Pranke
Pró-Reitor de Graduação: Profª. Cíntia Inês Boll
Diretora do Instituto de Informática: Profª. Carla Maria Dal Sasso Freitas
Coordenador do Curso de Engenharia de Computação: Prof. Claudio Machado Diniz
Bibliotecário-chefe do Instituto de Informática: Alexsander Borges Ribeiro

”In God we trust;
all others must bring data!”
— W. Edwards Deming

ACKNOWLEDGMENTS

I have to thank my family for all the support provided during and before the time of this graduation. I'd like to give thanks to the colleagues I've met and interacted with during the time I've spent on this course.

I'd like to give special thanks to the teachers that have taught and instructed me during this graduation, especially Anderson Rocha Tavares, for giving the support and orientation necessary for the conclusion of the current work.

ABSTRACT

Concept drift, the change of the statistical properties of data over time, is a characteristic common to data from multiple domains. It can substantially impact the performance of Machine Learning models over time. For this reason, developing methods to detect, evaluate, and mitigate these changes in data behavior is crucial for optimizing or viabilizing the utilization of Machine Learning (ML) models to solve real-world problems. In contexts where expressive drift occurs, it is desirable to understand thoroughly and to effectively utilize information on the phenomenon to apply ML solutions effectively. Hence, it is indispensable to have algorithms with demonstrated robustness. Nevertheless, to evaluate the theoretical robustness of algorithms, using data from real contexts can be a hindrance, for understanding the dynamics that generate the data in real contexts is often deficient or unknown. It is difficult to determine with sufficient precision when and why changes induced by drift occur, nor their expected impact on the performance of ML models. A solution to this problem is to use synthetic data. By using synthetic data, it is possible to better comprehend the effects of the changes in data by making the dynamics of the change of the distributions measurable and explainable by design. Another positive effect is the ability to compare and enhance algorithms, as having measurability and explainability allows the comparison of algorithm performance under drift to be more precise and objective. With this goal, an application for generating data with customizable concept drift has been developed. This application implements indicators of the theoretical limits of maximum achievable performance for models and indicators of expected performance loss for models that have yet to abstract the drift dynamics, iteration by iteration. Finally, a comparison was made with other implementations of synthetic data generators in the literature, comparing qualitative and quantitative characteristics. Contrary to other implementations in the literature, which lack at least one of the following characteristics, the proposed data generator can generate problems with varying degrees of complexity and multiple dimensions for input and output. It is suitable for classification problems, binary or multiclass, and regression problems.

Keywords: Machine Learning. Concept drift. Drift detection. Drift impact. MLOps. Synthetic data.

Medindo o Impacto de *Drift*: Um Gerador de Dados Sintéticos Customizável

RESUMO

Concept drift, mudança das propriedades estatísticas dos dados com o tempo, é uma característica comum a dados de múltiplos domínios. Ele pode impactar substancialmente a performance de modelos de Aprendizado de Máquina ao longo do tempo. Por esse motivo, o desenvolvimento de métodos para detectar, avaliar e mitigar essas mudanças no comportamento dos dados é crucial para a otimização, ou mesmo a viabilização, da aplicação de modelos de Aprendizado de Máquina à resolução de problemas reais. Em contextos em que há *drift* expressivo, para a efetiva aplicação de soluções de Aprendizado de Máquina, é desejável compreender com profundidade e utilizar bem informações sobre o fenômeno. Dessa forma, é imprescindível ter à disposição algoritmos com comprovada robustez. Contudo, para avaliar a robustez teórica de algoritmos, a utilização de dados provenientes de contextos reais pode ser um empecilho, visto que o entendimento da dinâmica que gera o dado em contextos reais é frequentemente deficiente ou desconhecido. É difícil dizer com suficiente precisão quando e por quê alterações por *drift* ocorrem, ou seu impacto esperado na performance de modelos de Aprendizado de Máquina. Uma solução para esse problema é utilizar dados sintéticos. Ao se utilizar dados sintéticos, é possível compreender melhor os efeitos das mudanças nos dados, por implementar no *design* da aplicação dinâmicas de mudança da distribuição dos dados que sejam explicáveis e mensuráveis. Outro efeito positivo é a possibilidade de comparar e aprimorar algoritmos, por possuir critérios objetivos. Com esse objetivo, foi desenvolvido uma aplicação para geração de dados com *concept drift* customizável. A aplicação implementa indicativos dos limites teóricos de máxima performance para modelos e indicativos da perda esperada de performance de algoritmos que não tenham abstraído as dinâmicas do *drift*, iteração a iteração. Por fim, foi feita uma comparação com outras implementações de geradores de dados sintéticos na literatura, qualitativa e quantitativa. Ao contrário de outras implementações na literatura, que carecem de pelo menos uma das seguintes características, o gerador de dados proposto é capaz de gerar problemas com diferentes graus de complexidade e com múltiplas dimensões para entrada e saída. Ele é adequado para problemas de classificação, binária ou multiclasse, e de regressão.

Palavras-chave: Aprendizado de Máquina, *Concept drift*, *Drift detection*, *Drift impact*, *MLOps*, Dados sintéticos.

LIST OF ABBREVIATIONS AND ACRONYMS

ADWIN	Adaptive Windowing
AI	Artificial Intelligence
AutoML	Automated Machine Learning
BO	Bayesian Optimization
CASH	Combined Algorithm and Hyper-parameter Selection
CDDG	Customizable Data with Drift Generator
CLI	Command Line Interface
DevOps	Development Operations
DDM	Drift Detection Method
EDDM	Early Drift Detection Method
GitOps	Git Operations
GP	Gaussian Processes
GUI	Graphical User Interface
HDDM	Heoffding's inequality-based Drift Detection Method
IT	Information Technology
K8s	Kubernetes
KSWIN	Kolmogorov-Smirnov Windowing
ML	Machine Learning
MLOps	Machine Learning Operations
MSE	Mean Squared Error
PDF	Probability Density Function
XGBoost	Extreme Gradient Boosting
YAML	Yet Another Markup Language

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

1.1 Context

The study of practical aspects related to making Machine Learning products for solving real problems, known as Machine Learning Operations (MLOps) (KREUZBERGER; HIRSCHL, 2022), has drawn attention to the necessity of examining aspects that are highly prevalent in real-world data and are of crucial importance for the successful operationalization of Machine Learning models (POSOLDOVA, 2020). One of these characteristics, common to data from multiple domains, is concept drift (LU A. LIU; ZHANG, 2018).

Concept drift (WIDMER; KUBAT, 1996) is the change in the statistical properties of data distribution over time. When this change occurs and is not considered, it leads to the degradation of model performance and can have disastrous effects. One way to avoid this problem is by automating tools into the MLOps pipeline to mitigate the effects of drift in the data (SYMEONIDIS E. NERANTZIS; PAPAKOSTAS, 2022).

1.2 Motivation

Understanding how concept drift occurs in a domain and using the insights from this analysis to one's advantage is challenging. In real-world data, a precise understanding of the dynamics that alter the statistical behavior of the data is often unavailable (WIDMER; KUBAT, 1996). As a result, estimating the detrimental effects of drift on models operating on such data is even more challenging.

As a means to evaluate and compare algorithms, it is necessary to have a reliable measure of their performance in different contexts. The problem of performance measurement, even without the dynamics of drift, is already complex (DIETTERICH, 1998; NADEAU; BENGIO, 2003). Therefore, the need for an understanding of drift dynamics poses an ever more significant challenge in developing algorithms that are resilient to this phenomenon.

One potential solution to this problem is artificially generating data with predictable drift dynamics. Even though there exist synthetic dataset generators in the literature, it has been observed that one often needs more satisfactory customization potential than what is currently available. The data and dynamics in other synthetic datasets often

need more inherent complexity to make them an exciting part of a relevant problem. Finally, while synthetic data offers better measurable insight into the effect of implemented features on machine learning models, it would be more promising to have more generators of data with drift that attempt to quantify the expected effects of drift on models.

1.3 Goal

This work develops a synthetic data generator with customizable concept drift. The implemented application allows, among other things, the creation of multi-dimensional data, the application of functions to the target variable, the generation of classification or regression data, and the application of linear combinations to the input space. A Command-Line Interface (CLI) and a Graphical User Interface (GUI) were implemented to aid dataset generation. For multiple configurations, an estimate is provided for the best achievable expected performance by an optimal model that has not abstracted the drift dynamics and the expected performance loss for this model at each iteration due to the changing characteristics that generate the data. The code incorporates elements that promote experiment replicability and code extensibility should there be a desire to retrieve past experiments or add new custom entities to data generation.

1.4 Contributions

The contributions and main advantages from this work include:

- A data generator with customizable drift. This generator:
  - Enable multidimensional data generation with adjustable parameters.
  - Support to generation of multidimensional output. The outputs can be continuous (for regression) or discrete (for classification).
  - Incorporation of multiple drift types.
  - Possibility to apply functions to the output values (of regression problems).
  - Estimate loss of performance: estimate the performance degradation expected from an optimal static model with each iteration of the drift dynamics.
  - Ensure reproducibility of generated data: expect little to no change from datasets generated from the same specification.

    - Extensibility: allows for easy extension through the use of abstract classes in the implementation of the main components - variables, labeling method, and sampling method).
    - Flexibility in class imbalance creation: possibility to generate datasets with varying degrees of class imbalance.
- A comparison with the drift generators and datasets with drift from the literature shows the advantages of the proposed solution to the others.
- Experiments that showcase the data generator's capacity to create intriguing data for algorithm evaluation and comparison benchmarks.

The code for the generator is located at <https://github.com/FelipeFuhr/customizable-data-with-drift-generator>.

1.5 Structure

This work is organized as follows: Chapter 2 presents a theoretical background with an overview of necessary and desirable requirements; Chapter 3 presents related works involving the generation of data with concept drift, both synthetic and from real-world; Chapter 4 contains a detailed description of the solution implementation; Chapter 5 explores experiments with drift detection algorithms on an example benchmark generated from the application; Chapter 6 concludes the work with an overview and directions for future research.

2 THEORETICAL BACKGROUND

This chapter will introduce key concepts necessary for understanding the rest of the text. These concepts are prerequisites for comprehending other essential topics, such as Random Variables, Datasets, Machine Learning, Supervised Learning, and DevOps. MLOps will shed light on why measuring drift is relevant. Other concepts, like probability distributions, play a crucial role in the developed data generator's data sampling algorithms. Rotation Matrices are introduced to make the generated data more interesting, as they combine multiple dimensions. Additionally, the Universal Approximation Theorem for Neural Networks is defined to ensure that these changes in generated data's input space resulting from rotations preserve desirable mathematical properties related to the maximum achievable performance of a model. We also define Concept Drift, as our focus is on measuring it. Finally, Drift Detection Algorithms are employed to assess the performance of a hypothetical benchmark dataset created with the data generator, thereby validating the relevance of the synthetic data generator we've developed.

2.1 Probability and Statistics

Probability and Statistics are fundamental subjects for developing and researching Machine Learning. In this work, we use probability distributions to generate synthetic data.

2.1.1 Random Variable

Random variable is a fundamental concept that is used in the definitions of the below distributions. A definition for random variable is in Hogg, Tanis and Zimmerman (2019):

Definition 2.1. *Given a random experiment with an outcome space S, a function X that assigns one and only one real number $X(s) = x$ to each element $s \in S$ is called a* ***random variable****. The* ***space*** *of X is the set of real numbers $\{x : X(s) = x, s \in S\}$.*

2.1.2 Probability Distribution

In the data generator application, probability distributions are offered as options for data sampling. The options available are the Gaussian Distribution, Weibull Distribution, and Student's t-Distribution, which are described briefly in this section.

According to a definition from Montgomery and Runger (2002), a Probability Distribution describes the probabilities associated with the possible values of a random variable *X*.

2.1.2.1 Gaussian Distribution

The Gaussian Distribution, also called Normal Distribution, is the most famous Probability Distribution used. A definition for the probability density function, from (MONTGOMERY; RUNGER, 2002):

Definition 2.2. *A random variable X with probability density function*

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \quad for \ -\infty < x < \infty$$

*is a **normal random variable** with parameters $\mu \in (-\infty, \infty)$, and $\sigma > 0$. Also,*

$$E(X) = \mu \quad and \quad V(X) = \sigma^2$$

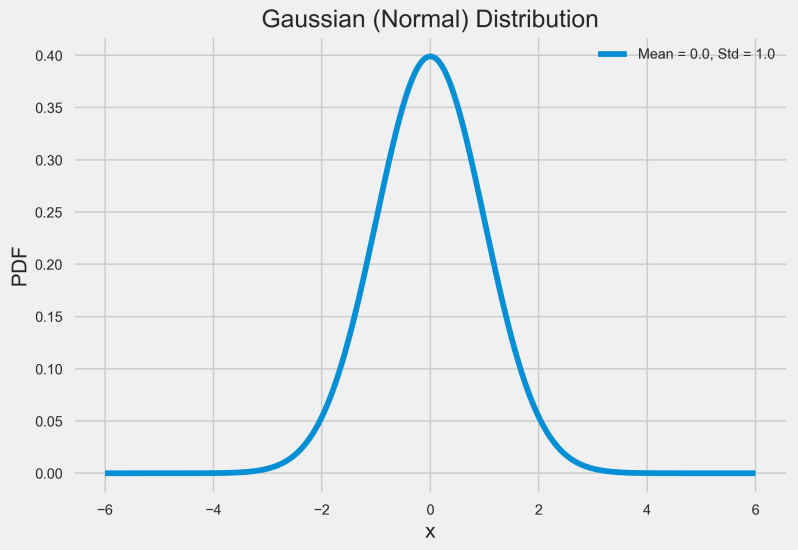
and the notation $N(\mu, \sigma^2)$ is used to denote the distribution. The mean and variance of X are equal to μ and σ^2, respectively.

2.1.2.2 Chi-square Distribution

The Chi-square distribution will be defined since it relates to the Student's t-distribution, whose implementation is used in sampling. The Student's t-distribution definition is also defined in the this section.

Chi-square, or χ^2 distribution (MONTGOMERY, 2009) is a distribution that can be defined in terms of normal random variables. If $z_1, z_2, z_3, ..., z_k$ are normally and independently distributed random variables with mean 0 and variance 1, then the random variable $x = z_1^2 + ... + z_k^2$ follows the chi-square distribution with k degrees of freedom.

Figure 2.1: Example of Gaussian Distribution

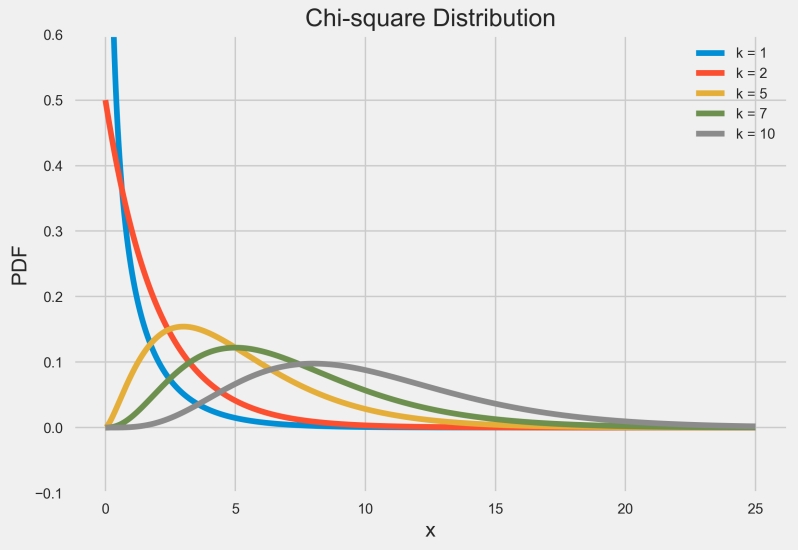


Source: The Author

The density function is:

$$f(x) = \frac{x^{(k/2)-1}e^{-x/2}}{2^{k/2}\Gamma(\frac{k}{2})} \quad \text{for } x > 0$$

Figure 2.2: Example of Chi-square Distribution with different degrees of freedom



Source: The Author

2.1.2.3 Student's t-Distribution

The Student's t-distribution (STUDENT, 1908) can be defined as follows (MONTGOMERY, 2009): If z and χ^2_k are independent standard normal and chi-square random

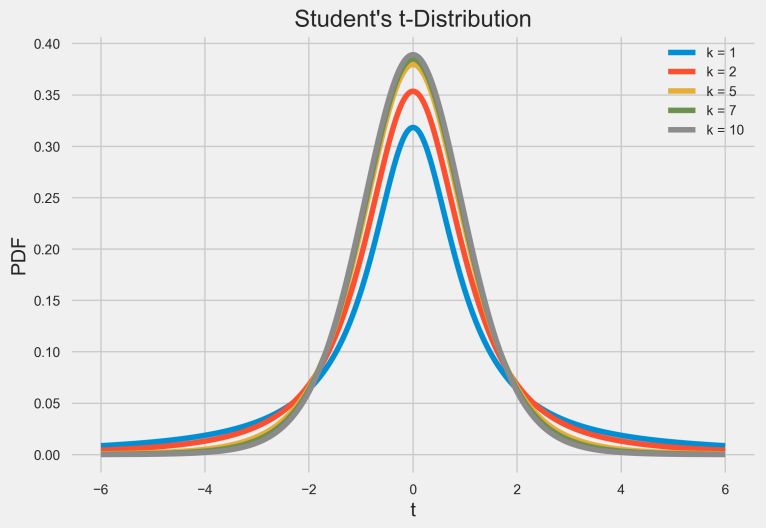
variables, then the random variable

$$t_k = \frac{z}{\sqrt{\frac{\chi_k^2}{k}}}$$

follows the t distribution with k degrees of freedom, t_k. The density function of t, whose mean and variance are $\mu = 0$ and $\sigma^2 = k/(k-2)$, for $k > 2$, is:

$$f(t) = \frac{\Gamma\left(\frac{k+1}{2}\right)}{\sqrt{k\pi}\,\Gamma\left(\frac{k}{2}\right)}\left(1 + \frac{t^2}{k}\right)^{-\frac{k+1}{2}} \quad \text{for } -\infty < t < \infty$$

Figure 2.3: Example of Student's t-distribution with different degrees of freedom



Source: The Author

2.1.2.4 Weibull Distribution

The Weibull Distribution (WEIBULL, 1951)(ABERNETHY, 2004) is a continuous probability distribution named after the Swedish mathematician Waloddi Weibull. It is versatile and can take different forms by adjusting its shape parameters. The Probability Distribution Function (PDF) for the Weibull Distribution, with shape parameter β and scale parameter η is:

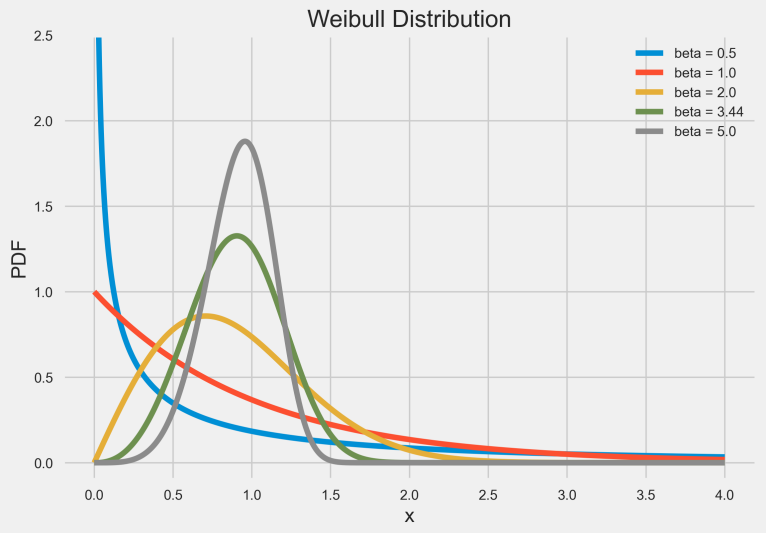
$$f\,(x) = \left(\frac{\beta}{\eta}\right)\left(\frac{x}{\eta}\right)^{(\beta-1)} e^{-\left(\frac{x}{\eta}\right)^{\beta}} \quad \text{for } x > 0 \tag{2.1}$$

The applications of Weibull analysis include failure forecasting and prediction, evaluating corrective action plans, maintenance, and cost-effective replacement strate-

gies. An advantage of using Weibull is acquiring reasonably accurate failure analysis and failure forecasts with few samples.

In the application, one can benefit from choosing the Weibull Distribution, for it is a more versatile distribution than the Gaussian and has configurable shape parameters, as illustrated in figure 2.4.

Figure 2.4: Example of Weibull Distribution with different shape parameters



Source: Adaptation from Abernethy (2004)

2.2 Rotation Matrices

Rotation matrices are used as a means to combine columns in the data generated while conserving desired properties. A rotation (AXLER, 2015) matrix is a matrix whose columns and rows are orthogonal. Geometrically, they are isometries. This means rotation matrices preserve the lengths of vectors and angles between vectors. In this work, rotation matrices comprised of sines and cosines are used. As an example, in the 2-dimensional form, they can be represented as:

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

The n-dimensional rotation matrices used in this work are all compositions of smaller 2-dimensional rotation matrices for simplicity. The orthogonal matrices used are all proper (determinant is 1) instead of reflections (determinant is -1). Some additional results from the literature that are used:

- Every proper rotation matrix is invertible.
- The inverse of a rotation matrix is also a rotation matrix.
- The inverse of a rotation matrix is continuous.
- Both the rotation matrix and the inverse of the rotation matrix are C^∞. This property derives from the fact that the rotation matrices used (and their inverses) are made of sines and cosines, and sins and cosines are both C^∞. A function is C^∞ if it is infinitely differentiable. In other words, it has derivatives of all orders.

2.3 Machine Learning

Machine Learning is a subfield of Artificial Intelligence (AI). An agent learns if it improves its performance after making observations. When the agent that learns is a computer, it is called Machine Learning (NORVIG; RUSSELL, 2020). Machine Learning builds algorithms from collections of examples. These examples can have numerous sources. Usually, this translates into collecting a dataset and then algorithmically training a statistical model on that dataset.

Machine Learning is usually separated into supervised, unsupervised, and reinforcement learning. This text focuses on supervised learning.

2.3.1 Supervised Learning

The data generator was developed focused on the solving of Supervised Learning problems.

Supervised learning is the name given to Machine Learning applications in which the training data consists of examples of the input along with the corresponding target values. Problems related to supervised learning are usually subdivided into regression if the target value consists of continuous values and classification if the purpose is to assign each input to one of a finite number of discrete values (BISHOP, 2006).

2.3.2 Dataset

A Dataset is a collection of data instances. In Machine Learning, datasets are typically used for training models by providing examples or for evaluating model performance. This text draws comparisons between the implemented dataset generator and other datasets and data generators from the literature. The main difference between a dataset generator and a dataset is the customizability of parameters, such as the desired amount of points to be generated. Usually, datasets refer to a more static version of the data and are often the product of data generators.

The notation $\mathcal{D}$ represents a dataset in this text.

2.3.3 Model Performance

In Machine Learning problems, evaluating a model's quality is crucial. One of the methods to measure the model's performance on a series of tasks is to evaluate the model with a score or metric function. The metric function assesses the distance between the outputs of the model and the desired results. The other method is to attribute a score to the model's results. Usually, for metrics, a lower value is better; for scores, a higher value is better.

An example of a metric is the Mean Squared Error. As for an example of a score, the accuracy is a widespread choice for evaluating Machine Learning models.

MSE is only used directly in this text to show the performance degradation of models under drift. Understanding MSE may make understanding other essential concepts, such as optimality, easier. It can be fruitful to understand learning problems first as error minimization problems. A low MSE is often related to a high accuracy, and lowering the MSE of a model is a tool employed when the main interest is to increase the accuracy of a model. Mean Squared Error (MSE) is a metric that computes the average squared differences between the actual values (target) and the model's predicted values. It can be represented mathematically as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

In this formula, n is the number of points, y_i is the actual value for the ith data point, and $\hat{y}_i$ is the ith predicted value.

2.3.4 Optimal Models

An Optimal Model is defined since it will be used to justify the theoretical bounds for the best achievable model trainable on the generated data.

An optimal model is a machine learning model that achieves the best possible performance according to some metric, given the context.

For example, given a loss metric $\mathcal{L}$ (such as MSE), a dataset $\mathcal{D}$, and a set of models $M = \{M_1, ..., M_n\}$, one could define an Optimal Model M^* as:

$$M^* = \underset{M_i}{\operatorname{argmin}} \mathcal{L}(M_i, \mathcal{D})$$

In other words, the optimal model M^* is the model that minimizes $\mathcal{L}$ when evaluated with $\mathcal{D}$.

2.3.5 Random Forest

The Random Forest algorithm is an ensemble learning method typically used for classification or regression. It combines multiple decision trees to achieve the desired result.

2.3.6 XGBoost

Extreme Gradient Boosting (XGBoost) is an ensemble learning algorithm for classification and regression that leverages predictions of multiple weak models (often decision trees). A weak model is a model that only needs to be slightly better in performance than a random model. It is known for its efficiency and speed, even in complex datasets. Thus, it is often the first choice for a user to do exploratory analysis in a dataset or ML problem.

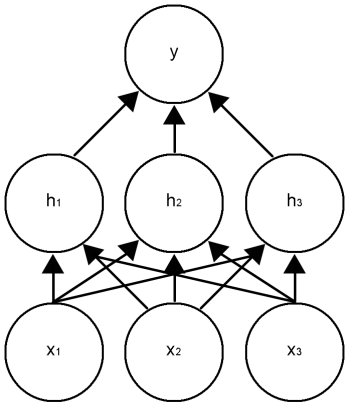
2.3.7 Neural Networks

Neural networks are popular models in ML. In this work, their importance is restricted to the *Universal Approximation Theorem for Neural Networks*, which serves as a

means to have guarantees on the preservation of properties that allow the estimation of the maximum expected performance of an optimal model trained on a given dataset generated by the synthetic data generator. They derive from attempts to find mathematical representations of information processing in biological systems (BISHOP, 2006). Inspired by the brain's structure, neural networks are probably the most famous class of Machine Learning models. The structure replicates the idea of neurons by having interconnected nodes that process inputs as signals and perform one or more desired tasks.

A depiction of the architecture of a hypothetical neural network is given in figure 2.5. It has an input layer, one hidden layer, and one output layer.

Figure 2.5: Architecture of a hypothetical Neural Network



Source: Adaptation from Goodfellow, Bengio and Courville (2016)

2.3.8 Universal Approximation Theorem for Neural Networks

This theorem is used to justify that theoretical bounds remain unchanged for the accuracy of hypothetical models on the generated data even if we make some changes to the original columns. For example, if an optimal model M^* was found for solving a particular task on a dataset $\mathcal{D}$, and if some rotations were made on the columns of $\mathcal{D}$, generating $\mathcal{D}_{new}$, a new model M^*_{new} with the performance as close as desired to that of M^* on $\mathcal{D}$ can be found. This model would be a composition of a neural network from the Universal Approximation Theorem for Neural Networks that maps the inverse of the rotations that generated $\mathcal{D}_{new}$ from $\mathcal{D}$ and the original model M^*.

The Universal Approximation Theorem for Neural Networks (CYBENKO, 1989; KIDGER; LYONS, 2020) states that, for any continuous function f defined on a compact subset of $\mathbb{R}^n$, $\epsilon > 0$, it is possible to find weights and biases for a single hidden layer

feedforward neural network (with output $F(x)$), such that $|F(X) - f(x)| < \epsilon$.

This means that we can approximate any continuous function within a compact subset with a fitting neural network.

2.3.9 Gaussian Processes and Bayesian Optimization

While studying Bayesian Optimization (BO) in the context of Automatic Machine Learning (AutoML) algorithms (AutoML definition can be found in Appendix B), the idea of using Gaussian distributions to fit a function, because of their properties of being highly expressive, smooth, and easy computability, served as inspiration for the use of probability distributions such as the Gaussian to create the input space by sampling, as well as generating metrics for understanding the generated behavior.

Bayesian Optimization is an iterative algorithm with two main elements: a probabilistic surrogate model and an acquisition function. It is a popular tool for solving hyperparameter optimization problems and is a crucial ingredient to many AutoML algorithms (HUTTER; KOTTHOFF; VANSCHOREN, 2020).

Gaussian processes (GPs) extend multivariate Gaussian Distributions to infinite dimensions (EBDEN, 2015). Traditionally, the target function to be modeled using Bayesian Optimization employs Gaussian Processes because it consists of closed-form expressions, is highly expressive, smooth, and can calibrate uncertainty estimates satisfactorily (HUTTER; KOTTHOFF; VANSCHOREN, 2020).

2.4 Concept Drift

2.4.1 Definition

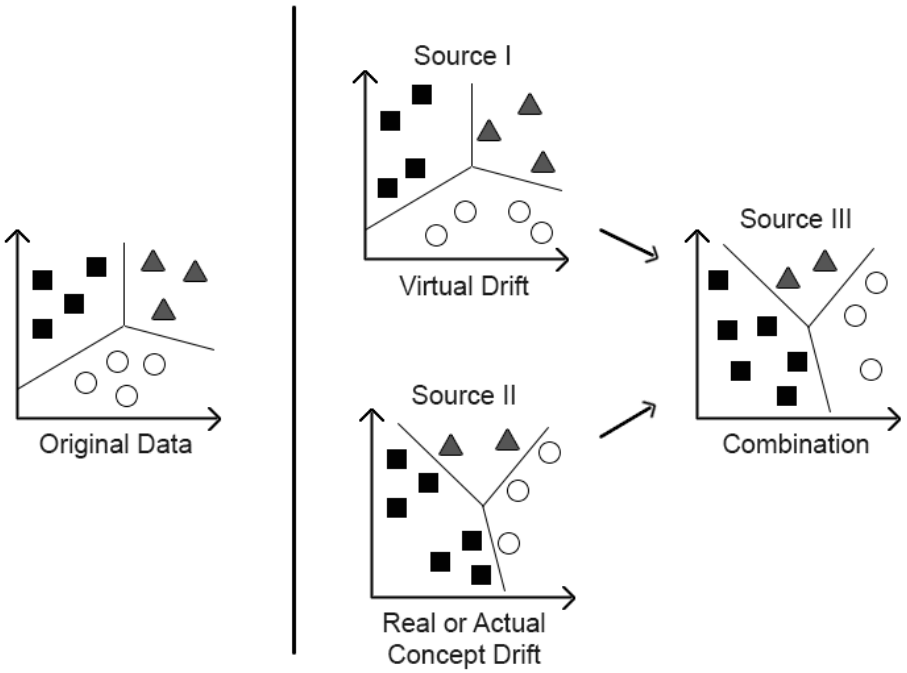
Concept drift can be defined as the phenomenon in which the statistical properties of a target domain changes over time (LU; ZHANG; LU, 2014). In Lu A. Liu and Zhang (2018), drift is defined as follows:

Given a sequence of time periods $[0, ..., t]$, a set of samples $S_{0,t} = \{d_0, ..., d_t\}$, where $d_i = (X_i, y_i)$ is one observation of a data instance, with X_i being the feature vector and y_i the label, and $S_{0,t}$ follows a distribution $F_{0,t}(X, y)$, Concept drift occurs at time $t + 1$, if $F_{0,t} \neq F_{t+1,\infty}(X, y)$. In other words, $\exists t : P_t(X, y) \neq P_{t+1}(X, y)$.

Concept drift can also be defined as the change of joint probability of X and y at time t. This can be triggered by three different sources:

- **Source I**: $P_t(X) \neq P_{t+1}(X)$, while $P_t(y|X) = P_{t+1}(y|X)$. It has been labeled virtual drift (RAMíREZ-GALLEGO et al., 2017). It means the decision boundaries remain unchanged, but the distribution of the data within this boundaries change (and thus can make a virtual change to the performance of a model).
- **Source II**: $P_t(y|X) \neq P_{t+1}(y|X)$, while $P_t(X) = P_{t+1}(X)$ and $P_t(X)$ remains unchanged. In practice it is a change in the decision boundaries. It is called actual drift.
- **Source III**: A mixture of the two other sources. Both $P_t(X)$ and $P_t(y|X)$ change at the same time.

Figure 2.6: Sources of concept drift



Source: Adaptation from Khamassi et al. (2018)

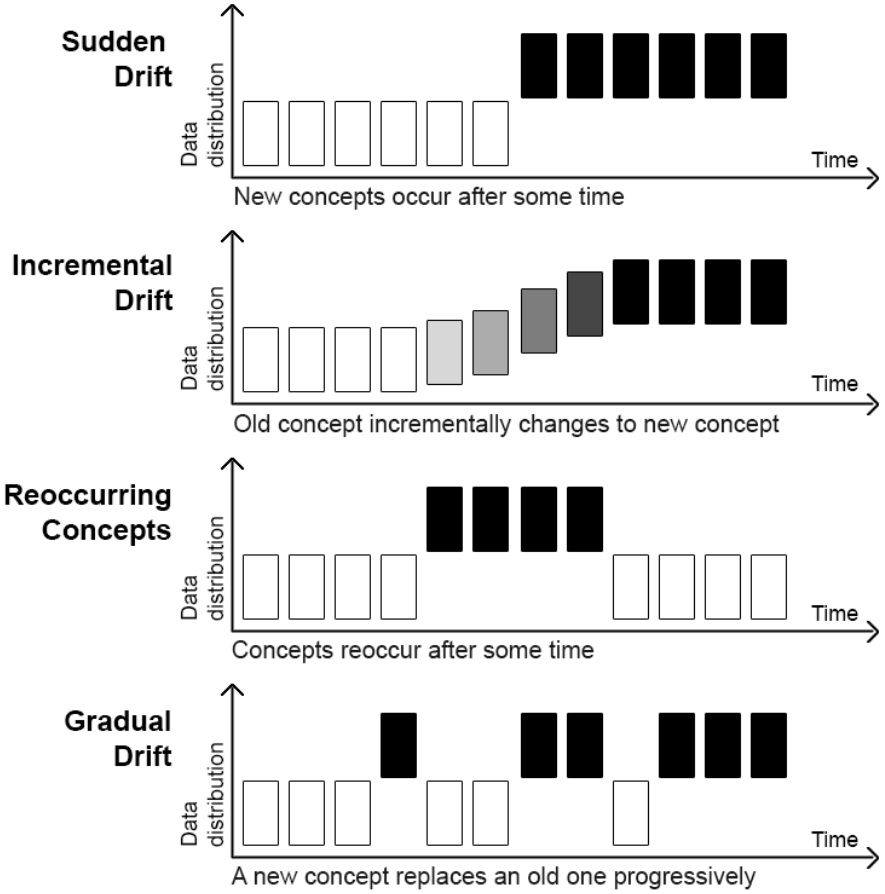
Some types of concept drift are:

- Sudden Drift, or abrupt drift: the concept changes suddenly at a point in time.
- Incremental Drift: old concept changes incrementally to new concept.
- Gradual Drift: an old concept is replaced by a new concept gradually over time.

Gradual Drift, Incremental Drift, and Reocorring Concepts. These types are depicted in

figure 2.7.

Figure 2.7: Some concept drift types



Source: Adaptation from Lu A. Liu and Zhang (2018)

2.4.2 Drift Detection

According to Lu A. Liu and Zhang (2018), *drift detection* refers to the techniques and mechanisms for characterizing and quantifying concept drift. A general framework is provided with the following four stages:

- Data Retrieval (Stage 1): retrieves data chunks from data streams.
- Data Modeling (Stage 2): abstracts the retrieved data and extracts key features.
- Test Statistics Calculation (Stage 3): quantifies the severity of the drift by measuring dissimilarity.
- Hypothesis Test (Stage 4): evaluates the statistical significance of the changes quantified in Stage 3.

Detection algorithms help to understand drift by retrieving information about "When" (when and how long the drift occurs), "How" (severity of the concept drift), and "Where" (regions of the domain where drift occurs). Some algorithms for drift detection are Drift Detection Method (DDM) (GAMA et al., 2004), Adaptive Windowing (BIFET; GAVALDà, 2007), Early Drift Detection Method (EDDM) (BAENA-GARCÍA et al., 2006), Heoffding's inequality-based Drift Detection Method (HDDM) (FRIAS-BLANCO et al., 2015), Kolmogorov-Smirnov Windowing (KSWIN) (RAAB; HEUSINGER; SCHLEIF, 2020), and Page-Hinkley (PAGE, 1954).

2.5 DevOps

2.5.1 Definition

According to Kim et al. (2016), Development Operations (DevOps) and its practices represent a convergence of philosophical and management movements. It is the outcome of applying successful principles from physical manufacturing and leadership to IT. These principles have come from the Lean, the Theory of Constraints, the Toyota Production System, and resilience engineering, among others.

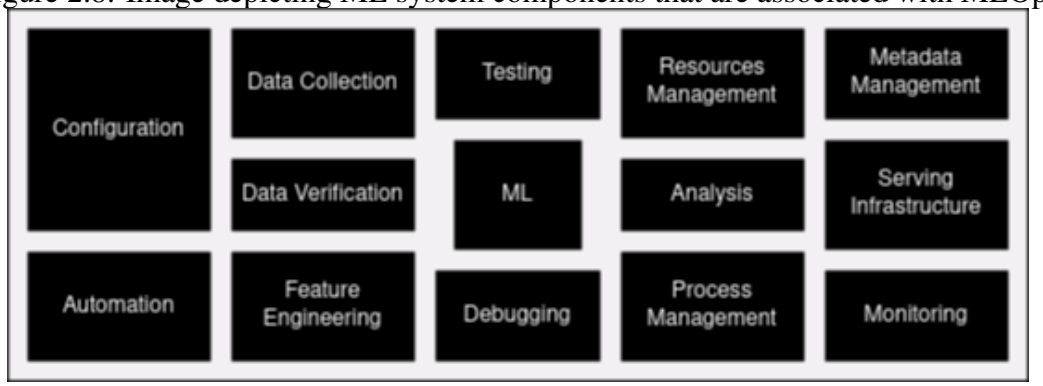
The DevOps concept can be used to define MLOps since many consider MLOps to be partly DevOps and partly ML. Ideas that come from DevOps tools, such as Kubernetes, and best practices, such as reproducibility, inspired the application's development.

2.5.2 MLOps

Machine Learning Operations (MLOps) can be defined as the standardization and streamlining of machine learning lifecycle management (TREVEIL; TEAM, 2020). MLOps is also an ML engineering culture and practice, and is often described as an attempt at unifying Machine Learning with DevOps (Cloud Architecture Center, 2023).

Image 2.8 illustrates components associated with MLOps besides a generic ML model. It can be inferred that the scope is different from traditional ML.

Figure 2.8: Image depicting ML system components that are associated with MLOps



Source: Adaptation from Cloud Architecture Center (2023)

3 RELATED WORK

This chapter discusses the characteristics of datasets exhibiting drift and the data drift generators available in the literature, highlighting their advantages and disadvantages. We begin with an inspection of real-world datasets, followed by a review of synthetic data, including both synthetic datasets and synthetic data generators. In becomes clear, with table .

3.1 Real-world Datasets

One of the advantages of using real-world datasets for developing ML algorithms is that real-world data provides a stronger conviction that solving a problem on that dataset will have practical utility. However, as observed from an extensive list of real-world datasets in Lu A. Liu and Zhang (2018), there are at least two shortcomings:

- There are proportionally a lot less datasets with more than two classes.
- Most datasets have either too few or too many attributes.

In addition to the noted limitations within these specific datasets, a general and significant concern can be added to the problem of using real-world datasets with drift. As observed in Rudin (2019) for blackbox Machine Learning models, explaining a behavior whose origins we do not fully comprehend after it occurs and thoroughly dominating its causes are two distinct things. The point made in the article is that there is a fundamental difference between having an explanation from an inherently explainable model and explaining a black-box model after its predictions were made.

However, the same argument can be made about drift in datasets. There is a difference between using a real-world dataset that has drift but whose origins for the drift lack understanding and having a dataset whose dynamics behind the drift are known and explainable. In the second case, one has the exact criteria for why a model object behaves the way it does; in the first case, the best one can have is a well-educated guess. Thus, it may be better to evaluate models using a dataset whose drift is interpretable by design than a dataset whose reasons for drift are unknown, depending on the context.

3.2 Synthetic Data

3.2.1 Synthetic datasets for concept drift detection purposes

In Lobo (2020), there is a collection of twenty diverse synthetic datasets: ten present abrupt and ten present gradual drift. These datasets were generated using stream generators and functions. Some of these generators are popular in the literature, such as *Sine* (GAMA et al., 2004). Although an exciting alternative for evaluating models, since using the same dataset adds some amount of reproducibility to experiments, there is an explicit limitation to this approach, which is the need for more flexibility. The drift behavior occurs at fixed times and often is too simple.

3.2.2 Concept Drift Datasets v1.0

In Song Qiao Hu (2023), some drift generators present thrilling behavior. In this repository, it is possible, for example, to generate both linear and rotating. Four categories are provided: “linear”, “cake”, “rotating chocolate”, and “rolling torus”. They all contain abrupt, sudden, gradual, and recurrent types of drift. Although the behavior is fascinating, a limitation is that there is no support for generating multidimensional input.

3.2.3 Popular synthetic datasets generators

Table 3.1, based on Lu A. Liu and Zhang (2018), lists some of the most famous drift generators from the literature, as well as some high-level features.

Some of the perceived advantages in the synthetic dataset generators include:

- Some allow to customize the number of classes and input dimensions.
- The number of instances is adjustable.
- In most of these, the behaviour that generates drift is known.

Some of the disadvantages are:

- Generators are predominantly limited to binary classification.
- Fixed and small amount of features in most datasets.
- Most datasets need more inherent complexity to be an exciting problem to solve

with state-of-the-art ML models.

Table 3.1: Adaptation of table from (LU A. LIU; ZHANG, 2018). Adaptation of table from (LU A. LIU; ZHANG, 2018). The number of instances is not specified because a user can create a custom amount of points. The number of attributes, number of classes, types of drift, and sources of drift are specified. The last row is the implementation from this work, Customizable Data with Drift Generator (CDDG).

Generator	*Attributes*	*Classes*	*Type*	*Source*
STAGGER	3	2	Sudden	II
SEA	3	2	Sudden	II
Rotating hyperplane	10	2	Gradual, Incremental	II
Random RBF	Custom	Custom	Sudden, Gradual, Incremental	III
Random Tree	Custom	Custom	Sudden, Reoccurring	II
LED	24	10	Sudden	II
Waveform	40	3	Sudden	II
Sine	2	2	Sudden	II
Circle	2	2	Gradual	III
Rotating chessboard	2	2	Gradual	II
CDDG	Custom	Custom	Gradual, Sudden, Reoccurring	II and III

The Author

It can be observed that, besides the functionality that allows the user to use regression instead of classification, the generator allows the user to have an estimative of the maximum achievable performance for the models, and the generator allows the user to have more flexibility than all of the previous datasets and data generators.

4 PROPOSED SOLUTION

This chapter is structured as follows: in the Overview, it is given an overview of the application's idea, some characteristics, and a brief introduction to the main components (Variable, Sampler, Labeler); then, a more in-depth description of these components, with illustrations; after, the possibility to use UI to aid in the interface is mentioned; then, the theoretical characteristics of the measurement are explored; then, explanations on the rotations; and finally, some implementation details.

The structure of this chapter unfolds as follows:

- Overview: this section offers a glimpse into the application's idea, highlighting key characteristics and providing a introduction to its primary components: Variable, Sampler, and Labeler.
- Application Main Components: a more detailed explanation of the already introduced main components.
- Dataset Generator: a explanation on how the Dataset Generator is supposed to be used.
- User Interface (UI): a brief description of the UIs that were implemented to aid the user in the generator usage.

Component Deep Dive: Following the overview, we delve deeper into these components, offering detailed explanations accompanied by illustrative examples.

User Interface Integration: Subsequently, we explore the potential of incorporating a user interface (UI) to enhance the user experience.

Theoretical Measurement Characteristics: In the next segment, we delve into the theoretical aspects governing the measurement characteristics, offering a thorough exploration.

Exploration of Rotations: We then shift our focus towards elucidating the intricacies of rotations and their role within the application.

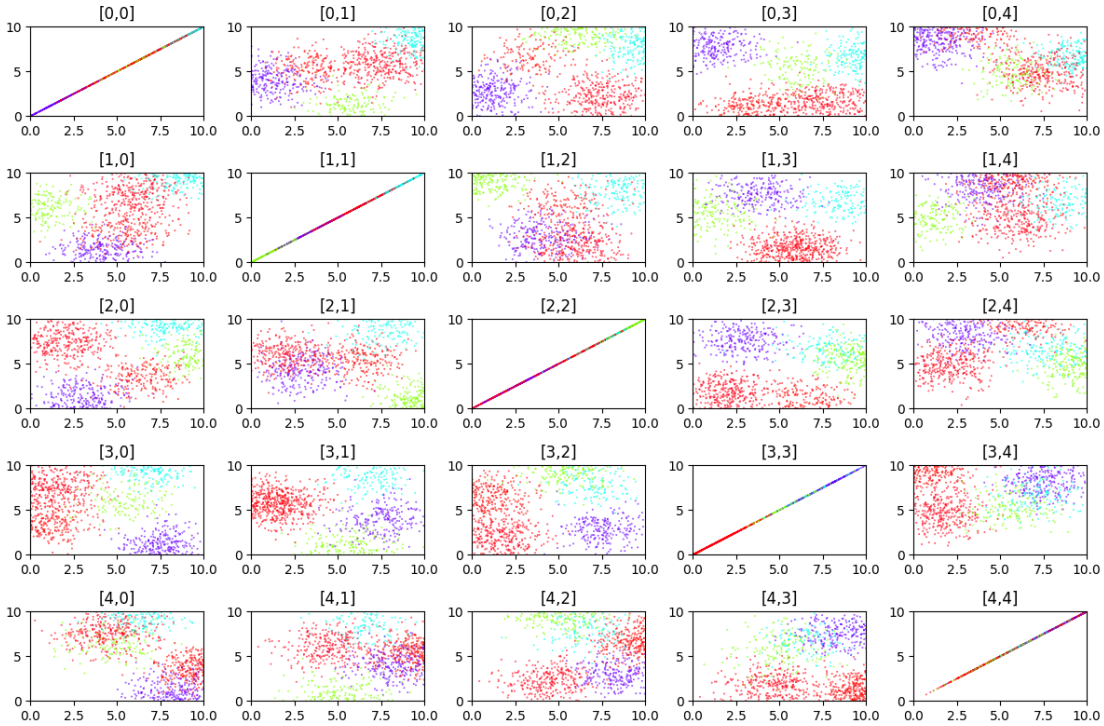
Implementation Insights: Finally, we conclude with a discussion of various implementation details, providing a comprehensive understanding of the practical execution of the application.

4.1 Overview

The proposed solution is a data generator with customizable drift. The Sampling and Labeling methods are implemented as separate entities. Some predefined options exist (such as Gaussians, Weibull, and Student's); however, the application's architecture was designed to be easily extendable to other desired distributions.

It is possible to determine the number of dimensions that are desired for the input of the resulting data, as well as the number of output (or target) columns. The user can also decide if the problem being solved will be classification or regression. Figure 4.1 illustrates the creation of a dataset with 5 dimensions.

Figure 4.1: Example of a dataset generated with five input dimensions. The specification for the data generator asked for a 5-dimensional dataset with multiple clusters. The 2-dimensional slices are depicted in the figure



Source: The Author

The data generation process requires the details of the desired data to be generated to be determined with a YAML file specifying the desired parameters for the data. The dynamics of change for each iteration must also be specified. For example, if a user is using a Harmonic Variable (periodic function such as sine and cosine, meaning the values of the variable have a predefined periodic behavior) as the value for the mean of a Gaussian Distribution, a period of 10 would imply that every 10 iterations of the generator, the mean of the Distribution would go back to its original value.

One of the advantages of this simulator is having theoretical bounds for the accuracy of a static optimal model. If a model is trained solely on the data from a specific iteration, this value can be considered a maximum achievable performance. There is also an expected loss of accuracy from one iteration to another for this hypothetical optimal model, which is also calculated in every iteration. Further research would be necessary for other metrics besides accuracy, such as F1. In section 4.3, this topic will be further discussed.

When the labeling method follows the distribution that samples the data (for example, a Gaussian) and the Gaussian changes, a drift of source III is generated. In this case, it is hard to separate the source I and the source II. If a labeling method such as a grid is used, which gives a label for each point in space according to a grid in space, the distributions that generate the clusters can change position in space, generating virtual drift. If the decision boundary of the grid changes, for example, it is a source II (or actual) drift.

With Harmonic Variables, it is possible to implement both reoccuring and incremental drift. The implemented changes for labeling on the classification case are limited to sudden drift. However, it can approximate gradual drift if enough clusters are used and only a few clusters change labels at each iteration.

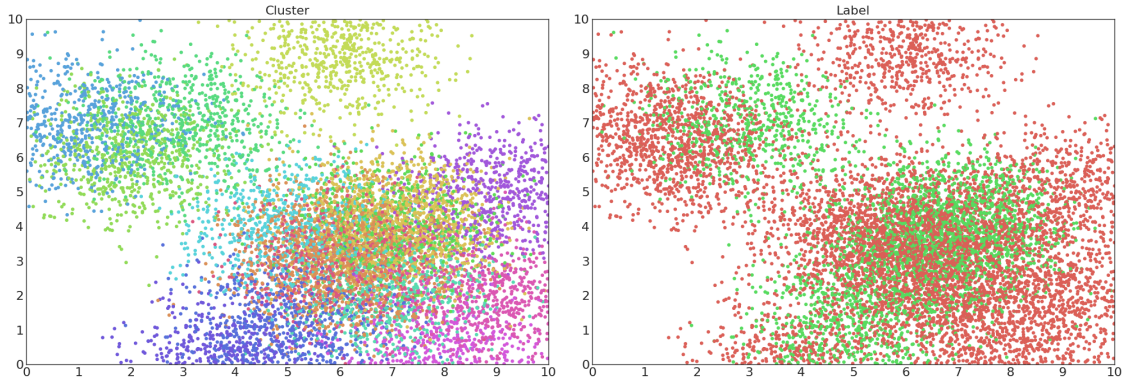
4.2 Application Main Components

The application has the following main components for creating dynamic behavior data: Variable, Sampler, and Labeler. Figure 4.2 shows points from a sampling before and after labeling; each component's details are given in a subsection.

4.2.1 Variable

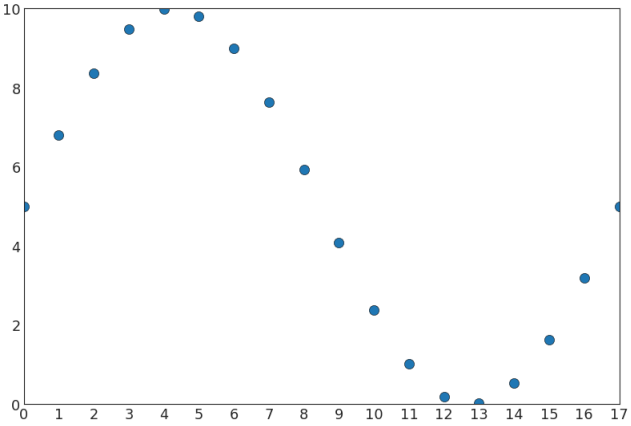
Variables are a building block for the specification of the sampling method. There are two kinds of variables implemented: constant and harmonic. Constant remains unchanged with every iteration; harmonic follows a sine or a cosine with a chosen period. Figure 4.3 illustrates a harmonic variable.

Figure 4.2: Sampling and labeling illustrated for a binary classification problem. On the left figure, points sampled and their corresponding clusters are identified; on the right figure, there are the same points with attributed labels.



Source: The Author

Figure 4.3: Example of Harmonic Variable with mean 5, amplitude 5 and period 17



Source: The Author

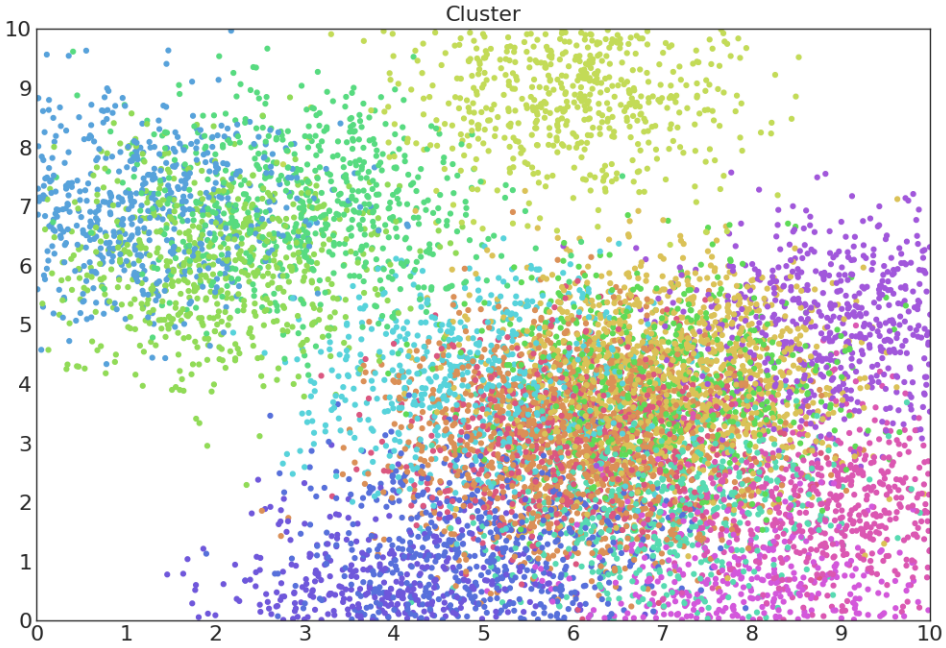
4.2.2 Sampler

Sampler is a component responsible for generating clusters of data. The default option is to use Gaussian, but Student's and Weibull are also supported. To simplify calculations, the distributions are not in their n-dimensional form, but are computed dimension by dimension separately. Figure 4.4 illustrates a dataset with multiple clusters.

Notice in the figure above that although the clusters appear convoluted, the original dataset has many dimensions in this particular case. Thus, the clusters are farther away than they appear in this specific 2-dimensional slice.

Combining Harmonic Variables with Gaussian clusters for sampling enables the possibility of generating complex drift behavior. For example, in figures 4.5 and 4.6, the weight of each cluster (affects the proportion of data points), the mean, and the variance of each cluster varies. The black dots represent the mean, their size, the weight of the corresponding cluster (which affects the proportional number of points drawn from that

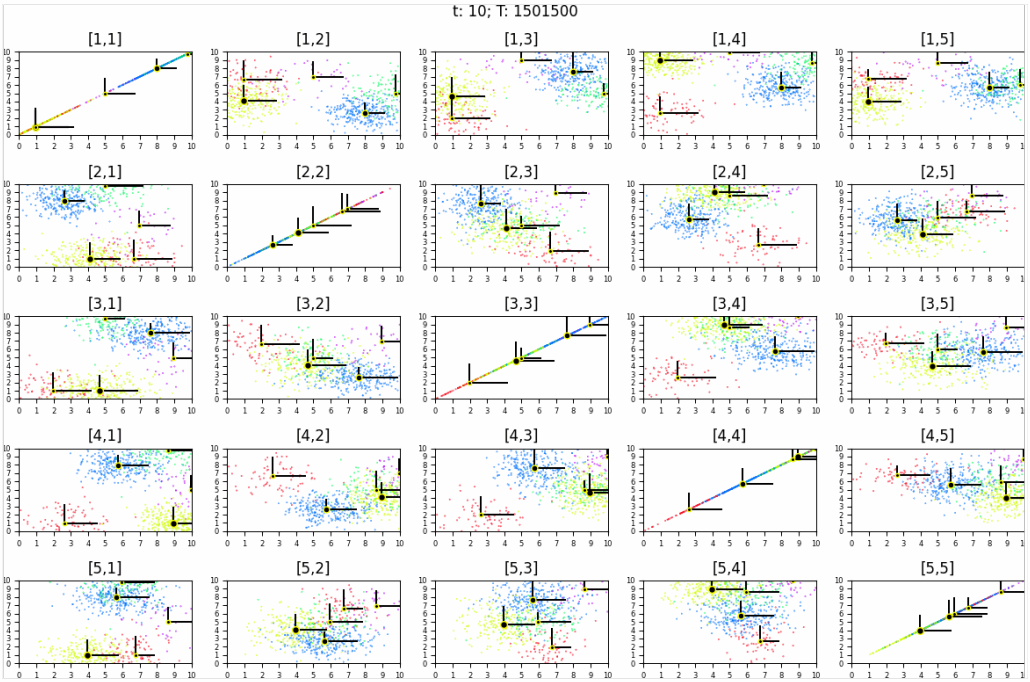
Figure 4.4: 2-dimensional slice of a dataset with multiple clusters. Each color represents one cluster. After Sampler samples, the Labeler can attribute classes to the clusters according to the input specification, as explained further



Source: The Author

cluster), and the lines the variance of the Gaussians for each dimension.

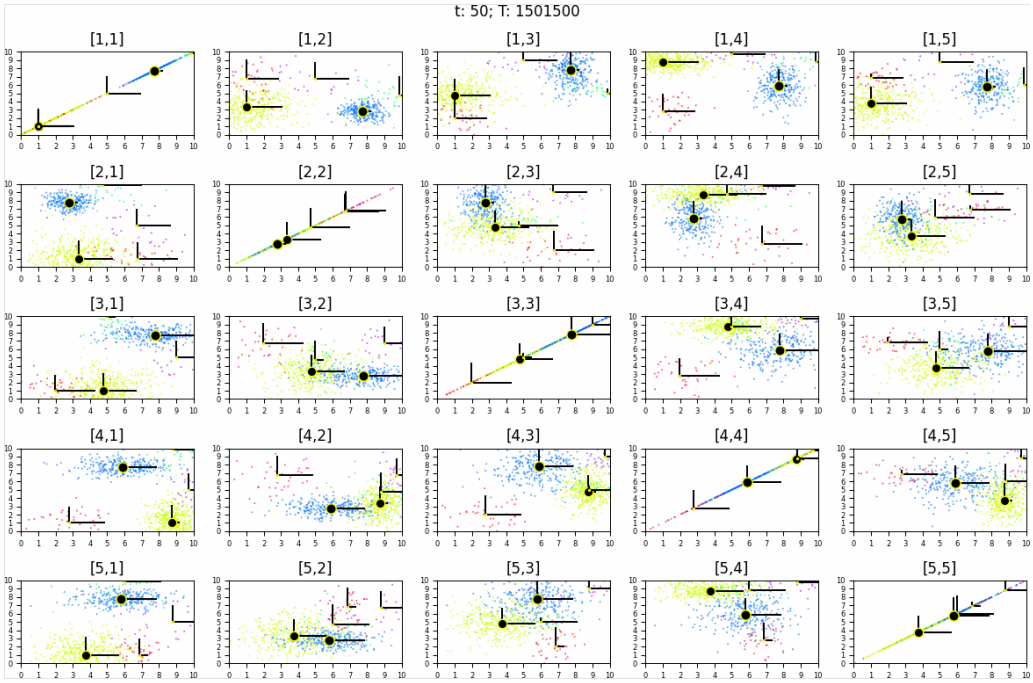
Figure 4.5: Hypothetical dataset at iteration 0



Source: The Author

Often, the algorithms and metrics used in imbalanced data problems differ substantially from those used in the balanced counterpart (SANTOS et al., 2018). The pos-

Figure 4.6: Hypothetical dataset at iteration 50



Source: The Author

sibility of varying the weights from each cluster, a feature that allows a user to create imbalanced clusters and classes of data, makes the data drift generator a convenient alternative for researching algorithms suitable for imbalanced data.

4.2.3 Labeler

Labeler is the method for assigning values to the points sampled with Sampler. For classification, it supports label by cluster (attributes one label to each cluster), thus making all points from each cluster have the same label, or by grid, thus making all the points within designated boundaries belong to the same class. For regression, one can either return the density of the cluster that has the highest probability of having generated the designated point, or a function applied to this value.

For the regression case, if the user decides to apply a function to the values, it is interesting to notice that if the chosen function is invertible and continuous, by the Universal Approximation Theorem, it is possible to create a Neural Network that approximates the continuous inverse function as closely as desired. Thus, the same argument used for choosing rotation matrices is valid since the theoretical boundaries for a static optimal model are maintained. If before the application of an invertible function a hypothetical optimal model M^* was the best model, after the application of the function, a new as

close as necessary optimal model could be inferred by composing the neural network that approximates the function with the previous optimal model.

4.2.4 Dataset Generator

The dataset generator provides a intuitive way of generating datasets. It requires the user to specify the desired parameters in a versatile way.

For example, users are granted the ability to define a myriad of parameters with multiple options. These parameters include but are not limited to, the number of dimensions, type of cluster, rotations, type of variable to use for each compatible numeric variable, dynamics of variables (including type, amplitude, and period if the variable is harmonic), and the nature of the task (binary, multiclass, or regression), among a plethora of other options.

A UI, in the form of a Command Line Interface (CLI), and another one, a Graphical User Interface (GUI), were developed to aid the user in the data generation and in the management of the specifications that are required to generate data.

The data being generated can be drawn for as much as the user likes and keep the same statistical properties as long as the user does not make one iteration. The characteristics change according to the specified rules every time an iteration is made. As an illustration, if a Gaussian Variable had a mean of 5, with harmonic variable or amplitude 3, and a period of 7, every 7 iterations, the dynamic of this variable would go back to its starting point at 5, as depicted in figure 4.3.

4.2.5 UI

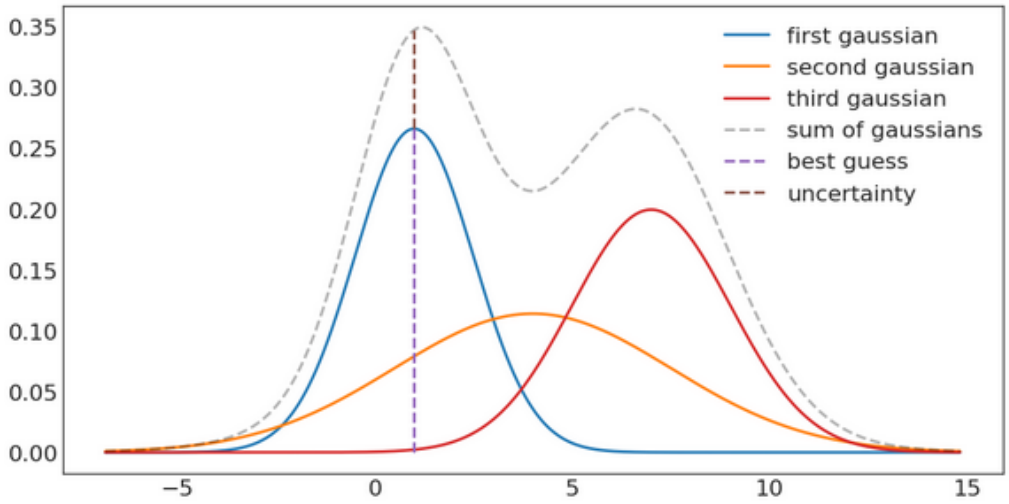
A CLI was developed to aid in the dataset generation. With the command *generate*, the user can specify the path to a specification file and generate a dataset with the desired properties.

A GUI has also been integrated into the system. This intuitive GUI empowers users to create the specification in YAML without knowing how this file format works or having to define each parameter, thus helping craft a custom dataset that aligns requirements with the specific type of dataset intended for exploration.

4.3 Measurement

Theoretical boundaries can be inferred for the composition of distributions. As an example, consider image 4.7:

Figure 4.7: Composition of Gaussian clusters in one dimension



Source: The Author

This image illustrates the best answer for each point in a one-dimensional problem. It has three clusters, of which points are drawn. If there are three corresponding labels, one for each cluster, the best guess at each point is the cluster whose Gaussian has the highest density at that point. For example, at point 1, the cluster of the blue Gaussian is the best guess. The best answer at this point will have an accuracy of

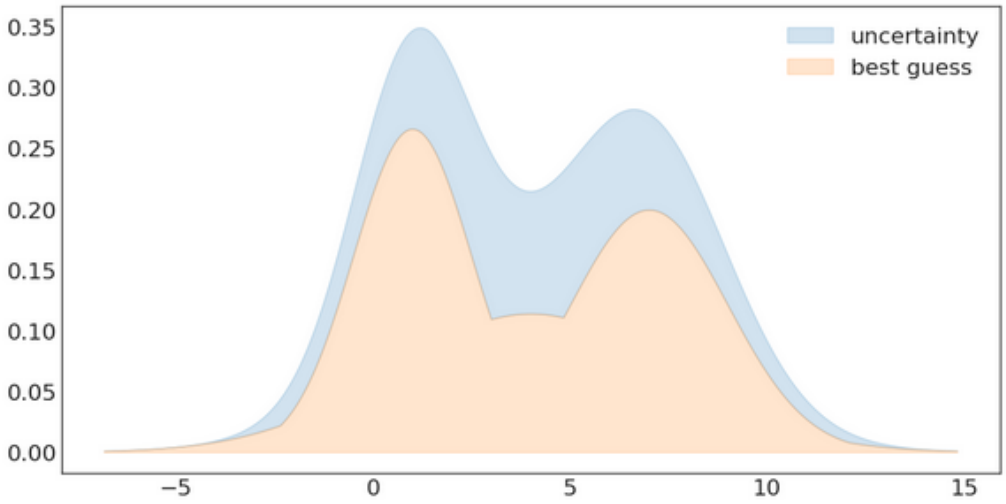
$$\frac{bestguess}{bestguess + uncertainty}$$

The best accuracy achievable would be the accuracy from a model M^* that, for each point in the dataset, predicts the cluster whose Gaussian is proportionally highest at that point. This value corresponds to the normalized area of the figure 4.8.

When the distributions corresponding to each cluster change due to an iteration of the drift dynamics, a hypothetical, optimal model that is static is expected to lose performance proportional to the areas of figure 4.9:

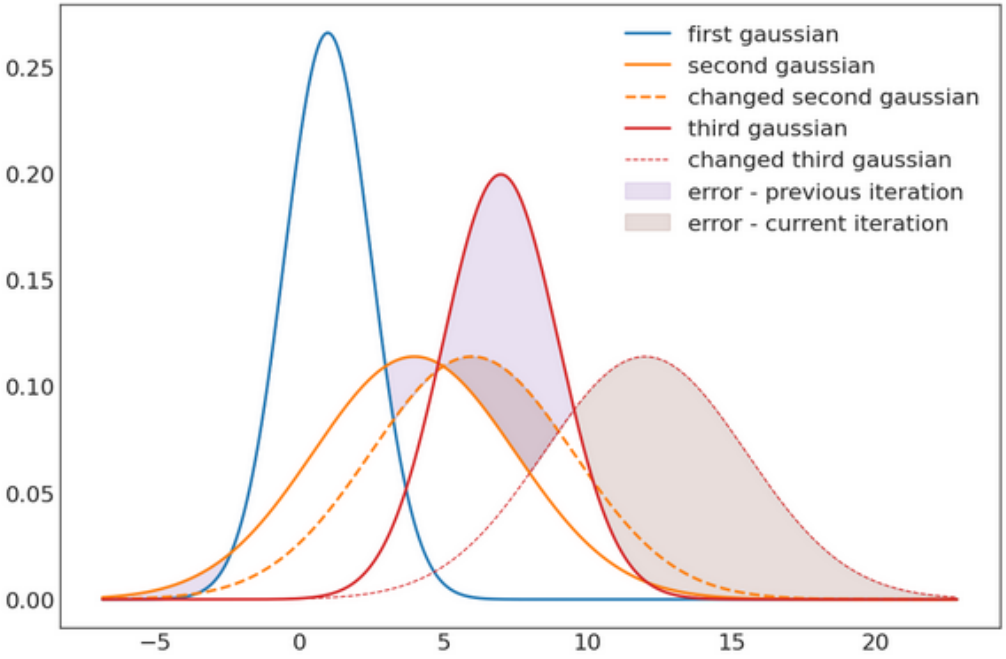
In the multidimensional case, this problem translates into finding the dimension whose Gaussian is proportionally the highest for each point. Although an interesting approach to create clusters and attribute labels to them, unfortunately, it limits the capacity to generate interactions between columns, which are complex behaviors typical to most

Figure 4.8: Illustration of the areas corresponding to the best guess at each point and to the uncertainty



Source: The Author

Figure 4.9: Illustration of the effects of a hypothetical change from one iteration to another



Source: The Author

relevant datasets.

The capacity to make static rotations in the hyperplane was added to solve this issue and add more relevance to the generator. This way, we conserve the idea of measuring the maximum achievable performance at each iteration and the expected differences iteration by iteration.

This approach is guaranteed by the Universal Approximation Theorem for Neural Networks since if an optimal model M^* can be found to match the generated dataset $\mathcal{D}$, and the inverse of rotations is infinitely continuous, it is possible to use the theorem to

find a neural network N that approximates the inverse of the rotation function as closely as desired. Thus, an optimal model M^*_{new} for the dataset $\mathcal{D}_{new}$ would be $M^*_{new}(\mathcal{D}_{new}) = M^*(N(\mathcal{D}_{new}))$

4.4 Rotations

As previously mentioned, the Universal Approximation Theorem for Neural Networks assures that finding an inverse rotation to the applied rotation is feasible, adding compelling complexity to the problem of identifying the dimension with the most pertinent information. Thus, in this scenario, an optimal model, or a near-optimal model, would be close to a composition of a neural network that abstracts the inverse rotation with the optimal model from the previous case.

The rotations can be dynamic; however, only the theoretical boundaries for the static case have been studied. If the rotation of the dimensions does not change over time, its characteristics do not negatively impact the expected loss of performance from a model from iteration to iteration. Rotations have other interesting properties, which can be intuitively seen in images: they are isometries, meaning they do not change the characteristics of a metric, such as distances. The clusters do not expand or contract due to the rotation by fixed values; they are only rotated a fixed amount from their original position.

4.5 Implementation Details

The application is implemented in the Python 3 programming language [1]. A GitHub [2] repository was used to store the code. GitHub actions [3] and pre-commit [4] are some tooling used for Git Operations (GitOps). Documentation for classes and functions was created following the Numpy standard [5]. Documentation for this application was generated using Sphinx [6]. Unit tests were created using pytest [7], and behavior tests

[1] https://www.python.org/
[2] https://github.com
[3] https://github.com/features/actions
[4] https://pre-commit.com/
[5] https://numpydoc.readthedocs.io/en/latest/format.html
[6] https://www.sphinx-doc.org/en/master/
[7] https://docs.pytest.org/en/7.4.x/

were created using behave [8]. Examples of Docker containers [9] and Kubernetes [10] resources were implemented to enhance portability and facilitate running the dataset generation service across different environments. The code for the generator is located at <https://github.com/FelipeFuhr/customizable-data-with-drift-generator>.

[8] https://behave.readthedocs.io/en/latest/
[9] https://www.docker.com/
[10] https://kubernetes.io/

5 EXPERIMENTS

This work was focused on implementing a data generator that aids in generating benchmarks for evaluating Machine Learning algorithms under the presence of drift. The following experiment was built to illustrate the practical utility of this generator.

5.1 Benchmark Experiment

A dataset with ten dimensions was generated for this validation. While trying to calibrate the number of clusters to pose an interesting problem, it was found that algorithms such as XGBoost and Random Forests could easily overfit the data if a small number of Gaussian clusters were used.

After some trial and error, a dataset of 150 clusters with binary classes was devised. The clusters are spread within 0 and 10 for each of the ten dimensions. The variances begin at 1, the weights begin at 1. With different (and usually long) periods, the variance, mean, and weights from each of the Gaussians vary each iteration.

With this configuration, XGBoost and RandomForest algorithms were trained. The first iteration drew 10000 points. 8000 points (80% of the 10000) were used for training, and 2000 (20%) for testing in the first iteration (iteration number 0). From the second iteration onwards (iteration number 2 to 299), the process was the same: the dynamic of the generator would iterate, resulting in a change of distribution; 2000 points were drawn for each iteration; and the models that were trained in the first iteration ran their predictions on the 2000 new points. Since the models are "static", meaning that they do not update in each of the following iterations, they naturally have a performance degradation. It can be seen in figure 5.1, through a Voting Classifier (which combines their votes by a majority rule), that indeed, the performance is gradually lost.

A Voting Classifier was built with these models to make the visualization of the loss of accuracy of the static algorithms over time in figure 5.1.

Table 5.1: Initial accuracy on training and test sets (iteration 0). Two XGBoost Classifiers, with maximum depths of 1 and 2, respectively, and three Random Forests, with maximum depths of 4, 5, and 6, respectively, were created.

Algorithm	*Train*	*Test*
XGBoost 1	0.672	0.648
XGBoost 2	0.844	0.813
Random Forest 1	0.764	0.743
Random Forest 2	0.849	0.830
Random Forest 3	0.914	0.890

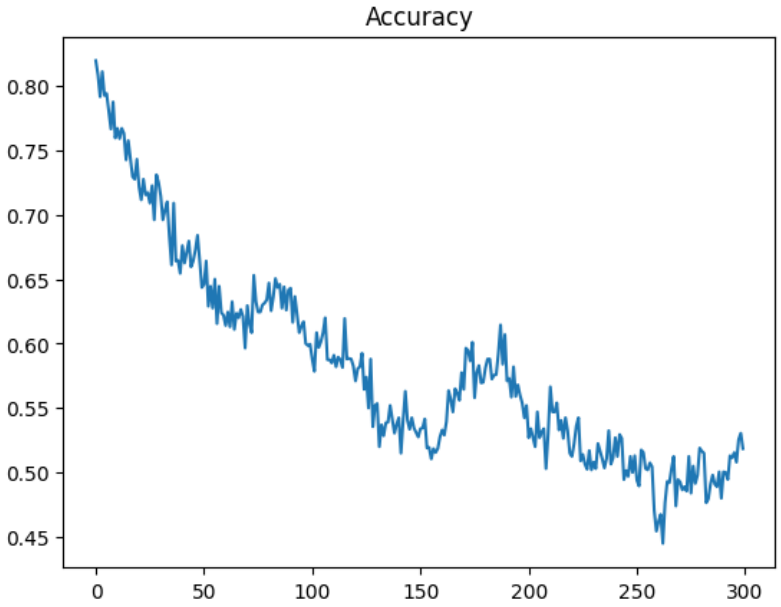
The Author

Table 5.2: Initial Mean Squared Error on training and test sets (iteration 0). The configuration is the same as in table 5.1

Algorithm	*Train*	*Test*
XGBoost 1	0.328	0.352
XGBoost 2	0.156	0.187
Random Forest 1	0.236	0.258
Random Forest 2	0.151	0.171
Random Forest 3	0.086	0.111

The Author

Figure 5.1: Illustration of the loss of performance of the Voting Classifier of the five models over the iterations of the dataset



Source: The Author

Since Harmonic Variables were used for generating this dataset, it is possible to observe that the algorithm's performance oscillates as the sines and cosines of the most influential factors reset to their original setting.

5.1.1 Drift Detection Algorithms

The following experiment shows at which points in time (which iteration) the drift from the previous dataset (using the voting classifier of the previous models) is detected for each of the following detection algorithms:

Table 5.3: Drift Detection Algorithms with 100 points per iteration

Algorithm	*Iteration*
DDM	did not detect
ADWIN	23, 41, 57, 61, 65, ...
EDDM	did not detect
HDDM	64
KSWIN	43, 56, 62, 118, 141, ...
Page-Hinkley	did not detect

The Author

Table 5.4: Drift Detection Algorithms with 1000 points per iteration

Algorithm	*Iteration*
DDM	did not detect
ADWIN	9, 11, 15, 18, 20 ...
EDDM	0 (4 times)
HDDM	166, 210, 237
KSWIN	19, 20, 22, 23, 28, ...
Page-Hinkley	did not detect

The Author

These experiments serve as an illustration of how the datasets generated can be used as a benchmark. Notice that some algorithms perform worse than the others. EDDM, for example, gives four known false positives for drift when using 1000 points per iteration. DDM did not detect drift within 300 iterations (for 100 and 1000 points). Other algorithms accused drift more frequently with more points per iteration than less.

6 CONCLUSION

The complexity of evaluating the efficiency and effectiveness of machine learning algorithms is increased when the assumption of drift is introduced. Most of the time, it is possible to use datasets with real-world data, get a good grasp of what is happening, and make accurate algorithm comparisons when dealing with stationary data. Several synthetic data generators are available for stationary data. However, even though algorithm performance evaluation is arguably more challenging in the presence of drift compared to the static case, there are much fewer data generators that incorporate drift into their implementation. Additional difficulties include that, as machine learning algorithms grow in complexity and simple datasets are no longer as useful for algorithm comparison and research, the few existing generators need more complexity and customizability, which help replicability and analysis.

This work proposed a simulator with these characteristics integrated into its design to address the abovementioned points. Additionally, solutions were developed to make using this simulator and the visualization of the data being generated easier. Implementing a data specification interface for the to-be-generated data aids in replicability and clarity. To improve the experience of the user and facilitate the process of data customization, a GUI and CLI were developed.

Finally, a data benchmark was generated, and a comparison of drift detection algorithms was performed on it. One of the conclusions from the experiment is that there are some limitations to the simplicity of generating complex behavior on multiple dimensions, and more types of drift (such as gradual) could be added. However, with this benchmark, it was possible to discriminate algorithms that detected drift better than others. The code for the generator and experiments is located at <https://github.com/FelipeFuhr/customizable-data-with-drift-generator>.

Although this new generator has favorable characteristics that differentiate it positively from others, future work could evaluate other classification and regression algorithms. Another area for improvement would be to seek heuristics to generate datasets with increasing difficulty levels more intuitively, a practical problem that is complex and discussed in Hutter, Kotthoff and Vanschoren (2020). Another potential prospect would be to find ways to generate and compare synthetic datasets with real datasets to increase the practical utility of this approach. One possible idea would be to find a way to quantify the distance between datasets, done as part of the AutoML algorithm in Feurer et

al. (2015). This way, we could consider generating synthetic datasets with explainable and measurable dynamics that approximate specific real-world problems, test algorithms that exhibit good performance on synthetic data, and finally validate if this performance translates back to real-world problems. Further research would be needed to evaluate the theoretical bounds of models on metrics besides accuracy, such as F1 score.

REFERENCES

ABERNETHY, R. B. **The New Weibull Handbook.** Fifth edition. [S.l.: s.n.], 2004.

AXLER, S. **Linear Algebra Done Right**. [S.l.]: Springer, 2015. ISBN 978-3-319-11080-6.

BAENA-GARCÍA, M. et al. Early drift detection method. In: CONFERENCE PAPER. **Proc. 4th Int. Workshop Knowledge Discovery from Data Streams**. [S.l.], 2006.

BIFET, A.; GAVALDà, R. Learning from time-changing data with adaptive windowing. In: . [S.l.: s.n.], 2007. v. 7.

BISHOP, C. M. **Pattern Recognition and Machine Learning**. [S.l.]: Springer, 2006.

Cloud Architecture Center. **MLOps: Continuous delivery and automation pipelines in machine learning**. 2023. <https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning>. [Online; accessed 24-Aug-2023].

Cloud Native Computing Foundation. **Kubernetes**. 2021. Available from Internet: <https://kubernetes.io/>.

CYBENKO, G. Approximation by superpositions of a sigmoidal function. **Mathematics of Control, Signals, and Systems**, v. 2, n. 4, p. 303–314, 1989. Available from Internet: <https://doi.org/10.1007/BF02551274>.

DIETTERICH, T. G. Approximate statistical tests for comparing supervised classification learning algorithms. **Neural Computation**, v. 10, n. 7, p. 1895–1923, 1998.

EBDEN, M. **Gaussian Processes: A Quick Introduction**. 2015.

FEURER, M. et al. Efficient and robust automated machine learning. In: **Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2**. Cambridge, MA, USA: MIT Press, 2015. (NIPS'15), p. 2755–2763.

FRIAS-BLANCO, I. et al. Online and non-parametric drift detection methods based on hoeffding's bounds. **IEEE Trans. Knowl. Data Eng.**, v. 27, n. 3, p. 810–823, 2015.

GAMA, J. et al. Learning with drift detection. In: **Advances in Artificial Intelligence – SBIA 2004**. Berlin, Heidelberg: Springer, 2004. (Lecture Notes in Computer Science, v. 3171). Available from Internet: <https://doi.org/10.1007/978-3-540-28645-5_29>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. ISBN 978-0262035613.

HOGG, R. V.; TANIS, E. A.; ZIMMERMAN, D. L. **Probability and Statistical Inference**. 9th. ed. [S.l.]: Pearson, 2019.

HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. **Automated Machine Learning**. [S.l.]: Springer, 2020.

KHAMASSI, I. et al. Discussion and review on evolving data streams and concept drift adapting. **Evolving Systems**, v. 9, p. 1–23, 2018. Available from Internet: <https://doi.org/10.1007/s12530-016-9168-2>.

KIDGER, P.; LYONS, T. **Universal Approximation with Deep Narrow Networks**. 2020.

KIM, G. et al. **The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations**. [S.l.]: IT Revolution Press, 2016. ISBN 978-1942788003.

KREUZBERGER, N. K. D.; HIRSCHL, S. **Machine Learning Operations (MLOps): Overview, Definition, and Architecture**. 2022.

LOBO, J. L. **Synthetic datasets for concept drift detection purposes**. Harvard Dataverse, 2020. Available from Internet: <https://doi.org/10.7910/DVN/5OWRGB>.

LU A. LIU, F. D. F. G. J. G. J.; ZHANG, G. Learning under concept drift: A review. **IEEE Transactions on Knowledge and Data Engineering**, Institute of Electrical and Electronics Engineers (IEEE), p. 1–1, 2018. Available from Internet: <https://doi.org/10.1109%2Ftkde.2018.2876857>.

LU, N.; ZHANG, G.; LU, J. Concept drift detection via competence models. **Artificial Intelligence**, v. 209, p. 11–28, 2014.

MONTGOMERY, D. C. **Design and Analysis of Experiments**. 7th. ed. [S.l.]: John Wiley & Sons, 2009.

MONTGOMERY, D. C.; RUNGER, G. C. **Applied Statistics and Probability for Engineers**. 3rd. ed. [S.l.]: Wiley, 2002.

NADEAU, C.; BENGIO, Y. Inference for the generalization error. **Machine Learning**, v. 52, p. 239–281, 2003. Available from Internet: <https://doi.org/10.1023/A:1024068626366>.

NORVIG, P.; RUSSELL, S. **Artificial Intelligence: A Modern Approach**. 4th. ed. [S.l.]: Pearson, 2020. ISBN 978-0134610993.

PAGE, E. S. CONTINUOUS INSPECTION SCHEMES. **Biometrika**, v. 41, n. 1-2, p. 100–115, 06 1954. ISSN 0006-3444. Available from Internet: <https://doi.org/10.1093/biomet/41.1-2.100>.

POSOLDOVA, A. Machine learning pipelines: From research to production. **IEEE Potentials**, v. 39, n. 6, p. 38–42, 2020.

RAAB, C.; HEUSINGER, M.; SCHLEIF, F.-M. Reactive soft prototype computing for concept drift streams. **Neurocomputing**, Elsevier BV, v. 416, p. 340–351, nov 2020. Available from Internet: <https://doi.org/10.1016%2Fj.neucom.2019.11.111>.

RAMíREZ-GALLEGO, S. et al. A survey on data preprocessing for data stream mining: Current status and future directions. **Neurocomputing**, v. 239, p. 39–57, 2017.

RUDIN, C. **Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead**. 2019.

SANTOS, M. S. et al. Cross-validation for imbalanced datasets: Avoiding overoptimistic and overfitting approaches [research frontier]. **IEEE Computational Intelligence Magazine**, v. 13, n. 4, p. 59–76, 2018.

Song Qiao Hu. **Concept Drift Datasets v1.0**. 2023. <https://github.com/songqiaohu/THU-Concept-Drift-Datasets-v1.0>. [Online; accessed 24-Aug-2023].

STUDENT. The probable error of a mean. **Biometrika**, v. 6, n. 1, p. 1–25, 1908. Available from Internet: <https://doi.org/10.2307/2331554>.

SYMEONIDIS E. NERANTZIS, A. K. G.; PAPAKOSTAS, G. A. **MLOps – Definitions, Tools and Challenges**. 2022.

THORNTON, C. et al. Auto-weka: Automated selection and hyper-parameter optimization of classification algorithms. **CoRR**, abs/1208.3719, 2012. Available from Internet: <http://arxiv.org/abs/1208.3719>.

TREVEIL, M.; TEAM, D. **Introducing MLOps: How to Scale Machine Learning in the Enterprise**. [S.l.]: O'Reilly, 2020.

WEIBULL, W. **A Statistical Distribution Function of Wide Applicability**. [S.l.]: Journal of Applied Mechanics, 18, 293-297, 1951.

WIDMER, G.; KUBAT, M. Learning in the presence of concept drift and hidden contexts. **Machine Learning**, Springer, v. 23, p. 69–101, 1996. Available from Internet: <https://doi.org/10.1007/BF00116900>.

APPENDIX A - KUBERNETES

The structure in which datasets and other entities can be specified in the proposed simulator are based on how Kubernetes objects are organized. These objects usually are organized declaratively in a well-defined nested structure with YAML files.

Kubernetes (Cloud Native Computing Foundation, 2021), or K8s, is an open-source tool for automating the deployment of containerized applications. It is a DevOps tool that is highly customizable and scalable.

APPENDIX B - AUTOML

The problem of automating Machine Learning pipelines is related to one of the related work's data generator. Automated Machine Learning (AutoML) also inspired the desire to measure the drift more objectively since having an objective measure of drift is a prerequisite to automating how to deal with drift adequately.

As formally defined in Feurer et al. (2015):

Definition 6.1. *For $n, m \in \mathbb{N}^+$, $i = 1, ..., n + m$, let $x_i \in \mathbb{R}^d$ denote a feature vector of d dimensions and $y_i \in Y$ the corresponding target value. Given a training dataset $D_{train} = (x_1, y_1), ..., (x_n, y_n)$ and the feature vectors $x_{n+1}, ..., x_{n+m}$ of a test dataset $D_{test} = (x_{n+1}, y_{n+1}), ..., (x_{n+m}, y_{n+m})$ drawn from the same underlying data distribution, as well as a resource budget b and a loss metric $\mathcal{L}(.,.)$, the **AutoML** problem is to (automatically) produce test set predictions $y_{n+1}, ..., y_{n+m}$. The loss of a solution $\hat{y}_{n+1}, ..., \hat{y}_{n+m}$ to the AutoML problem is given by:*

$$\frac{1}{m} \sum_{j=1}^{m} \mathcal{L}(\hat{y}_{n+j}, y_{n+j})$$

It can also be reduced to a Combined Algorithm and Hyper-parameter Selection (CASH) problem, as defined in Thornton et al. (2012):

Definition 6.2. *Given a set of algorithms $\mathcal{A} = \{A^{(1)}, ..., A^{(k)}\}$ with associated hyperparameter domains $\Lambda^{(1)}, ..., \Lambda^{(k)}$ and a loss metric $\mathcal{L}(.,.)$, we define the **CASH** problem as computing:*

$$A^*_{\lambda^*} \in \underset{A^{(j)} \in \mathcal{A}, \lambda \in \Lambda^{(j)}}{\operatorname{argmin}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A^{(j)}_{\lambda}, \mathcal{D}^{(i)}_{train}, \mathcal{D}^{(i)}_{valid})$$

APPENDIX C - F1 SCORE

A famous score that is closely related to drift is the F1 score. It is another popular metric for evaluating model performance. It is handy when dealing with imbalanced datasets, since it considers the model's precision (accuracy) and its recall. This work focuses on accuracy, but it may be interesting to study the possibility of exploring the theoretical boundaries of the F1 metric. The F1 score can be defined as:

$$F_1 = 2 * \frac{p * r}{p + r}$$

In the above formula, precision (or p) is the ratio of true positive predictions and total positive predictions, and recall (or r) is the ratio of true positive predictions and total actual positive instances.